Context-aware Process Management
for the Software Engineering Domain

Dissertation zur Erlangung des Doktorgrades Dr. rer. nat.
der Fakultät für Ingenieurwissenschaften, Informatik und Psychologie der Universität Ulm

Vorgelegt von:
Gregor Robert Grambow
geboren in Schwäbisch Hall

2016
Amtierende Dekanin: Professor Tina Seufert
Gutachter: Professor Manfred Reichert
Professor Franz Schweiggert
Professor Roy Oberhauser
Tag der Promotion: 13.05.2016
The results presented in this thesis are the outcome of my work as a research assistant at the Institute of Databases and Information Systems of Ulm University between 2012-2015 with Prof. Dr. Manfred Reichert, as well as at the Computer Science Department of Aalen University of Applied Sciences between 2009-2012 working with Prof. Roy Oberhauser on the Q-ADVICE project (Quality ADVisory Infrastructure for Collaborative Engineering), where we created the Context-aware Software Engineering Environment Event-driven frameworK (CoSEEEK).

I am very grateful to my mentor Prof. Dr. Manfred Reichert, whom I thank for his helpful advice, continuous support, and valuable feedback during the last years. He provided all the freedom needed for my research, but at the same time was always there to support me.

In addition, I am deeply grateful to my mentor Prof. Roy Oberhauser. This thesis would not have been possible without his initiative and extensive help and support, and I thank him for being a part of it. I could go to him with any problem or worry. He was a motivation and inspiration to me: not only his passion for science, but also his passion to really improve the situation for practitioners in software engineering, as well as his creativity and innovative ideas.

Besides my mentors, I would like to thank my second proofreader Prof. Dr. Franz Schweiggert and the members of my doctoral committee Prof. Dr. Peter Dadam, Prof. Dr. Helmuth Partsch, and Prof. Dr. Uwe Schöning for providing feedback for my thesis.

My thanks extend to my colleagues and students from the Q-ADVICE project for their support and help with the implementation of our concepts: Stefan Lorenz, Andreas Kleiner, Muhammer Tüfekci, Andreas Nägeli, and Alexander Grünwald.

I would also like to thank my colleagues and friends from Ulm University. It was a great time at DBIS, sharing so many different tasks with you.

Personally, I also want to thank Susanne for her help and continuous support in every situation. Her care and support during most of my thesis time made this the best time of my life, despite my spending so much time and attention on this project. She stood by me, and I am really looking forward to our future together.

Last but not least, I would like to thank my parents Heiderose and Peter Grambow. They were always there for me during the past decades, and always supported me with good advice in every situation. They enabled my studies in Aalen, Karlsruhe, and Ulm. For all of these things, and many more, I am deeply thankful.
Abstract

Historically, software development projects are challenged with problems concerning budgets, deadlines and the quality of the produced software. Such problems have various causes like the high number of unplanned activities and the operational dynamics present in this domain. Most activities are knowledge-intensive and require collaboration of various actors. Additionally, the produced software is intangible and therefore difficult to measure. Thus, software producers are often insufficiently aware of the state of their source code, while suitable software quality measures are often applied too late in the project lifecycle, if at all.

Software development processes are used by the majority of software companies to ensure the quality and reproducibility of their development endeavors. Typically, these processes are abstractly defined utilizing process models. However, they still need to be interpreted by individuals and be manually executed, resulting in governance and compliance issues. The environment is sufficiently dynamic that unforeseen situations can occur due to various events, leading to potential aberrations and process governance issues. Furthermore, as process models are implemented manually without automation support, they impose additional work for the executing humans. Their advantages often remain hidden as aligning the planned process with reality is cumbersome.

In response to these problems, this thesis contributes the Context-aware Process Management (CPM) framework. The latter enables holistic and automated support for software engineering projects and their processes. In particular, it provides concepts for extending process management technology to support software engineering process models in their entirety. Furthermore, CPM contributes an approach to integrate the enactment of the process models better with the real-world process by introducing a set of contextual extensions. Various events occurring in the course of the projects can be utilized to improve process support and activities outside the realm of the process models can be covered. That way, the continuously growing divide between the plan and reality that often occurs in software engineering projects can be avoided. Finally, the CPM framework comprises facilities to better connect the software engineering process with other important aspects and areas of software engineering projects. This includes automated process-oriented support for software quality management or software engineering knowledge management. The CPM framework has been validated by a prototypical implementation, various sophisticated scenarios, and its practical application at two software companies.
The presented research was performed in the Q-ADVICE project. Goal of this project was to achieve holistic support for software engineering projects and processes. The project was funded by the German Federal Ministry of Education and Research (BMBF) under grant number 17N4809. More information can be found at https://www.hs-aalen.de/en/users/121/seiten/q-advice-research-project.

The research conducted within this thesis has led to the publication of various articles listed in the following.

**SE Process Support, Context Integration, Process Dynamicity, and Quality Assurance**


**Extrinsic Process Coverage**


**Collaboration and Coordination**


**Process Exception Handling**
Knowledge Provisioning

Application to the Software Modernization Domain
The topic described in this publication describes the application of the concepts of this thesis to a software modernization process.

SE Process Modeling
The topic described in this publication is an extension to the concepts of this thesis for integrated SE process modeling and enactment.

SE Process Assessment
The topic described in these publications is an extension to the concepts of this thesis that establishes a connection to SE process assessment models like CMMI or ISO 15504.
# Contents

Part I  Problem Statement and Requirements ................................................................. 1  

1.  Introduction .................................................................................................................... 3  
   1.1. Problem Statement ................................................................................................... 5  
   1.2. Contribution .............................................................................................................. 6  
   1.3. Outline ..................................................................................................................... 7  

2.  Research Methodology .................................................................................................... 9  
   2.1. Research Questions .................................................................................................. 9  
   2.2. Information Systems Research ................................................................................ 9  

3.  Background ...................................................................................................................... 13  
   3.1. The Software Engineering Process ......................................................................... 13  
   3.2. Software Engineering Process Models .................................................................... 15  
      3.2.1. Classical Approaches ...................................................................................... 15  
      3.2.2. Agile Approaches ........................................................................................... 17  
   3.3. Summary ................................................................................................................ 20  

4.  Requirement Analysis ..................................................................................................... 21  
   4.1. Concrete Problems .................................................................................................. 21  
   4.2. Basic Requirements ................................................................................................ 28  
   4.3. Requirements Verification ....................................................................................... 29  
   4.4. Summary ................................................................................................................ 31  

Part II  Solution ................................................................................................................ 33  

5.  Foundations .................................................................................................................... 35  
   5.1. Process Modeling ..................................................................................................... 35  
   5.2. Basic Terminology and Premises .......................................................................... 37  
   5.3. Basic Definitions ..................................................................................................... 38  
   5.4. Correctness ............................................................................................................. 40  
   5.5. Types of Workflows ............................................................................................... 41

6.1. Requirements .................................................................................................................. 43
6.2. Framework Components ............................................................................................ 45
6.3. Discussion ..................................................................................................................... 53
   6.3.1. Computer-Aided Software Engineering .............................................................. 53
   6.3.2. Process-centered Software Engineering Environments ..................................... 54
   6.3.3. Modern Development Environments .................................................................... 56
   6.3.4. Other Contemporary Approaches ........................................................................ 59
   6.3.5. Related Work Summary ....................................................................................... 62
6.4. Summary ....................................................................................................................... 63

7. Contextual Extensions for Software Engineering Processes .................... 65

7.1. Requirements .............................................................................................................. 66
7.2. Contextual Software Engineering Process Extensions ............................................. 69
7.3. Software Engineering Workflow Governance ......................................................... 73
   7.3.1. Horizontal Governance ...................................................................................... 74
   7.3.2. Vertical Governance .......................................................................................... 76
7.4. Extended Software Engineering Activity Modeling ............................................... 78
7.5. Abstraction from Internal Workflow Logic .............................................................. 81
7.6. Automated Software Engineering Process Adaptation ............................................ 86
7.7. Conceptual Framework .............................................................................................. 89
   7.7.1. Basic Concepts ..................................................................................................... 89
   7.7.2. Consistency Checks ............................................................................................ 99
   7.7.3. Algorithms for Marking Workflows ................................................................. 100
   7.7.4. Basic Actions for Software Engineering Process Enactment .......................... 108
7.8. Discussion .................................................................................................................. 110
   7.8.1. Process Enactment Support .............................................................................. 110
   7.8.2. Dynamic Processes ........................................................................................... 110
   7.8.3. Contextual Process Support / Integration ....................................................... 113
   7.8.4. Related Work Summary ................................................................................... 118
7.9. Summary ................................................................................................................... 119

8. Extended Software Engineering Process Coverage ................................. 121

8.1. Requirements ............................................................................................................. 122
8.2. Hybrid Workflow Approach .................................................................................... 125
   8.2.1. Different Activity Types of Software Engineering Workflows ....................... 125
   8.2.2. Extrinsic Workflow Modeling and Enactment .................................................... 126
   8.2.3. Applying Situational Method Engineering ......................................................... 128
8.2.4. Information Gathering ................................................................. 129
8.2.5. Declarative Workflow Modeling .................................................. 130
8.2.6. Treatment of Different Workflow Types ........................................ 141
8.2.7. Concrete Procedure for Extrinsic Workflow Enactment ...................... 142
8.2.8. Modeling Effort ........................................................................... 146
8.3. Discussion .................................................................................... 146
  8.3.1. Declarative Process Models ......................................................... 147
  8.3.2. Process Model Configuration ...................................................... 147
8.4. Summary ....................................................................................... 148

   Processes ............................................................................................. 149
  9.1. Requirements ................................................................................ 150
  9.2. Quality Management Integration Approach ..................................... 152
      9.2.1. Solution Procedure .................................................................. 152
      9.2.2. Context Detection ................................................................... 154
      9.2.3. Quality Measure Processing ..................................................... 158
      9.2.4. Quality Post-Processing ........................................................... 170
      9.2.5. Conceptual Framework ............................................................. 171
  9.3. Discussion .................................................................................... 173
      9.3.1. Metric Application .................................................................. 174
      9.3.2. Measurement Tools .................................................................. 174
      9.3.3. GQM support ......................................................................... 174
  9.4. Summary ....................................................................................... 175

10. Workflow Coordination in Software Engineering Processes ................. 177
  10.1. Requirements ............................................................................... 178
  10.2. Automatic Activity Coordination .................................................. 179
      10.2.1. Passive Coordination Support ................................................ 179
      10.2.2. Active Coordination Support .................................................. 182
  10.3. Discussion .................................................................................... 186
  10.4. Summary ....................................................................................... 187

11. Exception Handling in Software Engineering Processes ....................... 189
  11.1. Requirements ............................................................................... 190
  11.2. Flexible Software Engineering Exception Handling ........................ 191
      11.2.1. Abstract Approach ................................................................. 191
      11.2.2. Conceptual Framework ........................................................... 193
      11.2.3. Concrete Procedure ................................................................. 194
  11.3. Discussion .................................................................................... 196
11.4. Summary ................................................................................................................................. 197

12. Knowledge Management Support in Software Engineering Processes .... 199
   12.1. Requirements ....................................................................................................................... 199
   12.2. Software Engineering Knowledge Management Approach ........................................... 201
      12.2.1. Basics for Enabling Software Engineering Knowledge Management ...................... 201
      12.2.2. Software Engineering Knowledge Management Specifics ...................................... 203
      12.2.3. Process-centered Knowledge Support ........................................................................ 204
      12.2.4. Software Engineering Knowledge Provisioning Procedure ...................................... 207
   12.3. Discussion ............................................................................................................................. 210
   12.4. Summary ............................................................................................................................. 211

Part III Evaluation ......................................................................................................................... 213

13. Technical Feasibility .................................................................................................................. 215
   13.1. Requirements ....................................................................................................................... 215
      13.1.1. Functional Requirements ............................................................................................ 215
      13.1.2. Technical Requirements ............................................................................................... 216
   13.2. Extending an Existing Architecture ..................................................................................... 216
      13.2.1. Design and Architecture Decisions .............................................................................. 216
      13.2.2. CPM Implementation .................................................................................................... 217
   13.3. Software Engineering Process Enactment with CPM ....................................................... 219
      13.3.1. Technical Aspects ........................................................................................................... 219
      13.3.2. User Interfaces .............................................................................................................. 220
   13.4. Software Engineering Workflow Adaptation Aspects ....................................................... 222
   13.5. Declarative Software Engineering Workflow Generation ............................................... 223
      13.5.1. Technical Aspects .......................................................................................................... 223
      13.5.2. User Interfaces .............................................................................................................. 228
   13.6. Software Engineering Coordination Aspects ....................................................................... 230
      13.6.1. Technical Aspects .......................................................................................................... 230
      13.6.2. User Interfaces .............................................................................................................. 231
   13.7. Software Engineering Exception Handling Aspects ........................................................... 233
      13.7.1. Technical Aspects .......................................................................................................... 233
      13.7.2. User Interfaces .............................................................................................................. 233
   13.8. Software Engineering Quality Management Aspects ....................................................... 234
   13.9. Software Engineering Knowledge Management Support .............................................. 235
   13.10. Summary ............................................................................................................................. 238

14. Practical Application .................................................................................................................... 239
14.1. Modeling the OpenUP Process ................................................................. 239
  14.1.1. Mapping the Process Concepts .......................................................... 239
  14.1.2. Process Model Enactment ................................................................. 240
14.2. Modeling the V-Model XT Process ............................................................ 242
  14.2.1. Mapping the Process Concepts .......................................................... 242
  14.2.2. Process Model Enactment ................................................................. 243
14.3. Modeling of the Scrum Process ................................................................. 245
  14.3.1. Mapping of the Process Concepts ....................................................... 245
  14.3.2. Process Model Enactment ................................................................. 246
14.4. Extended Process Coverage Scenario ...................................................... 247
  14.4.1. Bug Fixing Use Case ......................................................................... 247
  14.4.2. Further Use Cases .............................................................................. 249
14.5. Automated Quality Management Scenario .............................................. 250
  14.5.1. Process ............................................................................................... 250
  14.5.2. GQM Plan ......................................................................................... 251
  14.5.3. Concrete Situation ............................................................................ 252
14.6. Exception Handling Scenario .................................................................... 255
14.7. Knowledge Management Support Scenario ............................................ 256
14.8. Workflow Coordination Scenario .............................................................. 258
14.9. Sample Application: Software Modernization ........................................... 259
14.10. Lessons Learned from a Preliminary Industrial Application ................. 261
14.11. Summary ............................................................................................... 264

15. Discussion .................................................................................................... 265
  15.1. Enabling Comprehensive Software Engineering Process Support ........... 265
  15.2. Related Approaches ............................................................................... 266
  15.3. Problem Areas ....................................................................................... 268
  15.4. Overall Comparison ............................................................................... 270
  15.5. Threats to Validity ................................................................................. 271
  15.6. Major Findings ...................................................................................... 273
  15.7. Summary ............................................................................................... 274

Part IV Conclusion ............................................................................................ 275

16. Summary and Outlook ................................................................................... 277

Bibliography ....................................................................................................... 281

Acronyms ........................................................................................................ 303
Part V  Appendices ........................................................................................................... 305

A. Ontology ......................................................................................................................... 307
   A.1. Imperative Process Concepts .................................................................................. 307
      A.1.1. Template Concepts ......................................................................................... 307
      A.1.2. Individual Concepts ....................................................................................... 310
   A.2. Declarative Process Concepts .................................................................................. 313

B. Conceptual Framework .................................................................................................... 317
   B.1. Entity Concepts ....................................................................................................... 317
      B.1.1. Basic Concepts ............................................................................................... 317
   B.2. Consistency Checks .................................................................................................. 323
      B.2.1. Basic Concepts ............................................................................................... 323
      B.2.2. Extrinsic Workflows ....................................................................................... 326
      B.2.3. Quality Management ....................................................................................... 330
   B.3. Algorithms ................................................................................................................ 331
      B.3.1. Basic Workflow Enactment .............................................................................. 331
      B.3.2. Extrinsic Workflow Generation ....................................................................... 337

C. Basic Actions for Process Enactment .............................................................................. 339
Part I

Problem Statement and Requirements
1. Introduction

Software Engineering (SE) is a discipline that implies special properties for process enactment. On one hand, these are correlated with the special properties of the produced product, i.e., the software: complexity, conformity, changeability, and invisibility [Broo87]. On the other, IT support for SE processes is not mature yet, since SE implies a highly dynamic and creative process. Furthermore, the impact of process management in SE has been underestimated for a long time [Wall07]. Over decades, many SE process models as well as models for SE process improvement have been developed and been introduced to practice. In other areas, like industrial production, such processes have been automated and supported by process management technology [LeRo00]. Yet implementation and automated enactment of processes is not prevalent in SE, mostly due to the dynamic nature of these knowledge-intensive processes that contradict the rigid sequencing of process activities necessary for an automated enactment.

SE processes are essentially knowledge-intensive, i.e., they depend on knowledge workers to a large extent [KeHa02]. The highly intellectual SE process implies a high amount of communication. Compared to industrial production processes, SE processes rely much more on humans and highly collaborative team interactions. Note that each SE project constitutes a development project, producing a unique outcome. For such projects, the rigidity of prescribed processes mostly does not fit. Further, it was already stated that dynamic processes supporting collaboration as well as communication can be beneficial [Shet97]. Usually, SE processes deal with the development of a new product (i.e., the software), which is a knowledge-intensive task [RaTi99]. In this context, necessary facts, much information and comprehensive knowledge are handled manually and implicitly by the humans involved. Hence, automation is not feasible and SE processes are usually performed manually in a documentation-centric way [RBTK05]. In turn, this often implies high manual efforts for humans as they have to manage the process models. Moreover, actual process enactment largely depends on humans. Many tasks and activities are not part of the process models. Especially on the operational level, where activities like coding and testing are performed, only limited support for the software engineers is available from SE process models. Thus, there is a growing gap between the specified process and the one actually executed.

Another specialty of SE projects is the product developed. Software has special properties whose combination differentiates it from many other products: complexity, conformity, changeability, and invisibility [Broo87]. These properties make it difficult to be aware of the status of the software. In many software producing companies, human tasks, requirements, and the realization of the requirements are managed in some way. However, due to the often high number of humans working concurrently on numerous source code artifacts, the quality of the source code can deteriorate unnoticed. Thus, many projects struggle with bad source code quality [Jone10]. However, to be enforced, software quality must be defined and measured [Kan02]. In many cases, resources are wasted by neglecting software quality issues and respective software quality measures until the final stages of a project [Hami88, SHK98]. Another issue of software quality comes with the lacking ability of many companies to actually control, manage and support their knowledge-intensive and human-centric processes [BDS+99, Ambl02, Wall07, Dust04, SBBK08]. Due to these issues, projects suffer from bad software quality as well as exceeded budgets and deadlines having issues on both the process and the product side. Altogether, it is desirable to integrate software quality assurance tightly and smoothly with process management to enable the continuous monitoring of product quality.

Business Process Management and the introduction of Process-Aware Information Systems (PAIS) has been a continuous trend in various business areas. In particular, the explicit governance of activities by a PAIS enables improved repeatability of the process and can thus improve the quality of the product [ReWe12, DAH05]. Domains in which PAIS have been successfully introduced include
health care [LeRe07], automotive engineering [MHHR06, MRH08], finance [GoAk03] and transportation [Bass05]. To be able to comprehensively cover all activities executed as well as optimize the whole process executed in an organization, the business process lifecycle [GeTs98, vdAa04, WRWR09] is roughly separated into several phases (cf. Figure 1-1): First the process is defined, which implies a design process. Further, this phase might include the discovery of executed processes through process mining [vdAa11, vdWe04, vWM04]. Following the design phase, the process is implemented. In the subsequent enactment phase, the process is used to govern the activities it was designed for. Data from this enactment phase is then used for the diagnosis phase, which can be applied to optimize the process as well as to adapt it to environmental changes. With the results from the diagnosis phase the cycle can be restarted.

![Process Lifecycle Diagram](image)

**Figure 1-1: Process lifecycle (adopted from [vdAa04])**

To enable continuous support and guidance for a process, automated IT support is desirable. To achieve the latter, processes can be implemented using PAIS [vdvH02]. Such systems provide support for automated process enactment, automated task distribution to humans, coordination, and monitoring of different process instances. That way, process enactment can be guided and process diagnosis be supported, since the executed activities are explicitly governed. This makes the entire process enactment more traceable and repeatable.

The described factors hamper successful process enactment in the SE domain. Many of the issues discussed, however, are related to the dynamics of SE projects [BDS+99, Ambi02]. In fact, numerous obstacles inhibit automated SE process management (SEPM) at the operational level. These include a high number of dynamically executed small tasks like bug fixing, coding, developer tests, or integration tests. Respective tasks may not even be covered on the more abstract planning levels where the entire project, its process, and different phases are managed. Such activities also imply many contextual dependencies, i.e., they rely on properties relating to the current situation, e.g., time pressure in the projects or technology used. Another factor is the great number of involved artifacts, e.g., documentation artifacts, specifications, or the source code itself. These artifacts often have many relations with each other and are frequently changed by various persons. This involves a great amount of tacit knowledge crucial to the projects that is only implicitly managed by these persons. In turn, this puts high pressure on them: Because of the high dynamicty, the concurrent enactment of multiple projects, the absence of clearly defined and stable requirements, and many other factors, much is left to them. This constitutes a great burden as well as high efforts for software engineers. Due to the lack of repeatability and guidance of these knowledge processes, it is rather likely that the knowledge worker forgets important tasks or unintentionally introduces new problems to the source code.
1.1. Problem Statement

As shown by many studies, SE projects have been suffering from problems with exceeded budgets, missed schedules, and low product quality for a long time [NaRa68, Broo87, Glas98, Kruc04, Jone10]. Many of these problems are resulting from the adolescence of SE as a discipline and special properties of this discipline having a great impact on SE projects. These properties (e.g., the intangible product or the knowledge-intensive, human-centric SE process) are exhibited, in both the created product and in the SE process. Based on this, three main topics introducing serious issues to SE can be observed: First, the knowledge-intensive process puts much pressure on the involved humans. Second, the intangible product makes it difficult to control the latter and might introduce severe quality issues. Looking at these two problematic sides of SE projects, a third topic comes into mind: there exist tools that support various SE aspects but no comprehensive and automatic process support, incorporating humans and artifacts, is prevalent.

Note that we will use the term process and workflow with different meanings. Process will be referred to as something rather abstract that is not implemented in software. Workflow will be referred to as something more concrete and operational as well as something that is implemented using a software tool and, therefore, as an automated facility to govern the flow of activities.

**Manual process implementation.** Process automation was mainly applied in areas in which foreknown activity sequences exist, but not in scenarios requiring the enactment of a human-centric and knowledge-intensive process [MBR15]. In SE, therefore, there exists not much experience with process automation. Process models are available containing information important for the projects [BWHW06, RiJa00, Mall09]. However, these remain rather abstract and prescriptive [BDS+99, Ambi02]. Hence, manual implementation becomes necessary. Consequently, the involved persons are responsible for enacting the SE process without automated governance or enforcement. This implies shortcomings with respect to guidance, traceability, monitoring, and diagnosis of the activities executed, as abstract process models mostly do not reach the actual executing persons [Wall07]. In particular, they tend to fail in providing operational guidance. Since the quality of the software product is depending on the quality of the SE process [Wall07], this affects product quality as well. The gap between the abstract process models and the actual executed activities also prevents comprehensive coverage of all activities in the SE process. Many activities are executed ad-hoc and cannot be traced. However, if many are activities executed outside the SE process, knowledge about actual process enactment cannot be established. In turn, this makes it difficult to enable reproducibility of processes and projects or the process improvement measures applied.

**Knowledge-intensive processes.** The issue with lacking process automation is epitomized by the fact that the process is both complex and knowledge-intensive. As stated, SE processes involve new product development, which is a knowledge-intensive task [RaT99]. Even if not dealing with product development, SE processes are mostly knowledge-intensive [KeHa02]. There is a need for capturing and sharing various types of information, including domain knowledge, knowledge about technologies, or knowledge about national or local policies [LiRu02]. Supporting this with an automated tool can be beneficial [TFB00]. Often, Wikis are used for SE knowledge management since they can be easily created and information can be quickly accessed [SBBK08]. However, retrieving contextually relevant information from Wikis is a difficult task [SBBK08]. Thus, knowledge management as well as knowledge transfer is hampered. However, SE is essentially a collaborative activity [JYW07, CoCh06]. Consequently, the other side of knowledge-intensive processes concerns the collaborations of the various individuals working in these processes [MBR15, MuRe14]. The different connections between humans, teams, tools, and artifacts are of crucial importance for SE success [JYW07, SQTR07]. However, efficient communication and process-aware collaboration remain a great challenge [Dust04], and, to a large extend, team work remains unpredictable and unplannable [BSV07]. Moreover, collaborative work in SE is still not adequately supported by tooling in SE [LeBo07].
Product quality issues. The created product – the software – has specific properties making it difficult to monitor and control its status. In turn, this complicates SE projects. Software Quality Assurance (SQA) has proven to be essential for SE. In particular, it has been shown that SQA has impact on project costs [Hami88, KKKM00, HuBo06, MSG13], which makes effective and efficient SQA mandatory. Effective application of software measurement remains a big challenge for software vendors [STT06]. Furthermore, software quality measures are often applied too late in the projects, although it has been proven that their application in earlier stages could save time and money [Hami88, SHK98]. Note that the application of quality measures is also problematic, since their effectiveness as well as efficiency depend on various factors, like the applicability of the measure, the project timing, worker competency, or correct execution of the measure [Hami88].

1.2. Contribution

This work originated from the Q-ADVICE (Quality ADvisory Infrastructure for Cooperative Engineering) project, whose goal of this project was the creation of a concept as well as a prototypical framework supporting the SE process. That concept as well as the framework shall enable the automation of various supportive aspects enhancing the quality of the SE process as well as its product. In Chapter 13, various aspects regarding the technical implementation of a prototype framework are discussed. All chapters before that deal with the abstract approach that extends process management technology to enable holistic support for SE projects taking into account the different aforementioned problem areas. We call this approach CPM (Context-aware Process Management). Its core contributions are aligned to the core problems identified:

- **SE process model implementation support:** CPM supports the implementation of entire SE process models. It provides facilities to enable automated process enactment in SE projects. This includes support for all process levels ranging from abstract processes to the concretely executed workflows. Further, CPM provides facilities to integrate process enactment directly with the project environment. Thus, a connection between the abstract process models and the concrete activities on the operational level is established. Context information is automatically collected and utilized for various purposes.

- **Advanced SE process enactment:** CPM also integrates advanced process enactment features to support dynamic domains as SE. It features dynamic processes, i.e., predefined processes may be dynamically adapted to match different situations. Furthermore, CPM enables context-sensitive adaptations, i.e., automatically collected context data is utilized to adapt running processes to the needs of the current situation. Further, CPM features facilities to model and execute dynamic workflows that are usually not covered by SE process models. Thus, these workflows can be integrated with standard process enactment, and, hence, can be guided, traced, and can profit from other CPM functionalities. Finally, CPM incorporates advanced facilities for exception handling. These take various types of context knowledge into account, as, for example, the states of activities, artifacts, or the SE process. Furthermore, CPM enables flexibility and automation for handling exceptions and is capable of not only automatically determining the right exception handling, but also the right person and time point for applying the handling.

- **Integration of processes with other areas of SE projects:** CPM integrates process enactment with other areas important for SE projects. One of these is quality assurance. This includes the automatic detection of potential problems in source code artifacts as well as the management of quality goals, proactive quality measures and reactive quality measures. Furthermore, quality measures can be prioritized according to quality goals and be automatically and context-sensitively distributed to the executing persons in alignment with their standard process activities. Another important area is SE knowledge management. CPM enables automatic management of collected SE knowledge utilizing machine-readable semantics. That way, the context-sensitive selection of applicable knowledge for SE engineers becomes possible. Furthermore, that knowledge can be automatically injected into the running process to support SE engineers in various situations. Finally, CPM supports collaboration in SE projects. It features various types of meta information that allow automatically recognizing coherences between different activities and artifacts even if they are executed in different areas or departments of a project and by different
persons. With this information, different types of automated coordination become possible ranging from simple information distribution to fully automated creation and distribution of new activities.

This work provides an evaluation of the developed concepts as well. By implementing a prototypical framework, questions regarding the technical feasibility of the approach are dealt with. Furthermore, detailed studies demonstrating the applicability of the approach were conducted and the framework was applied in two practical settings.

1.3. Outline

This thesis is split into five parts:

**Part I (Problem statement and requirements)** provides the motivation of holistic process and project support for SE. In Chapter 2 research question and research methodology are described. Chapter 3 provides background information on the SE domain and SE processes, whereas Chapter 4 elicits basic requirements for a tool providing automated holistic support in this domain.

**Part II (Solution)** is devoted to the solution. It starts with Chapter 5 providing basic information needed for understanding the work. In Chapter 6, the abstract solution approach is described. Then, Chapter 7 discusses the contextual extensions to process management concepts being the basis for all other components of the solution. Chapter 8 elaborates on the approach taken for modeling and enacting dynamic workflows extrinsic to the SE process models. In turn, Chapter 9 discusses automated contextual support for SE quality management. Chapter 10 gives insights into task coordination and Chapter 11 deals with SE process exception handling. Finally, Chapter 12 describes the automated contextual integration of knowledge management into the SE process.

**Part III (Evaluation)** is dedicated to the evaluation. Chapter 13 gives details on the technical feasibility and the implementation of the approach. Chapter 14 shows the practical applicability of the solution to a set of concrete scenarios. Finally, a discussion of related work and threats to validity is provided in Chapter 15.

**Part IV (Conclusion)** concludes the thesis with a summary and an outlook.
2. Research Methodology

This chapter presents the research questions addressed by this thesis as well as the research methodology applied.

2.1. Research Questions

Chapter 1 presented a problem statement and distilled main problem areas backed up by literature references: (1) the inadequate process support and implementation in SE; (2) the inadequate support of humans and their interaction in these knowledge-intensive processes; and (3) the inadequate integration of the product and its quality management into these processes. The first item corresponds to the process itself while the other two items refer to the integration of the process and other important aspects of the SE project. Thus, this thesis deals with three main research questions, the first being the general leading theme of the thesis and the other two refining the first.

Research Question 1: Is it possible to support SE projects by not only documenting, but operationally guiding and supporting their processes?

Research Question 2: Is it possible to operationalize and guide entire SE process models with (existing) automated tools?

Research Question 3: Is it possible to connect SE process enactment comprehensively to the actual course of the projects including artifacts and humans?

To answer these research questions the course of action is to analyze SE projects in practice as well as to do a comprehensive literature study. Based on this, we will create more concrete requirements to be fulfilled to answer the research questions.

2.2. Information Systems Research

In particular, this work deals with information systems supporting humans in SE. Therefore, this work relies on a combination of two science disciplines applied for Information Systems (IS) research that was postulated in [HMPR04, HeMa03]: design science and behavioral science [MaSm95]. In the following, we will briefly explain this combination and its suitability for this work.

Information Systems research approaches following behavioral science seek to provide a better understanding of the interplay of organizations, humans, and technologies that have a huge impact on the performance of those organizations. Behavioral approaches, therefore, develop theories to explain or predict organizational phenomena concerning the management, implementation, design, and analysis of IS. Opposed to this, design science aims to create and evaluate concrete artifacts solving the problems identified. These approaches are problem- and solution-oriented having their roots in engineering and sciences of the artificial [Simo96]. As opposed to natural science, however, design science approaches do not examine natural phenomena, but rather deal with those relating to and created by humans [Simo96].
Design plays a central role in these approaches and is to be understood as the goal-driven and deliberate organization of resources for achieving these goals. The combination of these two disciplines has been chosen in this thesis due to its applicability for complex and application-centric IS problems. The latter often cannot be precisely specified (i.e., as mathematical model) and thus cannot be optimally solved by one approach. Instead, they demand for more flexible descriptions and solutions. As an example, [Simo96] presents the creation of a robust IS architecture and classifies solutions to such problems as ‘satisfying’. This means that they may not be optimal, but well suited and good enough for a certain class of problems.

In this work, the framework created in [HMPR04, HeMa03] is applied as shown in Figure 2-1:

As the first step, concrete experiences and information were gathered from two software-producing small- or medium-sized enterprises (SMEs). On one hand, this information comprises concrete requirements and the research goals. On the other, it consists of information about the concrete infrastructure of these organizations, including artifacts, humans and tools.

In the second step, a detailed literature study was conducted revealing information crucial for the SE domain. This included information from various applications for purposes like knowledge management or process management as well as SE domain knowledge (i.e., process models or best practices).

Based on the information gathered and aggregated, a framework was designed and developed. To ensure practical applicability, the evaluation not only included different case studies but also a concrete application of the developed framework in two practical settings.

For combining design science and behavioral science in IS research, [HMPR04, HeMa03] postulated seven guidelines that shall ensure the validity and effectiveness of that research (cf. Table 2-1). In the following, the application of these seven guidelines to this work is briefly discussed. Guidelines 1 and 7 are strictly followed as all efforts of this work result in concepts, algorithms and methods published in scientific papers. The relevance of the objectives was proven by sources from literature as well as the information gathered from two industrial software companies. The evaluation recommended by Guideline 3 is conducted through practical usage by two software companies. As such industrial evaluation with only three small teams is relatively fuzzy and error-prone, a set of concrete case studies has been created to evaluate the applicability of the different contributions of this thesis. Further, this work has a set of concrete contributions (cf. Guideline 4) outlined in Section 1.2. The research rigor (cf. Guideline 5) is facilitated by not only creating design artifacts, but also using these artifacts to create a concrete applicable solution (software) that can be practically applied. The search
process recommended in Guideline 6 was also followed. By the usage of concrete scenarios and a practical application, solutions that may not be optimal but yet satisfying for the problem were found.

Table 2-1: Research guidelines (adopted from [HMPR04])

<table>
<thead>
<tr>
<th>Guideline</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1: Design as an Artifact</td>
<td>Design-science research must produce a viable artifact in the form of a construct, model, method, or instantiation.</td>
</tr>
<tr>
<td>G2: Problem Relevance</td>
<td>The objective of design-science research is to develop technology-based solutions to important and relevant business problems.</td>
</tr>
<tr>
<td>G3: Design Evaluation</td>
<td>The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.</td>
</tr>
<tr>
<td>G4: Research Contributions</td>
<td>Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, foundations, and/or methodologies.</td>
</tr>
<tr>
<td>G5: Research Rigor</td>
<td>Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.</td>
</tr>
<tr>
<td>G6: Design as a Search Process</td>
<td>The search for an effective artifact requires utilizing available means to reach desired ends, while satisfying laws in the problem environment.</td>
</tr>
<tr>
<td>G7: Communication of Research</td>
<td>Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.</td>
</tr>
</tbody>
</table>
3. Background

This chapter provides background information on the characteristics of the SE process and SE process models, respectively. Section 3.1 briefly discusses basic properties of SE process enactment. Section 3.2 then introduces prevalent SE process models as suggested in literature. We provide historical background and then present models prevalent in contemporary SE projects.

3.1. The Software Engineering Process

[BrBe11] provides the following definition of a software process: "A software process is a framework for carrying out the activities of a project in an organized and disciplined manner. It imposes structure and helps to guide the many humans and activities in a coherent manner. A software project progresses through different phases, each interrelated and bounded by time. A software process expresses the interrelationship among the phases by defining their order and frequency, as well as defining the deliverable of the project. ... Specific software processes are called software process models."

As already stated, the SE process is highly dynamic. On one hand, this results from the properties of the created product (i.e., the software). On the other, the creation of the product implies a highly intellectual, creative process that, in turn, necessitates much communication. The latter is needed across different abstraction levels (i.e., from the high level process of a project down to the operational level where concrete activities are executed) as well as different project areas (e.g., ‘Quality Management’ or ‘Software Implementation’). This section enumerates the various groups of persons involved, tasks executed, and artifacts processed in order to explain communication channels as well as the highly dynamic properties of an SE process.

Usually, SE involves various roles [BrBe11]. First of all, there are the software vendor and its customer, who have to agree on the product to be delivered. As part of the software vendor, there exist vertically and horizontally divided areas. Vertically, there are levels such as company and business management, project management, and project staff. Horizontally, aligned from the first product idea to the final product, different teams participate: the requirements analysts communicate with the customer eliciting concrete requirements for the product to be developed. They represent the customer towards the developers. The architects are responsible for the technical foundations as well as architecture of the software and design decisions. In turn, the developers are in charge of the concrete realization of the requirements based on the chosen architecture. A test team verifies the technical functionality of the software, while the requirements analysts are in charge of the functional inspection of the software. Other responsibilities are related to configuration management, problem and change management, and administration. Furthermore, it is common that multiple companies collaborate to create one product or that in one company many projects are executed concurrently. Figure 3-1 shows a schematic description of a selection of different actors, artifacts, activities, and areas of an SE project together with their relations.
Besides the source code, the artifacts processed in a SE project include various plans and specifications. In the following, a selection of artifacts are presented that have been standardized by the Institute of Electrical and Electronics Engineers (IEEE). The software requirements specification (SRS) [IEEE98a] covers the requirements of the software to be developed. The requirements can be split into two parts: customer requirements (similar to the German ‘Pflichtenheft’) and development requirements (similar to the German ‘Lastenheft’). A software quality assurance plan (SQAP) [IEEE02] covers all development, testing and training activities in the project. The software configuration management plan (SCMP) [IEEE05], in turn, describes the necessary configuration management activities. The software test documentation (STD) [IEEE07] contains the documents needed for documenting the software tests. The software validation and verification plan (SVVP) [IEEE04] manages how the validation and verification of the software shall be documented. The design of the software is captured in the software design description (SDD) [IEEE09] and the governance of the entire project is described in a software project management plan (SPMP) [IEEE98b].

Usually, SE projects aim to create or extend software. In this context, various tasks need to be accomplished and coordinated among different groups of persons implying different artifacts. A project begins with the elicitation of its requirements. After their definition, the system architecture must be chosen and built. In parallel, the solution concept needs to be developed, which is mostly done in more than one step producing a preliminary concept first. The actual realization phase starts after having determined all parameters. To be finally deployed the solution must first be tested and, eventually, its different parts be integrated. The entire SE process is rather dynamic due to different factors: The intangibility of the created product makes it difficult to preplan it comprehensively implying a thing called ‘requirements creep’ [Jone96]. The latter describes the fact that in most SE projects requirements are evolving and cannot be concretely defined upfront. Another negative effect of the software’s properties is its aggravated measurability according to quality. To be able to improve
the latter, quality goals must be defined and measured [Kan02]. However, many companies are suffering severe problems in implementing effective measurement programs [STT06].

3.2. Software Engineering Process Models

Explicit SE process models have been developed and used for a long time in SE in order to enable governance, guidance and support for the SE process. In addition, such SE process models shall improve quality of the SE process as well as the produced product by enhancing repeatability and avoiding uncoordinated ad-hoc activities. Furthermore, process models can be the basis for process improvement since a process must be known to improve it. The following sub-sections give a brief overview about common SE process models and approaches.

3.2.1. Classical Approaches

Classical approaches in process specification have existed for many decades. Compared to the more recent agile approaches, they are based on a rather static and heavyweight process model.

Waterfall Model

The waterfall model [Royc70], which can be seen as the earliest structured system development approach, was mentioned first in 1970. It describes a sequential SE process, which originates from the manufacturing industries, and includes the phases depicted in Figure 3-2.

![Fig. 3-2: The waterfall model](image)

These phases are processed sequentially, assuming that a phase transition is only executed if the current phase is finished. The process allows going back one step to the preceding phase, but not further. The waterfall model has turned out to only poorly capture the properties of the SE process. In particular, in SE it is usual that the requirements cannot be completely elicited before development starts. As another disadvantage in SE, designs often cannot be translated into working products in a straightforward way due to various limitations like, e.g., regarding technology.

Spiral Model

The spiral model combines elements of prototype-driven process methods with the classical SE process of the waterfall model [Boeh88]. The former takes into account that it may be difficult to know all system requirements upfront and thus proposes the development of system prototypes first. Its primary focus is to manage and reduce the risks of the overall SE process. Figure 3-3 shows the different phases of the process model.
The process of the spiral model is represented as an expanding spiral corresponding to iterative developments. The inner cycles represent early development stages with system analysis and prototyping. In turn, the outer cycles represent the classic development cycle. Each cycle begins with the activity of risk analysis to incrementally identify critical factors in the project. The model is intended for big projects in risky areas and may imply too much management overhead for smaller projects.

**V-Model**

The V-Model is named after the alignment of its activities in the process model: activities are aligned like a "V" as illustrated in Figure 3-4. The left side represents the elicitation of requirements and the creation of various specifications, whereas the right side represents the verification and integration of the developed system parts. The main objectives are the improvement of product quality and the minimization of risks as well as the facilitated communication of stakeholders and cost reduction for the whole project. The V-Model was initially developed for the German Federal Ministry of Defense in 1986. It was refined later to the V-Model 97 incorporating new approaches like object orientation. In 2005, it re-experienced a major refinement to the V-Model XT (eXtreme Tailoring) [IABG15]. The focus of the new model was to be easily tailorable to various organizations. It further considered stronger involvement of the customer, stronger modularization, and orientation towards incremental approaches. As opposed to the models described before, the V-Model XT is a rather heavyweight model, not only roughly describing different development phases, but also comprehensively covering different project roles and groups as well as their communication (i.e., describing ‘Who’ has to do ‘What’ and ‘When’).
As shown in Figure 3-4, the V-Model not only comprises development tasks, but project acquisition and definition tasks as well. For concrete development tasks (starting with ‘System specified’ until ‘Shipment conducted’), multiple iterations may be applied. The activities to reach the milestones are rather abstract and comprise a number of more fine grained sub-activities. To group the latter, so called process modules are used. For example, process module ‘System Development’ comprises 49 activities (e.g., ‘Preparing overall system specification’) of which some are even specified as a workflow. Furthermore, the mentioned process module comprises 73 so-called products (i.e. artifacts) like ‘In-service documentation’ (includes all data needed by the customer to properly operate the system). These products have relations to the various activities as well as to roles (e.g., ‘Requirements Analyst’). Furthermore, they have complex mutual relations, which include the ‘Content-Related Product Dependencies’ describing content-wise relations in the products, and ‘Generative Product Dependencies’ describing that one product is needed creating another. A key feature of the V-Model XT (eXtreme Tailoring) is its capability to tailor it to the current project by adding or omitting certain process modules even while the project is active.

3.2.2. Agile Approaches

Agile SE approaches [FoHi01] have emerged since classical approaches often fail to cover the dynamic nature of the SE process. In particular, agile approaches put more emphasis on the humans enacting the process as on the process itself. Responding to change is more favored than rigidly implementing a process model. Consequently, small, self-organizing teams are installed. Furthermore, the customer is more tightly integrated into the SE process in order to be able to quickly communicate changing requirements. Another important aspect concerns the utilization of short cycles, which should always produce a working product. Thus, the customer can already get familiar with the product and requirements changes can be communicated earlier.

Scrum

Scrum [DeSi90, TaNo86, ScBe01] is rather a framework than a full process model. Thereby, many of the decisions in the SE process are left up to the team. Scrum teams are self-organizing and cross-functional, meaning they comprise members of different groups such as developers, requirements analysts, or testers. The Scrum process defines three main roles: ‘Scrum Master’, ‘Product Owner’, and ‘Scrum Team’. The ‘Scrum Master’ is a kind of team leader whose main responsibility is the support of the ‘Scrum Team’ by removing impediments that prevent the team from completing its tasks. In turn, the ‘Product Owner’ is something like a proxy for the customer of the project: He analyzes business needs and defines the requirements for the ‘Scrum Team’. The latter is in charge of realizing the functionalities of the software to be produced. Figure 3-5 illustrates the process model.
As shown in Figure 3-5, the Scrum process features different artifacts called ‘Work Products’. These can be separated into two categories: ‘Task Board’ and ‘Burndown Charts’ that list different activities to be accomplished within a certain timeframe. The requirements (i.e., different functionalities of the software) are represented by the ‘Product Backlog’, ‘Sprint Backlog’ and ‘Potentially Shippable Product’. At the beginning of a project, which starts with the ‘Release Planning’ activity, the ‘Product Backlog’ (including all desired functionalities) is specified and the number and length of sprints is determined. After that, the backlog items are estimated by the team and prioritized by the ‘Product Owner’. In the ‘Sprint Planning Meeting’, it is determined which items shall be realized in the current sprint. These items are then moved to the ‘Sprint Backlog’. Within a sprint, all scheduled backlog items are realized and everyday a short ‘Daily Scrum’ meeting is conducted for coordination purposes. At the end of a sprint, the backlog items are reviewed with a presentation of the ‘Potentially Shippable Product’ in the ‘Sprint Review Meeting’. Following the latter, there is an additional ‘Sprint Retrospective Meeting’ to discuss the past sprint.

eXtreme Programming

eXtreme programming [Beck00a, Beck00b] targets at smaller teams and the programming tasks constitute the main focus. As fundamental assumption, the customer does not know all requirements prior to project start. Therefore, the entire process is organized incrementally and dynamically. Requirements are described in terms of user stories which are a lean form of use cases focusing on the user’s view of the system. Extreme programming describes an open, fluent process that relies heavily on the participation of humans.

Some key practices are mentioned in the following: Programming is mostly done as pair programming where two developers share one computer to develop the software. That way, knowledge transfer shall be furthered and the error detection rate shall become high. Tasks are not distributed to humans, but to the team, and then become dynamically distributed. Humans do not have strict responsibilities and work is always shared. The dynamic process builds on permanent testing, integration and refactoring of the code.
Criticisms of extreme programming target at the low level of governance it provides, while relying heavily on the participation of the involved humans, which presumes ideal developers and customers. The process can be seen as too dynamic because it assumes continuous change. It has been proven that changes of the requirements get more expensive in later project stages. Furthermore, in extreme programming, it can be difficult to guarantee an exact amount of functionality at an exact time point.

Unified Process

The unified process [JBR99, Scot02] is an iterative SE process framework that is very popular and has many derivates. As its two main characteristics, this process strongly focuses on the architecture of the developed software and on addressing the risks in early project stages. The unified process knows four project phases as depicted in Figure 3-6.

![Figure 3-6: The Unified Process (OpenUP, adopted from [EcFo15])](image)

The four phases (Inception, Elaboration, Construction and Transition) are separated into iterations. In each phase, different amounts of work in the core disciplines are accomplished. These disciplines are business modeling, requirements analysis, system analysis & design, implementation, testing, deployment, configuration & change management, project management, and environmental tasks.

The unified process comprises several refinements with different focus. Probably, the most well-known is the rational unified process (RUP) [Kruc99]. RUP is a sophisticated and heavyweight variant, which governs activities in great detail. RUP contains over 30 roles and over 130 activities. Thus, it can be used effectively only in teams of more than ten humans. Another refinement is the Open Unified Process (OpenUP) [Ecfo15], which is part of the Eclipse process framework (EPF). All these RUP variants aim to provide a simpler, open version of the process, while capturing all essential characteristics of the unified process or RUP. OpenUP features three levels of granularity: At the project level there are four phases (as defined by the Unified Process): Inception (roughly agree upon the goals of the project), Elaboration (agree on the technical approach), Construction (realize main part of the system), and Transition (make the system ready for its transition to customer). Within each of...
these phases, multiple iterations may take place. The iterations, in turn, comprise several more fine-grained activities (e.g., ‘Develop Solution Increment’ for developing a new part of the software). These activities may be specified in terms of workflows containing even more fine-grained activities (like ‘Implement Solution’ or ‘Implement Developer Test’). The OpenUP process features different kinds of guidance to support the project participants. Activities on the most concrete level are supported by so called steps that roughly outline what has to be done to complete the activity. Furthermore, the model features concrete checklists to be applicable at certain points.

3.3. Summary

This section provides a summary about the SE process extracted from the properties and criticisms of the introduced process models. Due to the numerous efforts regarding explicit process models, it is evident that they are essential for SE projects enhancing repeatability, traceability and, first of all, quality of the process and thus of the product as well. Yet, it cannot be guaranteed that process models are followed since they are mostly abstract and applied manually and documentation-centric.

The waterfall model and in particular the criticism on it show that a rigid process is not the right choice for mirroring the dynamic properties of the SE process. This results to a great extend from the fact that all requirements can be known a priori only in very rare cases. The Spiral model, in turn, tackles this issue as it provides an iterative process, which strongly targets at risk analysis and prevention. Criticism on that model include that it is too heavyweight and not applicable to all kinds of organizations. Finally, the V-Model XT incorporates far-reaching tailoring facilities to be applicable to different organizations. It also puts a strong focus on risk management and communication support. Yet it is still rather heavyweight and thus not suitable for small projects or teams.

Agile approaches were developed as answer to the heavyweight classical process models. Scrum, in turn, puts a strong focus on humans and small self-organizing teams. eXtreme Programming is even more targeted towards the individual. These approaches are lean, but criticism includes that there is not enough governance and thus unpredictable results might be produced. The Unified Process is more static focusing on the architecture. However, some refinements, including RUP, are considered too heavyweight, same as the classical approaches.

All in all, comparing the criticism of the classical and the agile approaches, one can state that it is difficult to provide appropriate process support for SE projects. On one hand, comprehensive process models are often too static and require much cumbersome additional work imposed by the process model. On the other, leaner and more dynamic process models often lack comprehensive support. In particular, they considered to be chaotic and heavily relying on humans. Another fact we discussed constitutes the diversity in SE process models. It is therefore not easy to select process support matching the current company, organization and situation. None of the mentioned process models seems to be applicable to all types of organizations. Altogether, it can be stated that striking a balance can be beneficial, i.e., to provide process guidance without implying too much distracting additional work. A tool providing automated assistance may aid in reaching that goal, taking cumbersome tasks in heavyweight process models and supporting agile teams in the background.
4. Requirement Analysis

This chapter deals with concrete problems and elicits basic requirements from the abstract problems discussed in Chapter 1.

4.1. Concrete Problems

Basically, this work aims to support humans and to address the abstract problems discussed in the preceding sections by a framework. In SE, various tools are prevalent supporting different aspects of process implementation, knowledge management, or quality management. However, many problems remain unsolved as mentioned by the various studies we reference in the relating chapters. To better understand these problems and to support requirements elicitation, we split the three abstract problems up to derive a greater set of more concrete problems that can be better connected to SE tool support. As summary of these problem statements, we extract eight concrete problems relating to SE projects and their process having a big impact on the quality of both the SE projects and the products created by them (cf. Figure 4-1). The latter is separated vertically: On the left side, process specification utilizing abstract process models is depicted. In the middle, the (automatically supported) implementation of such process models is shown for concrete projects. Finally, the left side depicts the SE process as it is really executed by humans creating and manipulating artifacts using SE tools.

We will further support these problem statements by concrete scenarios. The latter were created with information from literature and especially with information gathered from two practical settings. For confidentiality reasons the scenarios are abstracted and generalized. Further, they are centered around a fictional company called ‘The Company’. Not all of the scenarios directly correlate with an abstract problem identified in the Problem Statement. In particular, the first three problems (Automated Process Governance, Context Integration, and Process Dynamicity) are of abstract nature playing a role in most of the scenarios.

Lack of Automated Process Governance (Prob:AutoProc). One problem area concerns process tracking and guidance, referred to as automated process governance in the following. If a project is to be executed in an effective, efficient and repeatable manner, studies have shown that it should be based on a defined process [GGK06]. Furthermore, process models may contain important information about the projects [BWHW06, RiJa00, Mall09]. As discussed in Chapter 3, many SE process models have been developed including Scrum [ScBe01], the Unified Process [JBR99], or the V-Model XT [IABG15, RBTK05]. As a problem, typically, these models exist only on paper or web pages, i.e., they are only used for process specification and documentation. In many cases, the process is rigid and prescriptive, and it differs from the real dynamic work performed in a project [BDS+99, Amb102]. Furthermore, the impact of the models on actors and concrete activities often remains low [Wall07]. Automated support for enacting such process models is desirable. There are numerous tools capable of automated workflow governance. These tools strongly focus on the control-flow perspective meaning they are capable of governing the sequencing of different activities and transferring different tasks to the humans. In addition, they often provide limited means for integrating data objects and an organizational model. However, they fail in covering the different aspects of process models like guidelines or checklists, or dynamic features like the V-Model XT’s dynamic tailoring (cf. Chapter 2). Consequently, the automatically assisted implementation of a whole process model with such tools remains a challenge.
Lack of Context Integration (Prob:ContInt). A second important problem area concerns contextual integration: even if some automated process implementation and guidance is present in a project, this does not necessarily mean that the specified and actually executed processes align. In reality, a myriad of environmental variables affect process enactment [Schw97, MaVe03, BPNS07]. In SE, the latter mostly deal with various actors using different tools (e.g., requirement management tools or IDEs) to manipulate various artifacts being crucial for the process. In turn, these activities and tool interactions are not directly captured in the process models since they are too fine-grained. Thus, a dichotomy between the planned and the actually executed process may exist.

Process Dynamicity (Prob:ProcDyn). Reality has shown that project enactment not always happens exactly as planned [BDS+99, RHD98]. A planned process is a good starting point. However, if the real course of a project deviates from this plan, it will be a challenge to keep the plan in line with reality [ReDa98]. Most contemporary PAIS still rely on rigidly predefined workflows and only feature rather limited abilities to cope with such dynamic changes [Pevd06, ReWe12]. Thus, the planned and the actually executed process diverge more and more, and the former becomes irrelevant over time.

In the following, we present a scenario relating to problems with process model implementation in SE projects. The scenario does not deal with a concrete use case or situation, but rather with a specific process model and issues relating to its automatically supported implementation. For this purpose, we chose the OpenUP [EcFo15] for several reasons:

- Availability: OpenUP is a freely available derivate of the Unified Process. It requires no licensing fees or other costs.
- Understandability: OpenUP is clearly structured and the Eclipse Foundation provides comprehensive documentation free of charge.
- Comprehensiveness: OpenUP covers both abstract and operational process areas including workflows ranging from abstract phases of a project to concrete developer workflows.
- Contextual relations: OpenUP specifies a rich set of entities that relate to real entities or persons in an SE project like artifacts, tools and roles.
- Comprehensive human tasks: OpenUP features various different activities and tasks of different granularities for humans.
- Comprehensive support features: OpenUP comprises a rich set of supportive artifacts like checklists or guidelines.
OpenUP is manually implemented. In particular, the whole process definition exists as web pages [EcFo15]. When implementing OpenUP in an SE project, the involved persons must gather information manually and apply it to the project. We will use OpenUP as scenario for illustrating the following problems: the basic automated implementation of the whole model (cf. Prob:AutoProc), the establishment of connections from this implementation to the ‘real world’ (cf. Prob:ContInt), and the dynamic nature of SE process enactment (cf. Prob:ProcDyn). For the sake of illustration, Figure 4-2 shows five excerpts from the OpenUP website comprising a list of various activities, an operational workflow specification, relations of artifacts and roles, a checklist, and fine-grained activity steps.

Figure 4-2: OpenUp excerpts (adopted from [EcFo15])
These excerpts show that the OpenUP indeed comprises important and useful information to aid the SE process. However, there is neither a tool implementing or supporting this process model nor a strategy on how to achieve this with any tool in place. Thus, this information remains disconnected from the real process enacted in the SE project. The information not only needs to be gathered manually by humans, it is also not tailored to the concrete project or situation. It does feature detailed information from abstract phases of a project to operation workflows. However, the latter, like the ‘Develop Solution Increment’ workflow, are rigidly predefined and not integrated with other tools or the humans in the project. Thus, support for handling unforeseen situations and adapting process enactment are also not in place.

Unplanned Activities (Prob:UnplanAct). The application of PAIS technology in dynamic and evolving domains such as SE is difficult [JaCo93]. Reality often diverges from rigidly pre-defined processes [McCo01, CNGM95] in that domain. In fact, process models cannot cover all workflows actually executed in an SE project. Hence, we distinguish between intrinsic workflows being part of the process and extrinsic workflows (cf. Chapter 8) being unforeseen in the latter. Such extrinsic workflows can be executed based on specific situations, but can be also recurring common tasks (e.g., bug fixing or technology evaluation). These tasks rely heavily on the current situation, remain unplanned and untraced, and may impact timely process enactment (cf. Example 4-1).

Example 4-1 (Ad-hoc activity):
Consider an ad-hoc activity as it was perceived during an interview conducted with a developer as part of an industrial case study. During the interview, a requirements analyst came in, telling the developer that he had to do a presentation for the customer soon. He had already received a current version of the software. However, shortly before the presentation he found a new bug endangering the success of the presentation. Hence, the developer quickly started to work on that issue, was able to fix it, and the requirements analyst received the fix via USB stick. According to the developer, such ad-hoc activities occur often, take up to half an hour, and remain untraced.

In the following, we will use the term process coverage to refer to the coverage of the actually executed processes in an SE project the used SE process model can cover. The models feature a list of standard SE processes. However, they do not cover a great number of activities executed in daily work in an SE project. Thus, these activities remain unplanned and untraced and can even influence the planned processes enactment. Due to these uncaptured activities the planned as well as the actually executed process can move increasingly apart from each other. Furthermore, the planned process can be delayed without exposing the reason for the delay. Finally, the unplanned activities are not guided, supported or governed. They are executed completely manually without any process or knowledge support. Example 4-2 deals with a concrete situation for such extrinsic workflows.

Example 4-2 (Process coverage shortcomings):
The Company uses a SE process model for standard development activities. However, there are various issues in everyday work not covered by such a model. These include activities like bug fixing, refactoring, technology swapping, or infrastructural issues. There have been efforts in The Company to model workflows for these issues in order to provide the humans with automated support and guidance. Since there are various kinds of issues with ambiguous and subjective delineation, however, it is difficult and burdensome to universally and correctly model them in advance for acceptability and practicality. Many activities may appear in multiple issues, but are not necessarily required, bloating different SE issue workflows with many conditional activities if pre-modeled. Figure 4-3 shows such a workflow for bug fixing that contains nearly 30 activities, many of them being conditionally executed for accomplishing different tasks like testing or documentation. An example is provided by static analysis activities that are eventually omitted for urgent cases. Furthermore, there are various reviewing activities, having different parameters (like effectiveness or efficiency), where the choice can be based on certain project parameters (e.g., risk or urgency). The same applies to different testing activities. Moreover, it has to be determined whether a bug fix should be merged into various other branches in the source control system.
As many decisions in the workflow rely on properties of the situation, many activities could be excluded prior to enactment as each situation requires another workflow that marks a subset of the workflow shown in Figure 4-3. However, the situational information for making such decisions is not always in place and gathering it would require additional efforts from humans. Another option, modeling many smaller workflows for different situations is also problematic, as the matching workflow for each situation would have to be determined manually. Additionally, that solution would result in a large number of modeled workflows making the selection of them even more inefficient. Finally, many of the activities and even whole fragments of the workflows would appear in multiple workflows resulting in redundant modeling. Usually, such redundant model fragments are difficult to maintain and might lead to diverging models over time [WRMR11].

Uncoordinated Collaboration (Prob:Collab). In a complex project, there are always persons, tools, activities, and artifacts related to each other [JYW07, SQTR07]. This fact implies that an activity a person executes to change an artifact can have an impact on other artifacts, which again has an impact on the activities of other persons. As example of a relation consider architectural specifications and relating source code artifacts. As some of these activities may be covered by the process, while others are not, this can result in problematic artifact states if many related adaptations by different humans are applied in an uncoordinated manner. As aforementioned, collaboration remains one of the biggest challenges in SE projects [Dust04] and team work is still not adequately supported [LeBo07].

As the sizes of companies, departments and projects grow, communication between collaborating humans and teams becomes increasingly challenging. Humans are often involved in multiple projects in parallel, each of them having its own artifact base the humans work on. Hence, humans are often switching between the projects and concurrently manipulate artifacts of these different projects. In turn, this can lead to a myriad of different problems relating to the artifacts or tasks conducted. Example 4-3 concretizes this.

Example 4-3 (Coordination shortcomings):

Being a growing small to medium sized enterprise (SME), The Company suffers from the inability to satisfy increased coordination needs. Team sizes are growing and various projects are executed in parallel. Humans often have to switch between different projects and within each project larger numbers of humans are working on the same artifacts. Without additional coordination effort things might be easily forgotten.

One concrete problem reported by developers is related to frequent project switches. A person doing this in such a multi-team / multi-project environment must manually gather context information after a switch in order to work effectively: Which assignment has to be processed for which project? What
are potential milestones and deadlines? What is the state of the currently processed assignment? What are upcoming activities to complete it?

Two other problems relate to cooperatively working on the same artifact base. As the first issue in this situation, activities and accompanied changes to artifacts often remain unnoticed by other humans. For example, if two teams (e.g. a development team and a test team) are working on the same source code artifacts they might want to get informed about changes of them. Such information is often transferred manually and is therefore prone to omissive errors.

The third problem directly relates to the artifacts and their relations: Artifact changes often imply certain follow-up actions that are hitherto coordinated manually. Figure 4-4 depicts a scenario detailing this: It deals with a source code artifact being part of an interface component: since the file belongs to an interface component, the applied changes might not only affect the unit tests of the file, but also other artifacts such as the architecture specification or integration tests. Usually, these additional activities are neither covered by the SE process nor governed by any workflow; manual coordination can lead to impacts being forgotten and result in inconsistencies, e.g., between the source code and the tests or specifications. The fact that these activities belong to different project areas with often also different responsible persons makes this even more difficult. Even if not forgotten, follow-up actions could benefit from automated governance and support. Furthermore, it can be difficult to determine which stakeholder should be informed about which change and when, especially considering the dynamic and diverse nature of the artifact-to-stakeholder relationship and various information needs.

**Figure 4-4:** Artifact and implied activity relations

**Process Exceptions (Prob:ProcExc).** During project enactment, unforeseen and exceptional situations occur as the SE process is not fully predictable [Schw97, BDS+99]. In turn, this poses a big challenge to any framework seeking to provide holistic process support for such projects. Contemporary workflow management technology has limited capabilities in this area, only dealing with exceptions directly relating to activities [ReWe12, RAH06]. In practice, process exceptions are often not that simple and also not easily detectable. Further, they may relate to processed artifacts even without the person working on these artifacts noticing them. Finally, to select an exception handling suitable for both the situation and person is challenging.
Complex exceptions are not related to the malfunction of a single tool or program, but to the prescribed process or other more complex coherences in an SE project. Such exceptions can relate to activities being part of the prescribed process or to others being extrinsic to the latter. They may also relate to artifacts processed in the course of the project, even if all activities seem to be executed as intended. Such exceptions are difficult to detect and handle even if the company uses a PAIS providing process implementation support. In the following, two concrete examples (Example 4-4 and Example 4-5) are provided to illustrate this.

Example 4-4 (Exception handling shortcomings):
The company uses an SE process model. However, there is no tool in place to govern, support or enforce the executed process. Consider the following situation: A developer creates new code as intended as part of a project: Assume that it is prescribed by the process of that project that he shall create and execute a unit test for this code. As the process is neither enforced nor supported, however, he can intentionally or unintentionally omit these activities. If such things happen, often a growing portion of the code remains untested. This, in turn, endangers reliability of the code base.

A second scenario deals with a known bug in a source code artifact that is, for example, reported by a customer, tracked in a bug tracking software. The bug is then assigned to a developer who shall fix it. When applying a bug fix to the source code file, the removal of the defect might unintentionally introduce other problems to that file. For example, source code complexity might increase if multiple humans applied “quick and dirty” fixes. Thus, the understandability and maintainability of that file might drop dramatically and raise the probability of further defects.

Non-optimal Quality Management (Prob:QualMan). Another problem affecting many SE projects concerns the quality of the software produced [Jone10]. Hence, quality assurance is a crucial factor for any SE project. However, in many SE projects, quality assurance is understood as applying some bug fixes at the end of the project when time allows for this. Studies have shown that this is ineffective and quality measures should be applied systematically during SE project enactment [Hami88, SHK98]. In particular, this requires proactive as well as reactive quality measures. The challenge is to effectively and efficiently integrate the application of these quality measures with the SE process. Concrete issues include the following: quality management is often considered a ‘nice to have’ discipline creating no additional value. Very often it is difficult to integrate quality management activities with the course of the standard SE process. Furthermore, quality management is often only executed in a reactive fashion applying fixes for known bugs. No quality goals are defined that could be proactively supported to prevent the occurrence of bugs. Example 4-5 illustrates such a situation.

Example 4-5 (Quality management shortcomings):
The company, being a growing SME, starts with various efforts to support reproducibility of project enactment as well as product quality with process management and quality management. As aforementioned, a process model for the SE process is used. Furthermore, as the number of bugs reported by customers shall decrease, quality management tools are applied. This includes bug trackers and static code analysis tools. However, both quality and process management are not well governed or supported. Quality goals are not defined for projects and thus, no proactive quality management can be applied. There is no real awareness of the execution of planned development activities. Thus, it is difficult to integrate quality management activities into the standard SE process. Static code analysis is only used at the end of projects and due to the time pressure often present in that situations, many detected problems still remain unsolved.

Utilized Knowledge (Prob:Knowl). The creation and modification of software is a complex and knowledge-intensive task [RaTi99] and software is an intangible asset. It involves knowledge from different sources, all of which are crucial for the success of the task [LiRu02]. This includes information on the process, the coding style and other specifics of the company, the used framework or area (frontend or backend development), and so forth. Companies often neglect this fact and do not implement proper knowledge management. Even if some knowledge store is implemented, knowledge
retrieval and effective knowledge usage remain an issue [SBBK08]. This often leaves software engineers without all required knowledge and thus makes their work ineffective and error prone.

As mentioned in Chapter 1, wikis are often used to let project participants store specific knowledge. They make recoding of information easy, but management and retrieval of the latter often constitutes a challenge. This is aggravated by the fact that knowledge related to SE is usually context-dependent meaning that it must match the properties of the situation and the involved person. As scenario for illustrating knowledge management shortcomings a situation comprising different information needs in The Company is presented in Example 4-6.

Example 4-6 (Knowledge management shortcomings):
As a growing SME, The Company frequently hires new developers. The latter get training at the beginning to ensure that they can work effectively as early as possible. However, they still might not have a great share of the concrete information relating to projects, tools or the process. For example, this might include information about the coding style applied in The Company. Also, specific process-related information might be recorded somewhere, but the new developer might not know exactly when and where to acquire that information. Another example are technical specifics about the project he starts to work in as, for example, how source control management is applied including information about different development branches and the commit procedure. Lacking all that information, there is a high probability that the developer will cause many issues when he starts working.

4.2. Basic Requirements

This section gives an overview on the high level requirements for a tool providing automated support for the SE process. We elicit these requirements based on two foundations: First, we refer to the problems discussed in Chapter 1, including their support by literature. Second, we refer to our observations from practice as indicated by the scenarios in this section. As aforementioned, the basic requirements listed in the following will be detailed with sub-requirements in the relating chapters of this work.

- **Requirement Automated Process (R:AutoProc):** The most basic requirement to a tool enabling holistic SE project support is to provide SE process support. Related problems have been discussed in Prob:AutoProc. This includes the automatic implementation and enactment of processes in the tool.

- **Requirement Context Integration (R:ContInt):** As elucidated in the problem statement (cf. Prob:ContInt), there is a myriad of contextual information in the project having a significant impact on process enactment. For example, context information plays an important role for collaboration, quality management, or exception handling. Therefore, a tool aiming at holistic SE project support must have facilities to integrate process enactment with context data.

- **Requirement Dynamic Process (R:DynProc):** SE projects are dynamic as already shown in the problem statement (cf. Prob:DynProc) and confirmed by the scenarios in this chapter (e.g. relating exception handling or quality management). Therefore, a tool supporting these projects must be capable of coping with dynamically changing situations and aligning the process with their properties.

- **Requirement Process Coverage (R:ProcCoverage):** SE process models cover many workflows. However, as shown (cf. Prob:UnplanAct), they disregard many activities and processes executed dynamically as part of everyday work. When aiming at true holistic support for SE projects, a tool must include these workflows and activities as well.

- **Requirement Coordination (R:Coord):** SE projects comprise numerous different areas, actors, and artifacts. Projects are executed in parallel and multiple persons work on the same artifact base. Activities, roles and artifacts have relations to each other and collaboration is not easy to maintain. This can lead to various issues as discussed (cf. Prob:Collab). A tool providing holistic SE project support must be aware of such connections and be capable of managing coordination and collaboration in such a project.
• **Requirement Exception Handling (R:Exc):** In an SE project, many unforeseen problematic situations might occur (cf. Prob:ProcExc). Newly created problems might not directly show up and be obscured from their creator. A tool aiming at SE project support should have facilities to detect such complex exceptions and automatically assist humans in handling them.

• **Requirement Quality Management (R:Qual):** Quality management is a crucial as well as an underestimated part of SE projects. This has been agreed upon in literature (cf. Prob:QualMan) and our practical observations confirmed this, as well. The intangibility of the produced asset (the code) makes it difficult to even be aware of its state. Furthermore, if quality problems are detected, counter measures must be executed in alignment with the process. If a tool shall provide holistic support for SE projects, it must be capable of supporting these complex tasks.

• **Requirement Knowledge Management (R:Know):** As SE projects are knowledge-intensive undertakings with a multitude of different complex information, the latter is not easy to manage. This is confirmed by literature (cf. Prob:Knowl) and practice (cf. the scenario in this chapter). A tool aiming at holistic SE project support must enable the collection, management and dissemination of that knowledge in a process-centered and context-sensitive manner.

### 4.3. Requirements Verification

We have conducted a comprehensive literature study comprising many aspects of SE projects. One reason for this was to support the elicitation of requirements for a framework providing comprehensive support for SE projects. However, the study also included a myriad of tools and approaches aiming at the support of different aspects of SE projects. In this section we discuss how the results of this study can be used to verify the requirements we have elicited.

We have examined approaches of different areas that relate to the topics identified as important for this thesis. An important area are Software Engineering Environments (SEE). These are tools aiming at comprehensibly supporting SE projects. In this area, we have examined various CASE (Computer-Aided Software Engineering) tools (e.g., [EKS93]), Process-Centered Software Engineering Environments (PCSEEs, e.g., [BFGL94, CLH95, BEM94, Barg92b]), modern SE Environments (e.g., [dZR+04, JYW07, HaLa10, WEB+09, dFOT10]), and other contemporary SE approaches (e.g., [BWHK12, PVPB12, GTS10, CAG12]). For a thorough discussion we refer to Chapter 6.

Another important area are processes and their automated enactment. In this area, we have examined WiMS/PAIS (e.g., [Cumb07, Inta15, vdtH05]), process configuration approaches (e.g., [RSS10, Gott09, HBR10, LDH09]), artifact-centric process approaches (e.g., [BHS09, KüRe11a]), process adaptation approaches (e.g., [Wesk01, SMO00, WRWR09, MTS08]), semantic process annotation approaches (e.g., [Mich15, PDB+08, AFKK07, BGM07, ABB+07]), declarative process approaches (e.g., [Pesi08]), and approaches for contextual process integration (e.g., [LSH+06, DGD07]). Besides that we also took into account context modeling approaches like [KMK+03, FaCl04, GPZ04] (see Chapters 7 and 8 for details). In the context of dynamic processes, we have also reviewed approaches for process exception handling (e.g., [MGR04]). Such approaches are discussed in more detail in Chapter 11.

We have also examined various approaches from other areas identified as important for SE projects. These are knowledge management approaches (e.g., [BDi08, Lia03, BWT04]) and collaboration and coordination approaches (e.g., [LeBo07, BSV07, Dust04]). For a more thorough discussion we refer to Chapters 10 and 12. Furthermore, we examined different approaches for SE quality management. These included approaches for software metric application (e.g., [OJe97, GKM02]), software measurement tools (e.g., [ScJe06, LiZh05]), and approaches for the Goal-Question-Metric (GQM) technique (e.g., [FaWu09, STS05, HuFa05]). For more information on these see Chapter 9.
Relevance and Completeness

The various approaches examined show the relevance of the elicited requirements for SE projects relating to various application cases. The need for tool support for SE processes (R:AutoProc) is confirmed by the various SEE approaches. Automated process support in general is the target of countless PAIS and WfMS approaches. The importance for contextual integration (R:ContInt) is discussed by various SEEEs as well. According to them, such information comprises artifacts, various types of knowledge, or persons and their interaction.

As many approaches confirm, processes also need to be handled dynamically (R:DynProc). On one hand, various SEEEs cover this topic and provide capabilities to change running processes. On the other, there exist a myriad of approaches for configuring or changing processes. Such approaches even offer the capability to automatically change running processes. This is often used for handling exceptions occurring during process enactment. This also confirms that process exception handling (R:Exc) is a relevant topic for a tool that automates processes. In addition to that, much attention has been paid to unstructured processes that are not pre-planned as part of a process model. Constraint-based and declarative process approaches deal primarily with such processes. This confirms the importance of capabilities of a tool to also cover such processes (R:ProcCoverage).

A crucial factor for any SE projects is quality management (R:Qual). This is confirmed not only by approaches explicitly dealing with this topic, but also by many SEE approaches that take into account source code artifacts and aim at supporting and improving their management. The same applies for collaboration and coordination support (R:Coord). Many specific approaches stress the importance of this topic for SE. In addition to this, various SEEEs also integrate facilities to support this. Another important area for SE projects is knowledge management (R:Know). This is confirmed both by various dedicated approaches as well as the integration of knowledge management capabilities in many SEEEs.

The goal of our approach cannot be to solve each and every problem in SE. Therefore, the requirements also cannot be considered as complete for SE. However, we can show that the selected requirement areas cover important aspects also mentioned in a myriad of other approaches and that those approaches do not discuss or cover important areas that we have omitted. SEEEs have existed for multiple decades now and each of them covers different areas and capabilities. However, topics that repeatedly occur are the following: processes, with a strong focus on dynamicity as well as people and collaboration aspects. Furthermore, they deal with various entities that can be considered as context to the tools and processes, as, e.g., artifacts and people. Furthermore, they deal with different kinds of knowledge that is crucial to SE projects. To the best of our knowledge, these SEEEs do not cover other core aspects that we have omitted in our discussion. Contemporary SE approaches, however, show two trends gaining momentum: cloud-based SE and global SE, which both correspond to each other. We have decided to put this not to focus in this work as it can be considered primarily as a technical aspect. Our requirement areas are more focused on content-related issues of SE projects.

Relatedness and Generalization

The various approaches we have reviewed not only show that our requirements are relevant for SE projects, they can also serve as indicator that they are related to each other and that their combination is essential for successful SE projects. Again, SEEEs serve best as comparative approaches as they share the same goal as our approach. These approaches often have a strong focus on the SE processes and they connect it with various other areas. None of them combines all of them but multiple approaches respectively combine it with contextual data, collaboration support, knowledge management, and quality management relating the SE artifacts. For a fine grained discussion of the different features of different approaches see Chapter 15.

In the first place, our approach is targeted at SE projects. This means neither the approach nor its requirements can be automatically seen as generally applicable for all domains. However, SE is a vast field and not necessarily a distinct domain. Software is developed in various domains like the
automotive or the healthcare sectors. Furthermore, in SE slightly different approaches to project and process management are utilized. Some projects apply huge and heavyweight process models with hundreds of controlled artifacts. Others apply lightweight and agile approaches that prescribe hardly anything and mostly rely on people. The various approaches and tools we have reviewed also show this diversity. They are applied in various domains like the automotive or the healthcare sectors or are applied in projects regulated by state authorities. Furthermore, they involve all the different approaches to project and process management that are prevalent in SE (e.g., Scrum or V-Model XT). Therefore, we assume that the requirements we elicited are applicable for the vast majority of SE projects regardless of their domain or process approach. Furthermore, in the evaluation of this thesis, we will show the application of our approach to slightly different process approaches for SE and also to a process of the software modernization domain.

4.4. Summary

This chapter elicited eight basic requirements for a tool that aims to provide holistic project and process support for SE projects (cf. Table 4-1). These requirements are aligned with the abstract problems discussed in Section 1.1. To further illustrate the requirements and demonstrate their practical relevance, a set of concrete scenarios was presented, which will be taken into account for validating the developed approach (cf. Chapter 13).

<table>
<thead>
<tr>
<th>Requirement Area</th>
<th>Requirement ID</th>
<th>Description</th>
<th>Detailing Chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic functionality</td>
<td>Requirement R:AutoProc</td>
<td>Automated process enactment / implementation</td>
<td>6</td>
</tr>
<tr>
<td>Basic functionality</td>
<td>Requirement R:ContInt</td>
<td>Contextual integration of process enactment</td>
<td>7</td>
</tr>
<tr>
<td>Basic functionality</td>
<td>Requirement R:DynProc</td>
<td>Dynamic process enactment</td>
<td>7</td>
</tr>
<tr>
<td>Extended functionality</td>
<td>Requirement R:ProcCoverage</td>
<td>Extended process coverage</td>
<td>8</td>
</tr>
<tr>
<td>Extended functionality</td>
<td>Requirement R:Coord</td>
<td>Task coordination</td>
<td>10</td>
</tr>
<tr>
<td>Extended functionality</td>
<td>Requirement R:Exc</td>
<td>Process exception handling</td>
<td>11</td>
</tr>
<tr>
<td>Specific functionality</td>
<td>Requirement R:Qual</td>
<td>Quality management integration</td>
<td>9</td>
</tr>
<tr>
<td>Specific functionality</td>
<td>Requirement R:Know</td>
<td>Knowledge management integration</td>
<td>12</td>
</tr>
</tbody>
</table>

The requirements are categorized as follows: Basic functionality requirements cover the basic facilities a tool must provide to holistically support the SE process. They do not refer to functionalities providing additional value to humans. However, they are crucial and constitute the basis for the other functionalities. Extended functionality requirements cover functionalities that enable general automatic and contextual support for humans in various areas of a project. The third area refers to specific functionality requirements and is targeted to specific areas of SE projects. Note that not all possible areas of an SE project are covered in this work as this would go beyond the scope of this thesis. Rather, the focus is on two areas of knowledge and quality management as these have been proven important parts of each SE project.
Part II

Solution
Chapter 4 has elicited basic requirements for holistic SE process support. To foster the understandability of our solution, this chapter introduces fundamentals and premises of this work. In particular, we present basics on process management and a distinction between the term process and workflow as well as two types of workflows relevant in the context of SE projects.

A process corresponds to a set of interconnected activities executed in a certain order with a certain goal and outcome. Process management includes modeling, enacting, analyzing, and optimizing processes. In particular, explicitly specified process models can have a positive impact improving the efficiency and repeatability of the specified activities.

5.1. Process Modeling

Process modeling refers to the explicit specification of the activities to be executed as part of the process. The sequencing of these activities can be modeled based on patterns like loops enabling the repeated execution of activities or parallel branching enabling the parallel execution of multiple activities. Usually, processes are modeled as directed graphs. There exist various prevalent notations for process modeling, like Event-driven Process Chains (EPC) [Sche01], Business Process Modeling Notation (BPMN) [OMG11a], Petri Nets [Peter81, vdAa98], and UML Activity Diagrams [Part10, OMG11b]. This work relies on BPMN as this notation is prevalent both in industry and academy. Therefore, we will briefly introduce basic elements of this notation as shown in Figure 5-1.

As illustrated by Figure 5-1, BPMN uses explicit start and end events, activities, and different structuring elements (i.e., workflow patterns). These are used to structure the sequencing of activities. Their semantics have been abstractly described in [RHM06, RHEA04a, RHEA04b, vtKB03, LWR14, BLWR12]. There exists a number of different patterns describing various situations and allowing for a high expressive power. The inclusion of all these patterns is often not required [ZuRe08] and can even have a negative impact on the understandability of the models [MRv10, ZSH+15]. Furthermore, this work aims at the automated implementation of processes. However, most WfMS only include a basic set of patterns (e.g. [DaRe09]). Therefore, this work builds on a small selection of patterns. These are detailed in the following. For an overview on BPMN, we refer the reader to [OMG11a].
Control Flow Patterns

For control flow modeling the following basic patterns are provided: Sequence, AND-split, AND-join, XOR-split, XOR-join, and Loop [vtKB03]. With these patterns which constitute the basis of any process specification language [Mend08, zuRe08], most workflows can be covered. Furthermore, the patterns can be easily transformed to languages like Petri Nets [Peter81, vdAa98] or WS-BPEL [BPEL07, VdH02]. There exist other control flow patterns like Multi-Choice / OR-split [vtKB03]. However, to set a focus this work presumes the sole usage of the basic control flow patterns. Particularly, the usage of other patterns can make the process model more complex enforcing errors proneness [MRv10, Kind06, MNA10]. Furthermore, it is possible to build other control flow patterns using the basic ones, e.g., to compose an OR-split using XOR- and AND-splits [MDA08]. Figure 5-2 illustrates the workflow patterns.

![Workflow control flow patterns](image)

Figure 5-2: Workflow control flow patterns

The ‘Sequence’ pattern describes the simple sequencing of activities illustrated by the activities ‘5’ and ‘6’ in Figure 5-2. The pattern is used implicitly and is therefore not considered when mentioning patterns in the succeeding chapters. The AND pattern (more precisely, the AND-split and AND-join) allows for the parallel enactment of two or more branches. In Figure 5-2, for example, activities ‘5’ and ‘6’ are executed in parallel to activity ‘7’. The XOR pattern features multiple branches, enabling the exclusive enactment of one of them depending on facts represented by workflow data elements. As aforementioned, this is depicted in BPMN by a question string next to the split pattern. In Figure 5-2, the XOR pattern allows for the exclusive enactment of either activity ‘3’ or ‘4’. In order to enable the repeated enactment of certain activities, loops can be used. This work presumes the application of an explicit LOOP pattern with well-defined semantics. Figure 5-2 shows such a pattern, allowing for the repeated enactment of the activities ‘5’, ‘6’, and ‘7’. When using the LOOP pattern, all activities between the LOOP-split and corresponding LOOP-join are set to state ‘Not Active’ when the LOOP-join becomes activated. The same applies to the arcs within the LOOP, which are set to state ‘not_signalled’. As illustrated in Figure 5-2, BPMN does not have an explicit symbol for the LOOP pattern, but uses the one representing the XOR pattern instead. The differentiation is made by the branch leading backwards in the workflow versus multiple forward-pointing branches at the XOR pattern.

Block Structured Process Models

To foster readability and limit error proneness, this work presumes block-structured process models and workflows. For a distinction between these two terms, we refer to Section 5.2. Block structured process models have lower error probability and are easier to understand [ReMe08, MRv10, CoGa09]. This approach on structuring is derived from the well-known structuring in programming languages [Dijk72]. In block-structured process models, the control flow patterns are organized in a way that enables easy separation of the process model into nested blocks. These blocks can be activities as well
as patterns or the workflow itself. Each block must have a unique start and end point [Reic00, RRKDO5, KHB00]. The blocks can be regularly nested, i.e. without overlap [ReDa98, KHB00, Reic00]. For workflows structured differently, in most cases a transformation to a block structured model can be applied [VVK08, MRv10]. Figure 5-3 illustrates the block structuring of a workflow.

The properties of a block-structured workflow are elucidated in Figure 5-3: Each block has a unique start and end point. This also applies to the workflow itself, which can be seen as a block as well. Blocks may be nested and do not overlap. Further information on properties of such workflows can be found at [Reic00, VdvH02, vdBa02].

5.2. Basic Terminology and Premises

Process and Workflow

The terms of process and workflow management are closely related. This section provides a delineation of the two. In literature, there exist different views on and definitions of the two terms. The most common distinction is that (business) process management is something rather abstract and strategic, whereas workflow management is more concrete. Furthermore, workflow management is mostly seen as technical implementation of process management. More precisely, workflow management focuses on the flow of orchestrated activities. The Workflow Management Coalition (WfMC) provides the following definition [Holl93] of workflow management:

"The automation of a business process, in part or in whole, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules"

Another definition is provided by Gartner Research [HPN08]:

"Business process management (BPM) is a process-oriented management discipline. It is not a technology. Workflow is a flow management technology found in business process management suites (BPMS's) and other product categories."

This work roughly follows these definitions. Process is referred to as something rather abstract not implemented in software. In turn, workflow is referred to as something more concrete and operational
as well as something being implemented with a software tool. Therefore, it provides an automated facility to govern the flow of activities.

Enactment Tools

Various kinds of tools exist enabling different levels of process automation. This section briefly delineates the different types and provides a clear terminology for the remainder of this work. As stated in Section 3, we presume a workflow as technical implementation of a process or a part of it. Usually, this is achieved by a Workflow Management System (WfMS) whose focus is on the automated enactment of the specified workflows. Recently, more complex tools emerged that enable a more comprehensive representation of processes inside the tools incorporating more aspects like organizational models. These are called Process Management Systems (PMS). Besides that, another kind of tools emerged, which are even more comprehensive, Process-Aware Information System (PAIS). In this work, we refer to the term WfMS as we rely on the core functionality of such tools: the ability to automatically and correctly govern the sequencing of activities as specified by a workflow.

Human-centric Workflows and Activities

In the introduction and problem statement, we have shown that SE processes are essentially knowledge processes that largely depend on the knowledge workers involved. Furthermore, the discussed process models almost exclusively describe activities that are carried out by such knowledge workers or cannot be processed without human intervention. Most of the activities involve tools (e.g., IDEs) executed by humans. On account of that, complete process automation where tools automatically process the involved activities is not feasible in SE. Furthermore, SE projects are rather dynamic and the SE process models remain rather abstract. The concrete enactment of the activities and the selection of the involved tools largely depends on the executing humans. Thus, a direct technical integration of such tools is also not feasible. Having this in mind, this work focuses on human activities.

5.3. Basic Definitions

This section briefly elaborates on the basic process management definitions. As the automated implementation of process models utilizing workflow management technology is one of the goals of this work, the following definitions refer to the term workflow. However, the defined meta-model applies to processes in the same way. Before showing the concrete definitions, the basic concepts are briefly elaborated. The specification of a workflow leads to the creation of a workflow template (often also referred to as workflow type). The concrete enactment of activities specified by the latter is then done using a concrete workflow (often also referred to as workflow instance). Both workflow template and workflow instance contain different elements for structuring the activities to be executed. For the workflow template, these elements are activity templates representing the activities to be executed, arcs for connecting them, data element templates representing the data in the workflow, and workflow patterns for explicitly structuring the sequencing of the activities. The workflow instance is specified analogously containing activities, arcs, and data elements used for representing the state of a particular workflow instance based on the definition in a corresponding workflow template. Figure 5-4 depicts a representation of these concepts in BPMN notation. Therein, the workflow instance is in a certain state: activity ‘3’ is activated whereas activity ‘4’ has been skipped by the XOR pattern.

We will now define the different concepts illustrated by Figure 5-4. First of all, the data elements of a workflow can be roughly separated in two categories: documents needed or processed as part of the executed activities and variables needed for evaluating decisions modeled by workflow patterns. Decisions determine, for example, whether and how often a loop shall be executed. In BPMN, variables are only implicitly shown as text next to a workflow pattern (cf. Figure 5-4). When implementing such a process by a technical workflow, decisions have to be represented by variables in the workflow. The other category, the documents, are often not integrated in the workflows at all as
many tools do not support this. As the following definitions refer to workflows, the data elements will correspond to the variables used to govern the workflows. Another detail refers to the workflow patterns created by gateways in BPMN: Exclusive decisions (XOR-split and join) have the same symbols as loops (LOOP-split and join). The only difference is that the LOOP-join is applied before the split and one of the outgoing arcs of the LOOP-split points backward in the workflow to the join.

![Workflow Template and Workflow Instance Concepts](image)

Given this background, the concepts are now formally defined, starting with the workflow template.

**Definition 5.1 (Workflow Template)**

A workflow template is a tuple \( \text{wfTempl} = (\text{type}, \text{name}, \text{nodeSet}, \text{arcSet}, \text{nodeTypes}, \text{arcTypes}, \text{arcCond}) \) where

- \( \text{nodeSet} \) is a set of nodes and \( \text{arcSet} \) is a set of directed arcs; \( (\text{nodeSet}, \text{arcSet}) \) builds a connected, directed graph.
- \( \text{nodeTypes} : \text{nodeSet} \rightarrow \{\text{Start}, \text{End}, \text{ActivityTemplate}, \text{AND-split}, \text{XOR-split}, \text{LOOP-join}, \text{AND-join}, \text{XOR-join}, \text{LOOP-split}, \text{DataElement}\} \) assigns to every node \( n \in \text{nodeSet} \) a node type \( \text{nodeType}(n) \).
- \( \text{arcTypes} : \text{arcSet} \rightarrow \{\text{ControlFlow}, \text{DataFlow}\} \) assigns to every arc \( a \in \text{arcSet} \) an arc type \( \text{arcType}(a) \).
- \( \text{arcCond}(a) \) assigns to every arc \( a \) with \( \text{arcType}(a) = \text{ControlFlow} \) a transition condition \( \text{cond} \) or \( \text{TRUE} \) (meaning the condition is always evaluated to true). For every arc \( a \) with \( \text{arcType}(a) = \text{dataflow} \), \( \text{arcCond}(a) \) remains undefined.

\( \text{WFTemplates} \) describes the set of all definable workflow templates.
The workflow template has properties indicating its type and its name. The latter must be unique and thus can be used as ID for the template. It contains a set of nodes and arcs that may have different types. A workflow template contains a unique start and a unique end node. We have decided to only permit one of these as this promotes understandable workflow models as stated in [MRv10].

Nodes having the type activity template have exactly one incoming and one outgoing arc connecting it with other nodes in the workflow template. This guarantees correct workflow enactment (cf. Section 5.4). By contrast, the split nodes have multiple outgoing arcs and the join nodes have multiple incoming arcs. An arc has always a distinct source and destination.

The aforementioned concept is utilized for specifying the activities for various tasks in the workflow template. The following concept, however, depicts concrete instances of the entities defined with the preceding workflow template. Its aim is to assign states or concrete values to a workflow instance based on a workflow template.

**Definition 5.2 (Workflow Instance)**

A workflow instance is a tuple $\text{wfInst} = (\text{type}, \text{name}, \text{wfTempl}, \text{nodeState}, \text{arcState}, \text{actInstSet}, \text{dataElVal})$ where

- $\text{wfTempl}$ is the workflow template on which $\text{wfInst}$ is based.
- $\text{nodeState}: \text{nodeSet} \rightarrow \{\text{Not Active}, \text{Active}, \text{Started}, \text{Finished}, \text{Skipped}\}$ assigns to every node $n \in \text{nodeSet}$ an enactment state $\text{nodeState}(n)$.
- $\text{arcState}: \text{arcSet} \rightarrow \{\text{Not_signalled}, \text{True_signalled}, \text{False_signalled}\}$ assigns to every arc $a \in \text{arcSet}$ an enactment state $\text{arcState}(a)$.
- $\text{actInstSet}$ is a set of all activities (i.e., nodes that are based on the nodes with $\text{nodeType}(node) = \text{ActivityTemplate}$ in $\text{wfTempl}$) contained in $\text{wfInst}$.
- $\text{dataElVal}(node)$ assigns every node $a$ with $\text{nodeType}(n) = \text{DataElement}$ a value.

WFInstances describes the set of all definable workflow instances.

The workflow instance has a connection to the workflow template it is based on. Furthermore, it assigns states to the nodes and arcs. When a workflow instance is created, all nodes are in state ‘Not Active’. Each node then becomes activated by its incoming arc and enters the state ‘Active’. In that state, it can be started (e.g., by a human if it is an activity), entering the state ‘Started’. When the human finishes the activity, it enters state ‘Finished’. If an activity does not get executed in a workflow instance because of a workflow pattern that implies the enactment of only one (or a distinct selection) of its outgoing arcs (e.g., the XOR pattern), the activity will enter state ‘Skipped’.

The arcs have states as well. The initial arc state is ‘Not_signalled’. When a node is finished, all outgoing arcs chosen for enactment enter state ‘True_signalled’. The arcs not chosen for enactment enter state ‘False_signalled’. For the enactment paths that are thus not chosen, a so called dead path (see [LeRo00] for more details) elimination is conducted. By this, all arcs and nodes in the ‘deselected’ path are set to the states ‘False_signalled’ respectively ‘Skipped’.

### 5.4. Correctness

If process models are to be executed by a WfMS, the correctness, i.e., behavioral soundness, of this enactment is crucial [Reic00, Hall10]. This work does not deal with behavioral soundness of workflow enactment. It rather assumes and relies on such correct enactment. To give the reader an idea about the notion of correctness assumed in this work, in the following 13 properties of correct enactment are listed. A workflow instance is structurally correct if it satisfies Properties 1 – 9 and is correct if it satisfies Properties 10 – 13 in addition to this. For a more in depth discussion and formal definitions of correctness we refer to [Reic00, Hall10].
**Property 1.** The workflow template has at least one start and end node. Each start node is connected with one outgoing arc and has no incoming arc. Each end node has one incoming arc and no outgoing arcs.

**Property 2.** Every activity is reachable by at least one path from at least one start node.

**Property 3.** From every activity, there is at least one path to at least one end node.

**Property 4.** Every activity has exactly one incoming and one outgoing arc.

**Property 5.** Every split pattern (cf. Figure 5-2) has exactly one incoming arc and at least one outgoing arc. Join patterns have at least one incoming arc and exactly one outgoing arc.

**Property 6.** Cycles / loops only exist by the explicit usage of loop patterns.

**Property 7.** A loop consists of an explicit loop entry and a loop exit pattern. Between these two there is exactly one loop arc enabling the repeated enactment of all activities between the two loop patterns.

**Property 8.** No data element is read before it has been written.

**Property 9.** A data element cannot be written by more than one activity in parallel.

**Property 10.** For each activity, there exists at least one state in the workflow instance, in which that activity can be executed.

**Property 11.** No deadlocks occur in workflow enactment. At any time during the enactment of a workflow, there is at least one activity active or in enactment. Otherwise, the workflow is finished.

**Property 12.** If an end node is activated there is no active or executing activity from which the same end node can be activated a second time.

**Property 13.** A loop terminates after a finite number of loop iterations. Every started activity terminates in finite time.

### 5.5. Types of Workflows

Another term that plays a rather important role in this work is the notion of intrinsic and extrinsic activities or workflows.

The core distinction between extrinsic and intrinsic activities is that intrinsic activities are explicitly covered by and thus executed within the process, whereas extrinsic activities are not even specified and thus hard to trace. This correlates to the problem discussed in Chapter 1: In SE, the planned process is often too rigid and cannot cover all activities concretely executed. Therefore, there is an ever growing gap between the plan and the real project enactment. In this case, process means a SE process model like the Unified Process [Scot02]. These intrinsic activities mostly deal with the development of new software and all comprising activities or the management of that development. Figure 5-5 illustrates the two different types of workflows.

![Figure 5-5: Extrinsic and intrinsic workflows](image-url)
Figure 5-5 shows, as example of an intrinsic workflow, the 'Develop Solution Increment' workflow of the OpenUP process being a refinement of the Unified Process. As example for an extrinsic workflow, a bug fixing workflow is depicted. Figure 5-5 further illustrates the problem with extrinsic activities: they may be supported by some tool, but are executed externally to the process. In turn, this hinders comprehensive process monitoring and tracing exacerbating repeatability in a controlled fashion.

This chapter introduces the developed framework for enabling comprehensive and context-aware process support in SE projects. As stated in Chapter 1, the goal is to achieve comprehensive support for SE projects and their process by combining various technologies and approaches into one framework. Chapters 1 and 4 have discussed specific problems that exist in SE projects as well as requirements that exist for a framework aiming at the holistic support of such projects. Many of these requirements are rather specific and target areas like quality or knowledge management. However, to integrate solutions for these requirements into one approach as well as to enable a holistic view on SE projects a framework must be established. On one hand, its purpose is to integrate the different solutions holistically. On the other, it must provide basic functionalities regarding process, human, and context management. Therefore, this chapter first details the requirements we have elicited in Chapter 4 as basic requirements for holistic SE process support. After that, we introduce the different components of our framework as well as the different technologies we have integrated into it. In particular, we present a solution for the basic requirements and the interplay of the components. More specific functionalities that rely on these functionalities, like quality management, will be detailed in Chapters 7 – 12.

In particular, this chapter presents concepts that enable the following optimizations:

- Comprehensive tool support for SE processes is enabled by connecting and integrating various areas of an SE project.
- The SE process is governed and supported by a tool that actively supports humans, but does not create cumbersome additional work interfering with actual process enactment.
- Tool and process enactment is contextually integrated and thus aware of various properties of the situation in which they are executed.

6.1. Requirements

This section elicits extended requirements to create an awareness of what has to be achieved to enable automated process support for SE. Example 6-1 shows a situation that illustrates some grievances often present during SE process enactment:

Example 6-1 (Process management shortcomings):
Consider the following situation in The Company: The SE process is enacted using various tools. A project management tool is used for planning high-level human tasks often called assignments. The Company uses a SE process model like the Unified Process [Scot02]. However, the planning of the human assignments takes place on a relatively coarse-grained level and only places these abstract assignments within a certain project phase or iteration. The assignments are distributed to the respective humans being in charge of their completion. Information about completion of the assignments is manually distributed back to the management software by the humans. The timely completion of an iteration, phase or project is continuously endangered as many dynamic factors come into play in real enactment: Quality management is not continuously applied and, thus, at late stages of a project, there often occur a large number of bugs and problems to be dealt with. In turn, these bugs are tracked using a specialized bug tracker that has no connection to the project management tool. Bug
This chapter explains the building blocks and components of the solution approach to comprehensively support SE process enactment. Thus, it deals with the first and basic requirement elicited in Chapter 4 concerning automated SE process support (R:AutoProc). We split this rather abstract requirement up into a set of more concrete sub-requirements. Thereby, we concentrate on advanced requirements for a framework aiming to support an entire SE project including its processes. Therefore, we omit basic workflow requirements like correctness of workflow enactment (e.g., [GHS95, Reic00]) or the ability to deliver human tasks to the respective humans or roles. The following requirements are backed up by literature as well as our observations from practice (see the scenarios in Chapter 4 and the one from Example 6-1).

- **Requirement R:AutoProc:Tool (Tool assisted process enactment):** To be able to automatically support SE process enactment, a tool must be in place that governs the process. However, literature on agile software development movement (cf. e.g., [Amb02, Kruc04, RiJa00, Schw97, BDS+99]) argues that tools should not be central in SE projects. Moreover, practice has shown that such support is desirable, yet the tool should always keep to the sideline, not require additional effort and not be cumbersome to use. This should be supported by easily accessible user interfaces that are well integrated with everyday work.

- **Requirement R:AutoProc:SEProc (SE process modeling capability):** Another basic requirement for automated support of SE processes is the ability to cover the utilized SE process model like the VM-XT or OpenUP. These processes have various properties representing the nature of each process. A tool that aims to provide automated process support must thus be capable of modeling these properties and cover processes to a great extent.

- **Requirement R:AutoProc:Auto (Automatisms in place):** Automated process support should not create cumbersome extra work for the humans involved. However, a tool aiming for comprehensive project support will face a myriad of standard situations, for which it should have a defined behavior. In literature, there exist various examples of such situations. [Frie03] presents three real-world examples: mail filtering, product configuration, and business rules. The latter are addressed by [Morg02] as well. Examples include a situation, where a customer should be granted the status of a premium customer if he has a certain balance or is a longtime customer. Our practical experiences show that such situations occur frequently in SE projects as well. As example take SE quality management: assuming that a tool supporting an SE project is capable of uniting quality management with process enactment, it will have to deal with standardized entities and events relating to quality management. An example are bug reports or reports from code analysis tools. In turn, the processing of respective reports might be a crucial building block for generating an awareness of the source codes state. Nevertheless, the behavior of such a tool in such situations must be specified in some way. Therefore, a tool aiming at comprehensive process support should be capable of easily executing automatisms to support repetitive tasks associated with process enactment. These automatisms should be easily configurable for humans to avoid ‘hard coding’ of procedures and thus loosing flexibility.

- **Requirement R:AutoProc:DynSit (Cope with dynamic situations):** The process of creating software is dynamic and many factors in SE projects are not easily foreseeable [Schw97, BDS+99]. Furthermore, many process models are in some way prescriptive [Amb02]. If a tool had rigidly implemented a prescriptive model, this would impose a high probability of creating a gap between that process and the dynamic events happening during the actual
enactment of the SE project. Therefore, a tool supporting the SE process should be able to deal with dynamic situations to a certain extent.

We have already shown that context awareness constitutes a crucial factor of any comprehensive support in a dynamic discipline such as SE (cf. Chapter 4). However, the relating requirement (R:ContInt) remained rather generic and abstract. To be able to specify more concretely what a tool should be capable of in this context, we now detail and concretize this requirement into two sub-requirements. These constitute the basis for more advanced contextual capabilities of a tool.

- **Requirement R:ContInt:GathCont (Gather contextual information):** To ensure access to contextual information a tool should incorporate some means to automatically gather and integrate information from various sources in its environment. This should be possible in an automated fashion and without disturbing humans or other tools.
- **Requirement R:ContInt:ProcCont (Process contextual information):** Contextual information must be processed to gain viable information from the environment. Thus, a tool should provide facilities to process the rough contextual data to make it usable for providing support in the process.

### 6.2. Framework Components

This section describes the components of the framework we develop to cover the various requirements of SE process enactment support. This includes an abstract architecture and technology decisions as well as a rough outline of the developed framework. We denote this framework as CPM (Context-aware Process Management) framework and will use this term throughout this work. The basic idea is to build a framework that fosters synergies by integrating different project areas and technologies. For this purpose, a number of core components is developed as well as a set of integrative components utilizing external technologies. Figure 6-1 gives an overview about the different components of our framework.

![Diagram of CPM components](image)

The central components of the CPM framework are the context management and the process management components. The latter integrates and enhances a WfMS to enact the different workflows belonging to a SE process. In turn, it is encapsulated by the context management component that adds high level functions like complex exception handling (cf. R:Exc and Chapter 11) and manages the interactions among the other CPM components. To be able to manage and store high level semantic information about a project, the context management component integrates a data store called project context. Such information can be gathered from the environment in two ways: on one hand, it can be gathered by interacting with humans by a set of user interfaces, on the other, one may acquire events...
from other tools used in an SE project. Such event management is done by a separate event management component. The latter, in turn, integrates explicit components for acquiring, storing and processing events (cf. R:ContInt:GathCont, R:ContInt:ProcCont).

The context management component integrates two further supportive components that support the course of SE projects: for standardized recurring situations (cf. R:AutoProc:Auto) a rules processing component and for dynamic situations not foreseeable (cf. R:AutoProc:DynSit) a multi agent system component is provided. Finally, as two of the most important assets of SE projects are the created software and its quality (cf. R:Qual) as well as the specific knowledge to produce it (cf. R:Know), we integrated two explicit components managing quality and knowledge.

For these components, we have integrated a set of technologies like for example, a WfMS or a multi-agent system into the framework. We have chosen these because their properties suit the elicited requirements well. In the following, we discuss the components and justify the selection of the used technologies beginning with a list of all components and their responsibilities in Table 6-1.

<table>
<thead>
<tr>
<th>Component</th>
<th>Type</th>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context Management</td>
<td>Core</td>
<td>Central coordination of other components</td>
</tr>
<tr>
<td></td>
<td></td>
<td>High level workflow governance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>User management</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Activity management</td>
</tr>
<tr>
<td>Process Management</td>
<td>Core</td>
<td>High level workflow governance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Automated access to WfMS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Transfer of User Management to Context Management</td>
</tr>
<tr>
<td>Event Management</td>
<td>Core</td>
<td>High level event coordination</td>
</tr>
<tr>
<td>Quality Management</td>
<td>Core</td>
<td>High level quality integration</td>
</tr>
<tr>
<td>Event Acquisition</td>
<td>Integrative</td>
<td>Gather environmental events</td>
</tr>
<tr>
<td>Event Processing</td>
<td>Integrative</td>
<td>Process / aggregate events</td>
</tr>
<tr>
<td>Event Storage</td>
<td>Integrative</td>
<td>Provide and store event information</td>
</tr>
<tr>
<td>Project Context</td>
<td>Integrative</td>
<td>Data storage for high level project data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Inference / reasoning over data</td>
</tr>
<tr>
<td>WfMS</td>
<td>Integrative</td>
<td>Workflow governance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Workflow enactment correctness</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Workflow adaptivity</td>
</tr>
<tr>
<td>Knowledge Management</td>
<td>Integrative</td>
<td>Provide and store user relevant knowledge</td>
</tr>
<tr>
<td>Rules processing</td>
<td>Integrative</td>
<td>Provide configurable automatisms</td>
</tr>
<tr>
<td>Multi-agent system</td>
<td>Integrative</td>
<td>Provide dynamic situation support</td>
</tr>
<tr>
<td>Integrated user interfaces</td>
<td>Integrative</td>
<td>Provide workflow governance to users</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Provide modeling and configuration to users</td>
</tr>
</tbody>
</table>

Central to the framework is the context management component. Its main responsibilities are to coordinate functionalities provided by other components as well as to integrate process enactment contextually with the SE project itself. The latter is achieved by the unification of process enactment that is managed by the process management component, and the project context that stores and manages high level context information of the project. That way, additional information can be connected to the processes as required in R:AutoProc:SEPpro. As a consequence, the context management component is in charge of high level workflow enactment, meaning the incorporation of human and activity management as well as contextual information necessary for the process. The direct technical governance of the executed workflows, however, is handled by the process management component. The concrete concept for integrating these two modules is explained in Chapter 7. The different components and their connections are shown in Figure 6-2, and their collaboration is detailed and exemplified in procedures and examples following Figure 6-2.
The project context is utilized for the high-level management of context information relating to the project. For this purpose, high-level domain specific information shall be gathered, stored and processed automatically. Ontologies have been chosen to facilitate this, as justified in the following. [UsGr96] provides a definition of the term ontology:

“‘Ontology’ is the term used to refer to the shared understanding of some domain of interest which may be used as a unifying framework... An ontology necessarily entails or embodies some sort of world view with respect to a given domain (e.g., entities, attributes, processes), their definitions and their relationships; this is referenced to as a conceptualization.”

Ontologies are used in information processing for providing a machine readable semantic knowledge store. The semantic web [DSW06] constitutes one of the most widespread installments of ontologies that enable the addition of more semantics to the world wide web. Semantic web technology provides opportunities for logical information management and sharing. According to [GDD06], the capabilities of ontologies provide a vocabulary for the modeled entities including taxonomies and logical statements about the entities. Furthermore, they offer the capability of reasoning about the contained data and inferring new facts. Finally, they enable enhanced interoperability between different applications, extending reuse possibilities, and the option for advanced content consistency checking.

In addition, semantic web technology is the most prevalent and widespread semantic technology [Fell13]. Consequently, semantic web technology has been chosen for this work. In the following, major advantages of this technology are explicitly listed:

- **Standardization**: Semantic web technology is standardized, i.e., it is openly accessible. The World Wide Web Consortium has published standards relating to it. Examples include languages for logical information storage and processing [WWW04a, WWW04b, WWW04c].
- **Maturity**: Semantic web technology is rather mature. Many vendors provide mature tools and related applications.
- **Interoperability**: Semantic web technology promises good capabilities for interoperability. This could be beneficial in the future for sharing acquired context knowledge or for integrating other knowledge. Examples for semantic descriptions with semantic web languages are web services with ontologies like WDSL [CrMu03] or OWL-S [CDL+04].
- **Knowledge Storage**: Semantic web technology provides a means for storing and structuring information.
- **Logical Information Processing**: Semantic web technology incorporates features for logical processing of the stored information based on description logic [BCM+07]. There are various mature implementations of semantic reasoners for this technology, including Pellet [SPG+07], HermiT [SMH08], and Fact++ [TsHo06]. These not only provide a means to answer questions about the information contained in the ontology, but also allow inferring new facts based on others.

The project context unites information about the environment that is automatically acquired by the event management component as well as extended information about the executed workflows.

The process management component is utilized to enable governance of the executed workflow instances. This is achieved by integrating a WfMS that is managed by the process management component. It has been decided to integrate such a tool because of its features concerning process governance and support [vtW03]:

- **Business process modeling**: that captures the steps taken to represent an abstract business process by means of an explicit business process model.
- **A process language is used to model, represent and enact a business process or a respective workflow.**
- **Business process analysis**: This aims at investigating various properties of the modeled processes. Examples include the verification of the models or the simulation of the processes.
- **Process enactment**: A WfMS provides capabilities to correctly execute workflow instances.
The process management component accesses the API of the integrated WfMS to provide standard functions like the creation of a new workflow instance. To accommodate the elicited requirement regarding dynamicity of workflows (R:DynProc), the possibility to adapt running workflow instances becomes essential as well. This enables the CPM framework to adapt workflow enactment to changing situations, e.g., to apply automated handling procedures for exceptional situations. Therefore, we have integrated an adaptive WfMS. Some additional properties of such a tool are listed in the following [WSR09]:

- Dealing with unanticipated changes: Such changes require the adaptation of a workflow instance. This should not affect other instances or the model.
- Support for changes at a high level of abstraction: This feature deals with high level change patterns. It enables for example, the insertion of a new activity between two others instead of multiple low level operations like ‘add edge’ or ‘add node’. This supports understandability and correctness of the changes.
- Correctness of ad-hoc changes: Ad-hoc changes should not leave any workflow instance in an inconsistent state. Therefore, a dynamic WfMS guarantees the structural and behavioral soundness of the workflows before as well as after the adaptation.
- Controlling concurrent changes: In a life system, multiple adaptations to the same model or instance could occur. Therefore, the WfMS has to guarantee that these changes do not interfere or introduce inconsistent states.

The requirement concerning dynamicity (R:DynProc) and the concept to accommodate it is handled in Chapter 7. The combination of the process management component, the project context and the context management component is used to achieve automated SE process support (R:AutoProc:Tool). In turn, the context management component enables human and activity management in alignment with contextual data. Usually, these tasks are handled by a WfMS itself. Therefore, control has to be transferred from the WfMS to the context management component. To achieve this, we apply the in-process component: This component is placed inside the workflows as part of each activity and communicates with the context management component.

The event management component is in charge of high level coordination concerning events occurring in a SE project and being relevant to the framework. These events have to be acquired, stored and processed to be usable for the framework. For this purpose, different technologies are integrated: The event acquisition component is utilized to automatically acquire events from the environment (cf. Requirement R:ContInt:GathCont). Environment in this case means events that occur when humans interact with tools and other humans to manipulate certain artifacts. The approach to acquire such events is to establish a connection to the used tools to get informed about occurring events. Two ways of realizing this are taken: In certain tools (as e.g., source control management systems), sensors are integrated that automatically generate events for the framework. For other tools that provide external interfaces (as e.g., web services), a polling approach is taken.

The acquired events do not necessarily have much meaning for the context. For example, the switching to the debug perspective in an IDE (e.g., Eclipse) may not have much meaning. Therefore, the acquired events can be combined and aggregated (cf. R:ContInt:ProcCont) by the event processing component. That component integrates CEP (Complex Event Processing) [Luck01] technology to support the creation of high level events having semantic value out of multiple atomic low level events. These events, no matter if high or low level, should be accessible to other components of the framework. Therefore, the event storage component integrates storage technology to allow for event persistence. That component is also in charge of a notification mechanism to automatically inform components concerned by occurring events. That way, the components are connected in a loosely-coupled fashion enabling the easy integration of new components if needed.

As quality management is crucial for SE (cf. R:Qual), a separate component is created for it. The latter as well as the context management component have access to other components that provide automation functionalities to satisfy requirements R:AutoProc:Auto and R:AutoProc:DynSit. To provide configurable automatisms facilitating recurring procedures, a component enabling automatic rule processing is used. That technology suits that purpose well as it offers features like declarative

programming, logic and data separation, an explanatory facility, understandable rules, flexibility, and adaptability [Rudo08, Brow09].

Dynamic situations involving multiple parties with conflicting interests that cannot be rigidly pre-planned are inherent to SE projects (cf. R:AutoProc:DynSit). Therefore, we needed to integrate a component or technology capable of dealing with such situations. Furthermore, it should be possible to apply this in a standardized and robust way. Because of this, during the course of this work, we opted for the integration of a multi-agent system because agents have the following properties [FrGr97]: reactivity, autonomy, proactivity, responsibility, interactivity, adaptability, rationality, cooperativity, and robustness (cf. Chapter 9 for details).

Another crucial factor for the success of SE projects is knowledge management (cf. R:Know). Therefore, a framework aiming to comprehensively support SE projects should integrate knowledge management facilities. In our approach, a separate knowledge store component is integrated to store and provide human-relevant knowledge. To facilitate automated support for knowledge dissemination and interweaving of process enactment and relevant knowledge, the knowledge store is connected to the context management component as well.

Finally, to foster communication with humans, the developed solution bears integrated user interfaces of two different types. On one hand, there is a GUI (graphical user interface) component for enacting the workflows such that human tasks can be automatically distributed to the respective humans. That component must easily integrate with the standard procedures of the humans to smoothly integrate process support with every day work. On the other, GUI components are required for different modeling tasks, like the specification of processes or automation rules. These shall be as easy as possible and support the humans during modeling.

![Figure 6-2: Interaction between CPM components](image-url)
The responsibilities and interactions between the different components is explained in the following. We therefore list the different actions of the different components in procedures and illustrate these actions in examples. Further, we refer to the numbers from Figure 6-2 in this context. The procedures do not cover all possible communication paths and functionalities, but incorporate every component at least once to illustrate how they are used. The Procedure 6-1 deals with events acquired and processed by the framework (cf. Figure 6-3).

**Procedure 6-1 (Event management):**
1. Event acquisition: An external event is detected by the event acquisition component (cf. Figure 6-2: 0 and 1).
2. Event Storage: The detected event is automatically captured in the event storage component (3).
3. Event Processing: The event processing component, which is automatically triggered, processes the event and, if it is detected as part of a complex event, creates a new event (2).
4. Event Storage: The newly created event is stored in the event storage component (3).
5. Context Storage: If the event is relevant to the context, the context management component adds the new knowledge to the ontology (5).
6. Event classification: The reasoner gets triggered (6) and eventually adds new knowledge to the ontology (7) by further classifying the event.

![Figure 6-3: Event management](image)

Example 6-2 elucidates Procedure 6-1 with a concrete situation:

**Example 6-2 (Event management):**
Assume that a data acquisition facility (a sensor) for a source control management system like CVS or SVN has been established. Thus, the sensor registers the check-in of source code files and produces an event, which is then stored in the Event Storage component. For that event, no further complex event processing becomes necessary. Yet, this event might be relevant for the context. Hence, the context management component is informed adding that knowledge to the ontology. The event is further classified using context knowledge: For example, it could be recognized that a human checked in newly developed source code files and that he should, according to his workflow, have created and run some additional unit tests before checking in. Omitting these activities, he has deviated from the workflow that was prescribed by the process. Hence, the event could be classified as ‘Process Deviation Event’. The application of this example is detailed in Chapter 11.

Procedure 6-2 shows standard workflow enactment as it is automatically coordinated by the framework in alignment with the project context. The procedure is followed by Figure 6-4 and Example 6-3 illustrating it. Both procedure and example describe automatic workflow instantiation by the framework. Yet manual workflow initiation by humans is possible as well.

**Procedure 6-2 (Integrated workflow enactment):**
1. Context Storage: The context management component receives an event from the event management component (4).
2. Decision / Reasoning: The context management component uses the new knowledge for decision making and decides that a new workflow shall be instantiated (6,7).
3. Information gathering: The context management component uses context knowledge to determine when and for whom the workflow shall be applied (5).
4. Workflow initiation: The context management component informs the process management component that a particular workflow shall be instantiated (8). The latter component uses the WfMS to create a new workflow instance (9).
5. Activity activation: As the workflow instance is initiated, one or more activities are activated. This also activates the in-process component(s) (10) being used as implementations for those activities. The in-process components inform the process management component (11) as well as the context management component (8) about activation of the specific activity.
6. Task distribution: The context management component distributes the task information to the GUI component of the respective human (12) that, in turn, displays the task information (13).

Example 6-3 (Integrated workflow enactment):
The context management component receives information about a coding activity by some human. Using context knowledge in the ontology, the component determines that the change to these specific source code files requires the adaptation of integration tests. Subsequently, the component determines that a testing team is responsible for these tests in the current project. Afterwards, the process management component initiates a workflow for adapting the tests whose tasks are then routed to be displayed in the GUI of the respective tester by the context management component. The application of this example is detailed in Chapter 10.

Procedure 6-3 deals with knowledge management and knowledge dissemination automation and is also followed by Figure 6-5 and Example 6-4 for illustration.

Procedure 6-3 (Knowledge integration):
1. Information storage: Humans enter information relevant to the project in the knowledge collection and management GUI and tag it (14).
2. Process enactment: Workflows are executed as part of the SE process.
3. Correlation detection: The context management component is aware of points in the process where the usage of additional knowledge could be beneficial for humans and the knowledge provider detects a correlation between currently executed activities and knowledge being present in the knowledge store (15, 24).
4. Information retrieval: The knowledge provider retrieves relevant knowledge from the knowledge store (24).
5. Information injection: The context management component attaches the acquired knowledge to an activity. That knowledge is then displayed to the human by the GUI for enactment (12, 13).
Example 6-4 (Knowledge integration):
Humans enter information into a company’s knowledge management system about specific backend development tasks like hints how to work efficiently with database cursors. A human in a backend development team works on a coding activity. This activity has an attached dynamic checklist to aid the human. The tool recognizes the connection between the information and the activity and compiles a checklist matching the current situation of the human. That checklist is automatically displayed as part of the process enactment GUI. The application of this example is detailed in Chapter 12.

Procedure 6-4 deals with the connection of quality management and workflow enactment and is followed by Figure 6-6 and Example 6-5 for illustration.

Procedure 6-4 (Quality integration):
1. Problem detection: An external problem is detected by the framework: An event is received from an external tool (cf. Figure 6-2: 0 and 1); the event is processed and classified as problem event (4-7).
2. Initial measure allocation: If the problem related to the quality of the produced software, the quality management component utilizes the integrated rules processing component to automatically assign a quality measure to it (21).
3. Work status detection: A quality measure must be applied by a human. That human must have time for it such that other planned activities do not get delayed. Therefore, the context management component observes process enactment for situations when someone could apply a quality measure without delaying his future activities (5,8).
4. Measure selection: When such an opportunity is found, a measure can be applied. Usually there is not enough time in a project to treat every problem and thus the most effective quality measure should be applied. A selection procedure is applied by the multi agent system and the quality management component: The former prioritizes the available measures according to the different quality goals of the project (22) and the latter finally selects the measure best fitting to the current human context utilizing information obtained from the context management component (23).
5. Measure application: The selected quality measure is integrated into the humans’ workflow by the process management component and the integrated WfMS (8, 9).

![Figure 6-6: Quality integration](image)

Example 6-5 (Quality integration):
A sensor is applied in a static analysis tool used to conduct measurements on the source code. That sensor generates an event containing values of metrics like McCabe’s Cyclomatic Complexity [McCa76]. When the value of a metric exceeds a defined threshold, it is considered a problem and a measure is assigned to it (e.g., to refactor the concerned code). Meanwhile, during process enactment, a human finished one activity earlier than expected. The framework perceives this as an opportunity for a quality measure application. Assuming that quality goals like ‘Maintainability’ and ‘Reliability’ are important, the aforementioned problem with the cyclomatic complexity could be selected for application. Subsequently, the refactoring of that respective source code artifact would be automatically integrated in the humans’ workflow.

Besides these components and functionalities, GUI components allow humans to define various concepts for the context and process management component. (16, 18).
6.3. Discussion

This section reviews approaches that seek to provide automated support for the SE process. It further shows how automated SE process support evolved from the mid-eighties until now.

6.3.1. Computer-Aided Software Engineering

Computer-Aided Software Engineering (CASE) is a discipline that seeks to improve software engineering and dates back to the 1980s [CNW89, Case85]. There are many definitions of CASE, which are more or less similar, e.g.:

"Computer-Aided Software Engineering (CASE) encompasses a collection of automated tools and methods that assist software engineering in the phases of the software development life cycle." – [Sodh91]

"CASE tools are computerized software development tools that support the developer when performing one or more phases of the software lifecycle and/or support software maintenance." – [Gali04]

According to these definitions, CASE has two purposes: Supporting various SE activities based on automated tooling and integration of different tools to comprehensively support the phases of the SE process or even the entire SE process. [Fugg93] classifies CASE tools into three categories:

- CASE tools: "A CASE tool is a software component supporting a specific task in the software-production process." This category comprises different types of tools: graphical or textual editing tools, programming tools (e.g., compilers), verification and validation tools (e.g., test-case generators), configuration management tools (e.g., source control systems), measurement tools (e.g., static code analysis tools), and project management tools.

- "CASE Workbenches: Workbenches integrate in a single application several tools supporting specific software-process activities." Thus, several related tools can be combined with a common presentation and data set as well as easy invocation facilities. Different categories of these workbenches can be recognized. These are business planning and modeling, analysis and design, GUI development, programming, verification and validation, maintenance, configuration management, and project management.

- CASE Environments: "An environment is a collection of tools and workbenches that support the software process." Different types of environments with different properties exist: Tool-kits are a set of loosely integrated tools. A higher level of integration is provided by integrated environments that encompass facilities to provide tighter tool integration, e.g., a common presentation or data layer. A special class of environments are the process-centered environments that not only integrate tools but also put strong emphasis on the SE process.

The integration of tools has different dimensions [Wass90]:

- Platform integration: This type of integration deals with the transparent provision of services to applications (e.g., networking services).

- Presentation integration: To enhance usability, tools may share a common presentation layer enabling the human to execute all tools from a single environment.

- Data integration: Real integration of tools can only be established if the tools work on a common data repository.

- Control integration: This type of integration enables tools to automatically exchange notifications among each other. It is also crucial for real integrated environments.

- Process integration: This concerns the alignment of the tools along an SE process model. This was not achieved by classical CASE, since at that time, no mature process enactment support via tools was prevalent.

CASE has primarily focused on the integrated support of different activities. This section gives a brief overview about two CASE approaches in the following.
GOODSTEP

The GOODSTEP project [EKS93] developed a CASE environment based on object-oriented database systems (OODBMS). The focus of GOODSTEP is to provide inter-document consistency for the SE process. That way, things like function names should match, for example, in design and implementation documents, and change propagation is managed by the system. In GOODSTEP, documents are managed as abstract syntax graphs. Manipulation is done via functions operating on the graphs. A query language is provided for navigating within the graphs. To provide concurrency control, GOODSTEP divides each user session into multiple transactions for short human interactions. Thus, concurrency conflicts are minimized and change propagation to other humans is supported.

Software through Pictures

The commercial CASE environment Software through Pictures (StP) provides a set of integrated tools on top of a common model [Aon97]. Communication of the tools is implemented by a shared repository and an object management layer. StP features an open architecture to enable easy integration of new functions. All tools share a common GUI including various editors and notations. The environment integrates various facilities for consistency and completeness checking to assist the humans. The shared repository is implemented on top of a relational database and can be accessed through a set of object management routines and a data manipulation language.

Summary

CASE tools are not suitable for satisfying the requirements of modern SE projects. They feature facilities for integrating different tools and data sets. However, they fall short in providing more complex support features regarding the SE process. This concerns all elicited requirements because they are all centered around activities, humans, and context.

6.3.2. Process-centered Software Engineering Environments

Process-centered Software Engineering Environments (PCSEEs) are integrated environments supporting the SE process. As opposed to plain CASE environments, they not only integrate and support different phases of the SE process, but are also built around a facility to model and enact the whole process explicitly. A representative set of PCSEEs is described in the following:

SPADE

“The SPADE project [BFGL94] aims at defining and developing a software process-centered environment to describe, analyze and enact software process models.” It is a reflective approach that is built around three main components to provide a strict separation between process model interpretation and human interaction. A human interaction environment manages communication to humans via integrated tools. A process enactment environment is responsible for enacting process models created in SLANG (SPADE LANGUAGE), a Petri nets language. The third component is a filter component that manages the communication of the two environments.

Process modeling in SLANG has three basic features: Processes can be hierarchically structured using process fragments called activities. These activities are capable of mutually manipulating each other. All process data is typed and represented as a token in the process, whereas a taxonomy of predefined and human-defined data types classifies that token, e.g., as ‘test case’. Besides such basic features for process enactment, SPADE incorporates basic features for process evolution. The latter is accomplished by a meta-process, enabling the change to process models.
EPOS

“EPOS [CHL+94, CLH95] is a kernel Software Engineering Environment” that focuses on process evolution and provides a set of key features: Process modeling is done in an object-oriented way and supported by different types of rules (i.e., meta, static, and dynamic rules). The process models support customization and evolution. EPOS supports cooperative work through overlapping transactions. The EPOS architecture consists of four layers. A data management layer (EPOSDB) offers a sophisticated transaction model capable not only of long-running and nested transactions, but also of overlapping and cooperating transactions. A process modeling layer comprises functionalities to support process specification. A tasking layer manages the enactment of processes. The latter also features a planner that dynamically generates new sub tasks for composite tasks by an AI non-linear planning algorithm. A process model layer contains tool-related task types and knowledge about tool enactments. EPOS also features a project context to enable the management of changes to all objects associated to a project and project interconnections. The management of these interconnections is supported by the project manager component.

ADELE-TEMPO

Adele-Tempo [BEM94] builds upon Adele, which initially was a source control system [EHK84] that has been extended in several ways. On top of that, process management concepts have been implemented, called Tempo. Same as most other PCSEEs, Adele-Tempo is based on an advanced data management component that includes configuration management facilities. This is extended by an activity management component that features events and Event-Condition-Action rules. That component realizes a trigger mechanism for actions, which are programs defined in an Adele specific language. The Tempo component, in turn, was developed to allow for support of the high level SE process. The process concept is built around the concepts of role and connection. A role is used to define the properties of an object such that their behavior can be changed depending on the process they are used in. Process collaboration is defined by the connections. Processes are separated into three layers: (1) a software process model that contains several software process types; (2) a software process type is a set of activities and can be hierarchically structured; (3) a process instance, in turn, is a concrete occurrence of such a process that is processed by one or more humans with certain tools in certain work spaces. Tempo supports the collaboration of different processes by collaboration roles like, for example, notification.

ALF

“ALF [CBD+94] applies Knowledge Based Systems and advanced Information Systems techniques to Software Engineering Environments (SEE).” ALF is built on top of the object management system PCTE [BGMT89] and supports SE process in various ways by providing an open configurable framework for SEEs. Furthermore, it comprises different models for operations, objects, rules, and ordering. Process models, however, can also be enacted. Therefore, they are instantiatted meaning they are transferred into an associated work context containing object and tool instances. ALF features basic features for collaboration as well. Various agents can work together in one context and access the same objects. Finally, basic facilities for process evolution are also provided including some limited effect on running work contexts.

ALF has a language for each of its models. The object-model description language is derived from PCTE’s data definition language and enables the definition of structures and relationships between objects. The rule model description language allows for the definition of ECA rules. The ordering model is made up of annotated path expressions.

Merlin

“Merlin [JPSW94] is a prototype Process-centered Software Development Environment (PSDE)...”. Merlin follows a rule based approach and provides the human with a special GUI that displays the
documents to be manipulated, their dependencies, and the required activities that contextually belong together. A Merlin process description consists of the following base elements: activities that are carried out to produce a software product, roles for structuring the activities, the various documents belonging to the SE process and the participating humans, called resources.

Merlin's rule-based process enactment approach is based on preconditions for all activities. The process engine evaluates these preconditions and updates all working contexts accordingly to distribute the activities to the humans. The former is capable of providing the human with additional contextual information on the activities, like other involved humans or time constraints. For modeling of the processes, Merlin combines the rules with Extended-Entity-Relationship (EER) diagrams and finite-state-machines. The EER diagrams are used to define the concrete documents and their relations and each document has rules and finite-state-machines as properties. The state of a document is modeled as a finite-state-machine, whereas the rules are utilized for status transitions.

MARVEL

MARVEL [Barg92a, Barg92b] is a tool belonging to the class of rule-based PSDEs with a strong focus on supporting collaboration and coordination among software developers. The data model containing all product and process data is based on an object-oriented schema. Processes are specified utilizing the rule-based MARVEL Strategy Language (MSL), whereas each process step is represented by a rule. For process enactment, MARVEL provides forward and backward chaining over the rules. When a human enters a command, MARVEL selects the best fitting rule and invokes it. If the rule condition cannot be satisfied, backward chaining is attempted. If it is satisfied, forward chaining with the effects of the invoked rule is applied.

To support concurrency and coordination, MARVEL employs different concepts: On the one hand, a transaction manager can group operations into transactions. If operations interfere, a so-called Semantic Concurrency Control Protocol is used to detect and resolve them. On the other hand, a Coordination Rule Language is put in place to explicitly model coordination between team members.

Summary

PCSEES offered some interesting options for holistic SE process support. However, they could not rely on today’s advanced process technology. Processes were realized with technologies that, in the meantime, have not proven to be suitable in this context, like rules or Prolog. Process specifications were often complex and cumbersome and the configuration, usage, and maintenance of the environments involved substantial efforts. Furthermore, none of them could satisfy all requirements elicited in Chapter 4. On one hand, most environments only support a selection of these requirements (e.g., collaboration). On the other, the support for them is only basic. Finally, despite providing facilities for process evolution, PCSEEs do not provide enough features to model and enact dynamic SE process.

6.3.3. Modern Development Environments

Recently other approaches for supporting SE projects have emerged. This section gives an overview on contemporary SE environments.

DOSDE / EOSDE

The fact that specific knowledge is important for most human-oriented process enactment is favored by the approaches provided in [dZR+04], [ZdR02], and [VdS03]. In particular, the authors focus on knowledge regarding the domain and the specific enterprises where the process is executed. Thus, they use the terms of Domain-Oriented Software Development Environment (DOSDE) and Enterprise-Oriented Software Development Environment (EOSDE). To incorporate and support specific task knowledge, a task ontology as well as a problem solving method (PSM) is incorporated. The former
provides a vocabulary to formally describe a problem solving structure while the latter emphasizes the concrete solving of the problem by decomposing it into smaller problems. Both components are combined into a single model called Problem Solving Theory (PST). The authors have created a concrete DOSDE called Cordis. Further, they extended the latter to an EOSDE called Cordis-FBC, which not only incorporates specific task knowledge but also information about the enterprise context of the process. Cordis-FBC contains a set of dedicated knowledge components for SE knowledge, generic task knowledge, specific company knowledge, and domain knowledge.

**CASDE**

The focus of CASDE [JYW07, JYWF06] lies in supporting the concrete activities performed by humans and their collaboration. It builds upon Activity Theory and thus incorporates different factors influencing the software produced by an activity: the human, its motive, the used tool, and the team, in which the individual works. CASDE distinguishes three levels of cooperative activities. The coordinated level describing the routine activity and their interactions, the co-operative level, where humans focus on a shared problem, and the co-constructive level meaning a set of complex interactions where the participants reconceptualize their own organization and interaction. Technically, CASDE is realized as a client-server architecture. The clients feature a set of UIs that can be customized. Legacy tools can be integrated into the architecture by specific wrappers. Artifacts are stored in different repositories that are monitored by an awareness module. A process module governs the operations of the supported projects, and a communication module supports classical communication between the participants.

**Syde**

Syde [HaLa10, HaLa09] is an environment based on an extended view of source control systems. It enables synchronous development, meaning that every developer can be informed about any changes other developers make even if not checked in. This is achieved by representing an object-oriented system as an abstract syntax tree (AST) and the changes to the system as tree operations. That way each change operation is explicitly stored on a centralized server and the system is evolved by a set of change operations, i.e., not by source code versions. Syde features a central repository for the changes and distributes them to all developers informing them about potential conflicts just as they appear.

Syde is realized as a client-server architecture where the clients are represented by a set of Eclipse plug-ins. One of these collects information about the source code translates them to change operations and stores them on the server. A second plug-in features a set of different visualizations to enhance awareness of developers: a decoration of the package explorer to visualize changes, an aggregated view of the changes made to classes, and a view displaying the effort spent on each class. Finally, a third plug-in displays conflicts of the developers local copy with the rest of the system.

**SPACE**

The Semantic Process- and Artifact-oriented Collaboration Environment (SPACE) [WEB+09] primarily aims at artifact-centered support. It comprises interconnected process and artifact models. The former is intended for modeling and enactment and thus contains process models as well as instances. It enables the creation of personalized views on the process to support humans in their concrete tasks. Processes may be semantically annotated on the model and instance levels. This enables, for example, traceability for the artifacts applied and used in the process and realize a tight integration to the artifact model. The latter enables, same as for the process model, personalized views on artifacts and artifact structures to aid humans.

SPACE is a generic approach that was adapted to the SE domain by combining it with another approach of the authors, the Software Organization Platform (SOP) [WTA+08] that emphasizes the following features: lifecycle artifact and process management supporting the creation and management
of artifacts, knowledge management providing personalized knowledge support, and stakeholder collaboration supporting collaborative development of artifacts.

**ADAMS**

The ADvanced Artefact Management System (ADAMS) [dFOT10] is a project support tool focusing on artifact-centric support as well. Projects are defined in ADAMS in terms of their artifacts, whereas it enables complex artifact hierarchies and compositions as well as sophisticated versioning and locking approaches for the artifacts. Furthermore, it enables fine-grained artifact traceability by defining traceability links between artifacts. ADAMS can actively inform software engineers about changes on artifacts by event subscription and notification facilities. ADAMS comprises a set of sub-systems that enable different functionalities relating to the managed artifacts. It provides administrative functionalities regarding human resources like allocation of the latter for a project and to concrete artifacts. Further, ADAMS allows for product-oriented work breakdown structures. Finally, ADAMS is capable of supporting quality of the managed artifacts with checklists and collaboration of involved humans with synchronous and asynchronous communication between project members.

**Transforms Environment**

The Transforms environment [MdMR09] was created to support MDA (Model-Driven Architecture) processes for software development. Therefore, the approach it is based on is situated on the M2 level of the OMG model layers and uses parts of the SPEM model [OMG08] and tailors them for MDA processes. These concepts are explicitly described, like for example, concepts for different models like a CIM (Computational Independent Model), PIM (Platform Independent Model), or a PSM (Platform Specific Model).

To make the approach useable for SE projects, the Transforms environment supports their modeling and enactment. This is achieved by a set of graphic editors supporting the modeling and specifying of the processes and to automatically generate diagrams for specifications of the models. The environment also features editors for model-to-model and model-to-text transformation languages.

**Model-driven Approach to Software Process Variability**

The approach described in [AFdK11] aims at supporting deployment and variability of SE processes by applying a model-driven procedure. In particular, a way of integrating variability into process modeling is shown that incorporates the automatic generation of a customized process specification. Furthermore, the automatic transformation into models enactable by a workflow engine is enabled. The approach is founded on the principles of software product lines and model-driven engineering.

The approach comprises a set of different steps implemented with different technologies to enable that: The first step is process modeling. After that, a variability model is created comprising all variation point in the process. The two models are then used by the GenArch [CKd07] product derivation tool to create customized process specifications. The latter are then automatically transformed by model-to-model transformation into executable specifications. A further model-to-text transformation is then conducted to generate customized web forms to be also deployed and executed.

**VRPML**

The Virtual Reality Process Modeling Language (VRPML) [ZIK05, ZaLe03] is a visual process modeling language targeted at SE processes. It allows specifying SE processes by modeling process step abstractions that represent concrete activities a software engineer has to perform. Various activity types are supported, e.g., rather generic activity types like a multi-instance activity node as well as relatively specific ones like a meeting activity node. For each of these activities, a separate workspace is created. VRPML enables the dynamic allocation of resources to planned activities through an
enactment model. This comprises a resource exception mechanism that enables it to dynamically manage the availability of resources. The authors have also developed a concrete support environment for VRPML including various components like a Graph editor for specifying VRPML graphs, a runtime interpreter for enacting the graphs, a workspace manager for managing activity workspaces in a virtual environment, and a resource manager for retrieving artifacts out of a database.

Summary

Numerous contemporary approaches regarding SE environments exist. However, they neither aim for nor achieve real holistic SE project and SE process support. They rather target a specific area like developer collaboration or artifact-centric support. For example, DOSDE/EOSDE strongly focuses on different kinds of knowledge, while neglecting other important aspects like automatic SE process support (cf. Requirements R:AutoProc, R:DynProc, R:ProcCoverage) or contextual integration (cf. Requirement R:ContInt). Actually, it leaves much to the humans and, thus, automatic support and seamless integration of other important aspects like quality management remains an issue (cf. Requirement R:Qual). The same issues apply to the following environments: CASDE focusing strongly on collaboration or Syde, SPACE, and ADAMS target at artifact-centric support. Key issues that remain are always a holistic view on the projects, SE process support, and contextual integration. However, the other discussed approaches (the Transforms environment, the model-driven approach, and VRPML) have other issues. They focus primarily on the process and neglect other aspects of the SE projects like context integration or quality management.

6.3.4. Other Contemporary Approaches

Recently, there have been approaches apart from SE environments targeting some kind of support for SE projects. There even exist production ready platforms from major vendors exist to facilitate SE project success. This section reviews these platforms as well as current SE trends like global SE, mash-up environments, SE recommendation systems, social media in SE, and cloud-based SE support.

Existing Tools / Platforms

Software vendors are aware of SE projects needs and have built solutions to support these projects in a more holistic way. Microsoft’s Team Foundation Server (TFS) [BWHK12, MTM+07] integrates directly with their Visual Studio IDE and offers a set of additional features for holistic SE project support. These include advanced source control facilities, agile task management practices with task boards or residue management, and an integrated web interface for access to team support functionalities. Furthermore, it enables comprehensive human task management. Tasks can relate various items like problems, exceptions, or test cases. With its data warehousing functionalities, it offers access to many kinds of information around the project.

These features are offered though the integration of the main component TFS with the IDE. IBM takes the challenges of SE projects by another approach: the development of the open platform Jazz [Stan11, Fros07]. The latter is built to offer comprehensive integration facilities for tools built on top of it. With these tools, holistic SE project support is targeted. Examples of such tools include Rational Team Concert, which is an integrated environment for all project participants with work item management, source control and process management. The Rational Quality Manager is a test management environment enabling test planning, enactment, and reporting. The Rational Asset Manager is a tool enabling the management and reuse of any asset relating to SE. Such tools offer comprehensive and holistic support facilities, however they are mostly passive and lack facilities for active guidance of the project participants.
Global Software Engineering

Global SE is a recent trend in SE induced by globalization. Software developments teams are often distributed with participants coming from different counties, continents, cultures, or time zones. Better supporting such teams and overcoming issues related to this distribution is the goal of global SE.

[LEPV10] reviews different collaboration platforms for distributed development like SourceForge or Rational Team Concert. In a detailed comparison, different features of such platforms, like their communication tools, build tools, or knowledge centers are shown. In a conclusion, the authors state that no tools currently support all aspects of global SE. In [PVPB12], a systematic literature review is conducted revealing a total of 132 tools intended for global SE. Findings of the article include that most of the tools are in the categories of virtual meeting tools, SE management tools, and knowledge management tools. Furthermore, the most common features provided by the tools are awareness of the actions of the team members, social aspects, interoperability of tools, communication support, and knowledge management. The authors state that a comparison is difficult and that there is a lack of connection between tools.

As global SE is still a relatively new trend, [SWGF10] reviews empirical studies dealing with this topic. Findings of this article are the following: global SE is still immature. The biggest problem seems to be geographical, temporal, and cultural separation. While additional investments would be required to overcome these problems, companies are still driven by cost reduction, which was also one of the main drivers of global SE. [RCBM10] also identifies geographical, temporal, cultural and linguistic distance as the biggest problems of global SE. Furthermore, they highlight the importance of a team-based approach for any SE project. They intend to integrate global SE into the SE process. Therefore, they create a SE process called Global Teaming for global SE that includes a set of specific practices suited for this domain. In [CED10] another approach called “Follow the Sun” workflow is taken. The core idea is that in globally distributed team, the work of one team is handed off at the end of a day to another team whose day is just beginning. The authors develop a conceptual foundation for such practice and, based on that investigate under which circumstances it can be really beneficial by reducing task durations.

As shown by this sub-section, global SE is not yet matured as a science. Most articles review other studies, develop theoretical concepts, or discuss tools supporting one particular aspect of SE projects. Holistic project support is not yet prevalent within this area.

Mash-up Environments

In today’s modern internet environment with various Web 2.0 technologies, an idea of consistently combining multiple applications into one has emerged: the mash-up environments. While there has been some non-scientific approaches, there are also first scientific efforts to develop consistent methodologies for mash-up environments.

The approach shown in [PTR+10] facilitates model-driven development of composite mash-up applications by a platform independent metamodel. It enables the definition of reusable parts of an application, communication between them, and screen layout and flow. It further enables the definition of context-based behavior of mash-up applications.

[GTS10] proposes to adopt the idea of mash-up environments to SE environments. An information mash-up environment for SE is developed, which is capable of uniting different applications containing different kinds of related information. The approach focuses on a lightweight combination of the information of different tools obtained by web services.

Same as for global SE, mash-up environments constitute a recent trend in the scientific community. Not many approaches exist and they are often targeted at surveys or theoretical concepts. If tools are really developed, they only cover a small area of an SE project.
Software Engineering Recommendation Systems

Recommendation systems have been used in many areas for a long time and the idea of using them in SE is not new. [RWZ10] gives a brief summary of such recommendation systems for SE. First a basic definition of such systems is given. After that, possible support features are discussed. The article also reviews a number of recommendation systems from the years 2002-2008 that will be not further covered in this work as it will concentrate more on more contemporary approaches.

[CAG12] proposes a recommendation system for source code related web sites to aid developers when dealing with exceptions in their code. Therefore, information of the exception stack traces is extracted, represented in a graph, and indexed. Concrete exception and method references are identified with regular expressions and the results of the web search are ranked due to their relevance to the structural and lexical content of the exception trace. To implement this approach, a plug-in for the Eclipse IDE was developed directly integrating the search results in the IDE.

[CPPM12] envisions a recommender system supporting open source communities. This system should use a profile for each human to provide personalized recommendations. To acquire this a semantic analysis is conducted on the humans artifacts as well as a quantitative analysis on the humans activities. The system is not yet in place and the authors plan on refining its model and for evaluating it in the open source community.

[VBG12] provides a technique for a recommender system capable of identifying emerging teams in an organization. Using the activity structures of the different project participants it builds social network graphs. Utilizing the latter, recommendations on which humans should work closely together become possible.

[SGWB12] proposes a recommendation system for software developers to be created on top of a tool previously developed by them. The tool shall enable recommendations on bases of discrepancies, for example between software artifacts. The tool shall also incorporate facilities for feedback cycles and learning. As this is a position paper, there is also no system in place yet.

[Denn12] presents ongoing research for a system capable of recommending relevant code artifacts for a given change request. To enable that, a combination of three different information retrieval based prediction approaches are incorporated to unify information about change requests described in natural language and source code. To reduce the number of results of this prediction and to weigh the different predictors, machine learning is applied.

There exists a fair number of scientific approaches targeting recommendation systems for SE. However, all of these systems target a relatively narrow area of an SE projects and thus fail to provide holistic project support.

Social Media in Software Engineering

SE is a knowledge-intensive, collaborative and thus also a social activity. There are first attempts to investigate and utilize social connections between project participants and, in particular social media technology for SE. [BDZ10] suggests research to understand the benefits, risks, and limitations of social media use in SE. It summarizes social and collaborative technologies used in SE and poses research questions on a set of topics regarding community and end user involvements, project coordination and management, and the SE activities. [STDC10] also focuses on investigating the use of social media tools in SE. Therefore, they first name and discuss the various technologies. After that, they categorize the different ways, which social media tools can benefit such projects. Finally, they present some challenges concerning the usage of social media tools in SE. There are first scientific considerations about utilizing social media for SE. However, this area is still in early stages of development. Holistic project support is not prevalent.
Cloud-based Software Engineering Support

Cloud computing is an emerging paradigm that is becoming more and more prevalent. Thus, there are considerations and approaches to make its benefits useable for SE and adapting SE to better integrate with cloud computing to achieve better SE project support. [GKKL12] discusses the benefits and issues of cloud computing for various aspects of SE. The aspects are requirements, architecture, testing, quality, development methods, service-oriented architectures, or project management. In [Yau11] service-oriented SE respective SOA and its relations to cloud computing are discussed as the authors advocate that these two topics are closely related. First, they discuss both paradigms briefly and then describe how integrated developments utilizing both of them could look alike. Then, they present a set of issues emerging when combining these two paradigms. [BCJ10] proposes the application of a combination of cloud computing and model-driven engineering (MDE). Therefore, it introduces the notion of modeling as a service (MaaS) and provides a list of possible applications of the latter with cloud technology, like collaborative modeling tools, modeling mash-ups, or improving scalability of MDE.

[GuAl10] investigates the applicability and impact of cloud computing and web 2.0 technologies for SE. They state, that SE should integrate specific activities to deal with that new situation. Further, they provide an extension of the eXtreme Programming that integrates the cloud service provider into the process. [SiLu12] reviews other papers dealing with the combination of SE and cloud computing. It identifies a list of 10 ways, researchers approach this new paradigm, for example, proposing reengineering processes for the cloud, or cloud service composition. Finally, the authors also provide a discussion about some practical issues related to cloud computing.

In [MaSa13] different aspects of the combination of SE and cloud computing are discussed. Some target the usage of SE to better exploit cloud computing’s properties for applications (e.g., “Efficient Practices and Frameworks for Cloud-Based Application Development”), some aim for enhancing the quality of cloud applications (e.g., “Testing Perspectives for Cloud-Based Applications”), others target a better overall development cycle for cloud applications (e.g., “Business Requirements Engineering for Developing Cloud Computing Services”). However, none of them envisions or provides an approach for actively and holistically supporting SE projects.

As the combination of cloud computing is a very new trend in the scientific community, most articles concentrate on investigations how cloud computing could aid SE or how SE could improve development for the cloud. Cloud-based holistic project support is not prevalent.

6.3.5. Related Work Summary

There have been numerous approaches in SE that target the automated support of the SE process. These approaches can roughly be separated timely into three phases: In the first phase that started in the 1980s, the notion of CASE was created. Various CASE applications were created that supported various SE activities. CASE tools were created to support single activities like programming or compiling code. CASE workbenches integrated multiple of these tools into a single application to provide more holistic support. CASE environments, in turn, combined tools and workbenches to provide support for the whole SE process. However, in this phase of automated support a process focus was still missing and explicit process modeling, enacting or monitoring was not present.

This main shortcoming of CASE was overcome by PCSEEs, which added facilities to explicitly deal with the process itself. Most of these PCSEE included a proprietary process specification and enactment language. Respective languages were based on different foundations like Petri-Nets or Prolog. Most of these tools realized SE process support by means of specifying relations between humans and artifacts and are not capable of implementing and supporting a complex SE process model as used nowadays. Furthermore, the PCSEE and, first and foremost, their languages were difficult and cumbersome to use and process specifications were not well readable or understandable.
Recently, different types of SE support environments have been developed. However, these were not united under a common name like PCSEE. Names for these environments were rather picked more specifically same as their focus. An example is DOSDE the domain-oriented software development environment the focuses primarily on integrating domain knowledge and thus supporting the developers. Another example are CSDEs, the collaborative software development environments focusing on supporting the collaboration of project participants. All of these environments have in common that they provide advanced support but also that they are limited to a specific aspect of an SE project (see [MaRa12, dBBG10, ZCL05, LTXF04, ADOV02, CuGh1998, ACF97] for further details).

Other current SE approaches that aim to support software development as a whole mostly belong to one of the following categories: Industrial solutions that support the whole product lifecycle (ALM tools) or integrate different other tools and academic approaches that focus on specific parts of projects or a current supportive technology. There are, for example, efforts to integrate and utilize social media, mash-up environments or cloud technology into consistent SE approaches. However, these approaches are centered around one technology and its specific aspects, but fail to provide holistic SE project support. Other approaches target at the support of globally distributed teams and their issues. Nevertheless, these focus on specific aspects, driven by the issues such teams have. Finally, another category of approaches are recommendation systems for SE projects. As shown in this section, they have the same property as the other approaches just mentioned: They focus on one specific aspect and do not target holistic project assistance. The only ones capable of this are the discussed platforms by IBM and Microsoft. But they fail in providing real active support and guidance and do not go far beyond the integration of the participating tools.

None of the discussed approaches succeeds in providing the same as we intend with this work: Semantically and contextually enhancing process management technology and process enactment to be able to reach a higher level of automation and support. Further, they fail to implement and support SE process models, integrate operational support for the participating humans, integrate process enactment with other areas like quality management and the ability to actively intervene with process enactment to support, coordinate and optimize all activities that are part of the process.

**6.4. Summary**

SE projects are challenging and have been facing problems with their inherent complexity and their dynamic properties ever since. Despite numerous approaches to add more automated and holistic support to these projects these problems persist. One reason for this is the inability of automated solutions to really map and support SE process models that aim to support projects by structuring the myriad of different interrelated activities. On one hand, the models are often rather complex and abstract and are thus difficult to map in a uniform way that supports enactment. On the other, the models often do not really reach the humans and it remains challenging to connect them to the event happening in real everyday work environments.

This chapter has outlined the solution strategy of this approach to enable holistic SE project and process support and shown what basic requirements must be satisfied to achieve this. For different problem areas, different technologies are included and enhanced by additional new components that integrate the technologies and assure their seamless cooperation. As the whole equals more than the sum of its parts, new features are also enabled by this. By integrating process management technology, correctness of workflow enactment and human task distribution can be assured. The integration of rules processing technology facilitates the specification and enactment of recurring automatisms to disburden humans. Multi-agent technology adds autonomously acting and collaborating components that can cope with dynamic situations. Event acquisition and management technology enables the framework to be aware of the properties of various situations in SE projects. Finally, the creation of a central context management component based on semantic web technology enables the coordination and unification of the aforementioned technologies and thus supports the automated mapping and enactment of SE processes.
7. Contextual Extensions for Software Engineering Processes

In Chapter 1, we discussed fundamental problems of SE projects that relate to process implementation and contextual integration. SE processes are essentially knowledge processes [KeHa02]. In particular, they often deal with product development, which is a knowledge-intensive task [RaTi99]. Accordingly, SE processes strongly rely on the knowledge workers performing them. Furthermore, full process automation is not feasible and processes are mostly provided in a documentation-centric way [RBTK05]. Projects should utilize a defined process [GGK06] and applied process models contain a substantial amount of project knowledge [BWHW06, RiJa00, Mall09]. Such models are prescriptive [BDS+99, Ambl02] and are often implemented incompletely and often supported manually. Thus, the documented processes are not fully present in the minds of the executing humans [Wall07] and differ from the actual project execution. To optimally support operational SE process implementation, SE process models should provide automatic support. In turn, this implies that their implementation uses automated process tooling like a WfMS. Further, facilities are needed that integrate SE process models with the project context since SE project execution largely depends on contextual factors [Schw97].

This chapter describes the contextual extension of process management concepts (i.e., activities and workflows) being the basis of the CPM framework. This extension enriches the enacted workflows with supplementary information to comprehensively implement entire process models and integrating the process with various project areas, like quality management. Further, it fosters a better connection to the real-world process enacted by humans in an SE project. The following features are provided by this approach:

- Process specification in WfMS is extended to enable process enactment to be contextually integrated with the project;
- The implementation of entire SE process models is supported;
- The other components of the CPM framework get direct access to the enacted workflows to achieve automated integration of process enactment with other project areas and greater reactivity to changing properties of the situation; and
- The process specification is extended with features that support dynamicity in various ways.

The remainder of this chapter is organized as follows: Section 7.1 details three abstract requirements dealing with automated process support, dynamic process enactment, and contextual integration. Section 7.2 describes a basic concept for contextual process extension. Section 7.3 then discusses different aspects of workflow enactment regarding the CPM concept. Other aspects are detailed in the succeeding sections: extended activity modeling in Section 7.4, abstraction from internal workflow logic in Section 7.5, and the basics for automated process adaptations in Section 7.6. Section 7.7 then adds a formal and technical basis with formal definitions of the applied concepts, consistency checks, algorithms, and concrete actions applied for process enactment with the CPM concept. After discussing the concept, we go into detail about related work concerning that concept in Section 7.8. Finally, Section 7.9 summarizes the results and concludes the chapter.

---

1 This chapter is partially based on the publications [GOR10b], [GOR11c], [GOR11d], and [GOR12c].
7.1. Requirements

This section presents basic requirements regarding process specification, enactment, and dynamicity for SE. First, we present a set of problems of contemporary SE projects in Example 7-1.

Example 7-1 (Process implementation shortcomings):
Based on problems in its SE process, The Company aims at better support of this process by implementing processes with a WfMS. Such a tool shall not only be used to plan and document the process, but to actively govern its enactment. A team has been assembled to evaluate different options and to finally apply process implementation based on a WfMS. That team perceives several difficulties, which can be roughly summed up as follows:

1. The process specification lacks several features to implement the SE process. On one hand, it does not support a connection to its environment, meaning the process as it is really executed by different humans using various tools. On the other, it does not allow for a comprehensive integration of the concepts related to a SE process like the OpenUP [EcFo15]: Such processes often include different types of human activities or different ways of grouping them. Further, as operational support is desired, different levels of activities are involved. They often relate to each other and not all of them shall be governed by a workflow. Figure 7-1 illustrates this. Activity ‘Develop Feature X’ is an abstract assignment, which was planned and estimated from the business side. That activity implies sub-activities ‘Implement Solution’ and ‘Test Solution’, which are concretely planned and executed by the developer. In turn, these activities imply tasks on a more concrete level (i.e., activities ‘Check out source code’, ‘Create source code’ and ‘Check in source code’), which often have important connections to the environment, as they require, for example, certain tools.

![Diagram](image1)

Figure 7-1: Human activity granularities

2. Existing WfMS feature one kind of human activity. To model complex activity structures as illustrated in Figure 7-1, hierarchically structured workflows become necessary making both modeling and enactment complex and cumbersome. Further, the properties of the activities, such as additional supportive information, would be disregarded that way. Furthermore, some operational activities might occur in high frequencies and cannot be directly governed by a WfMS (e.g., when a human quickly switches between different source code artifacts he processes).

3. The process is rather rigid and is unable to mirror the dynamicity of the SE process. On one hand, it does not allow for dynamic changes of executed workflows to respond to a changing situation. On the other, connections between different workflows are limited, only allowing the modeler to connect a sub-workflow to an activity in a workflow. The Company, however, aims at further structuring their activities into work packages, as illustrated by Figure 7-2. In the planning phase of a project, for instance, the packages and their corresponding activities are planned. This means that the activity of planning a package depends on the completion of the planning of the contained activities. The same applies to the processing of a package. That way, there are multiple connections between the main workflow and its sub-workflow, and the completion of a particular activity depends not necessarily on the completion of a whole sub-workflow, but on the completion of one or multiple activities in one or multiple other workflows.

![Diagram](image2)
4. Indeed, the WfMS supports automated process enactment. Yet it has not provided other applications access to the process to support automated reactions on dynamic project situations. Thus, an automated tool will be unable to adjust workflow enactment to exceptional situations as they often occur in The Company.

5. If workflows are specified on paper, they only serve as guidance, but do not provide real process enactment support. However, if workflows are implemented within a WfMS on a fine-grained level to support software developers, this raises another challenge: As such workflows are rather dynamic, the modeled workflows must contain many decisions. To illustrate this, Figure 7-3 shows a simplified version of a SE workflow, which comprises seven decisions and related variables governing these decisions. Their values must be acquired from the human processing of the workflow. In our case, developers perceived this as cumbersome and error-prone, as it was not really clear which combination of variable values corresponds to which sets of future activities to process. On the operational level, this was one of the greatest obstacles for automated process implementation.

Regarding Example 7-1, the following requirements must be met. First of all, refinements of the previously elicited requirement R:ContInt (cf. Chapter 4) are made in order to add further detail on context integration within process implementation. This concerns WfMS’ lack of capabilities to integrate the workflows with contextual data and model different features of SE process models (cf. Example 7-1, point 1 and 2).

- **Requirement R:ContInt:ContProc (Contextual process specification):** To enable comprehensive process implementation support for SE, a framework should be able to integrate the process contextually into the project (cf. R:ContInt in Chapter 4). Therefore, contextual data should be available to that framework (cf. R:ContInt:GathCont and R:ContInt:ProcCont). This data must be also usable during process enactment, otherwise the latter might not conform with reality (cf. Example 7-1). Thus, the specification of the workflows should support the connection to and integration with contextual data.

- **Requirement: R:ContInt:ActGran (Activity granularities):** To support the SE process as it is actually executed and to be able to map the various interconnected types of human activities (cf. Figure 7-1) that appear in reality and in the SE process models, a framework must also support such activities with different granularities and properties.

The second area of requirements concerns dynamicity of process enactment. The relating requirement R:DynProc (cf. Chapter 4) is detailed in the following:

- **Requirement R:DynProc:VertCon (Vertical workflow connection):** In actual SE project execution, the traditional way for WfMS to connect activities with sub-workflows is
insufficient. Support for dependencies to single activities in other workflows shall also be supported (cf. Example 7-1, point 3).

- **Requirement R: DynProc: Adapt (Adaptable workflows):** To be able to react to the changing situations in the dynamic SE area, a framework should be able to incorporate changes to workflows. These changes should be possible during workflow enactment, as the need for these changes will not always be known prior to workflow initiation.

- **Requirement R: DynProc: AutoAdapt (Automated situational adaptations):** As SE project execution involves many different factors, not all information necessary to perform a workflow adaptation with maximum effectiveness might be available to a human. Therefore, a framework should allow for context-dependent automatic adaptations of running workflows.

The last area of problems as described in Example 7-1 is the automation of process implementation. Therefore, additional sub-requirements are added to R: AutoProc:

- **Requirement R: AutoProc: PrMult (Sub-process multiplicities):** The implementation of a process model requires that an arbitrary number of sub-workflows may be connected to the activities of a workflow. For example, the construction iteration of the OpenUP process [EcFo15] has one reference to sub-workflow ‘Develop Solution Increment’. This workflow is applied for the development of software. In a real project, however, there will be multiple references needed as multiple humans will be working on multiple activities.

- **Requirement R: AutoProc: AbstractOp (Abstract and operational areas):** Most process implementations are either abstract or concrete. On one hand, there exist project management tools that focus on the rather abstract parts of a process like phases or iterations. On the other, WFMS are applied mostly in rather concrete cases for operational activities. A framework providing comprehensive support for the SE process, however, must implement the process in a consistent way, including the connection of the abstract process areas (e.g., project, phase, and iteration) with the operational workflows of the software engineers.

- **Requirement R: AutoProc: ProcAcc (Automated access to process):** Automation is important to support and govern SE process enactment. Therefore, the specification of workflows comprised in the processes should grant other applications access to the workflow enactment. This is crucial for extended coordination, easy integration of contextual facts, and support for dynamicity (cf. R: DynProc: AutoAdapt and cf. Example 7-1, point 4).

- **Requirement R: AutoProc: AbstrIntLogic (Abstraction from internal workflow logic):** A framework for SE process support should encompass a way to abstract from the internal workflow logic and provide humans with easily understandable decisions for choosing the activities to process (cf. Example 7-1, point 5).

To provide an overview of the elicited requirements, Table 7-1 shows the three requirement areas relating to this chapter and the newly added sub-requirements.

<table>
<thead>
<tr>
<th>Req ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R:ContInt</td>
<td>Contextual Process Integration</td>
<td>The process specification shall support contextual integration into the project accommodating the properties of SE.</td>
</tr>
<tr>
<td>R:ContInt:ContProc</td>
<td>Contextual process specification</td>
<td>The process shall be contextually integrated.</td>
</tr>
<tr>
<td>R:ContInt:ActGran</td>
<td>Activity Granularities</td>
<td>Activities shall be supported in different granularities.</td>
</tr>
<tr>
<td>R:DynProc</td>
<td>Dynamic Process</td>
<td>The process implementation shall be dynamic.</td>
</tr>
<tr>
<td>R:DynProc:VertCon</td>
<td>Vertical workflow connection</td>
<td>Extended connections between workflows shall be supported.</td>
</tr>
<tr>
<td>R:DynProc:Adapt</td>
<td>Adaptable Workflows</td>
<td>Workflow instances shall be adaptable during runtime.</td>
</tr>
<tr>
<td>R:DynProc:AutoAdapt</td>
<td>Automated Situational Adaptations</td>
<td>Automated situational process adaptations shall be supported.</td>
</tr>
</tbody>
</table>
7 Contextual Extensions for Software Engineering Processes

<table>
<thead>
<tr>
<th>R:AutoProc</th>
<th>Automated SE Process Support</th>
<th>SE process implementation shall be automatically supported.</th>
</tr>
</thead>
<tbody>
<tr>
<td>R:AutoProc:PrMult</td>
<td>Sub process multiplicities</td>
<td>Various sub-workflows shall be able to be connected to a super workflow.</td>
</tr>
<tr>
<td>R:AutoProc:AbstractOp</td>
<td>Abstract and operational areas</td>
<td>Abstract process specification shall be connected with operational workflows.</td>
</tr>
<tr>
<td>R:AutoProc:ProcAcc</td>
<td>Automated access to process</td>
<td>Workflow specifications shall provide support for automated access.</td>
</tr>
<tr>
<td>R:AutoProc:AbstrIntLogic</td>
<td>Abstraction from internal workflow logic</td>
<td>Internal workflow logic shall be abstracted from humans.</td>
</tr>
</tbody>
</table>

### 7.2. Contextual Software Engineering Process Extensions

We detail the developed concepts and model of the CPM framework. Before designing this model, we checked existing models for their applicability to the SE domain and requirements. Our findings indicated that the model with the greatest applicability is the Software Process Engineering Metamodel (SPEM) [OMG08]. SPEM constitutes an abstract, generic and comprehensive model for describing processes in the SE domain. For various reasons, however, we do not choose SPEM:

- Due to its genericity, SPEM is rather complex and heavyweight, i.e., it contains a fair number of concepts and properties not applicable to our case.
- Our goal differs, since we want to develop a model that is rather concrete and focused on the enactment aspect, not just describing processes. There exist various approaches towards making SPEM executable [BCCG07, DRGC06, FMZ06]. However, these always involve a number of compromises and omit certain parts of SPEM.
- Another goal concerns the integration and processing of contextual events. This is not adequately realized in SPEM.
- Our model must not only include concepts and properties enabling automatic contextual processing, but also facilities for enabling interaction between context and process.
- SPEM lacks the integration of prevalent concepts from the process management domain, which is also important to facilitate the enactment aspect.

As discussed in Chapter 6, the CPM framework features a set of interacting components. These are applied to implement processes, enact them, and integrate them with their environment in SE projects by means of contextual data. To enable the context management component to be the central coordination unit between the process and the environment, as well as to integrate the process with the project environment, context information is received from the event processing component. To be able to directly use that information for process enactment, the context management component needs extensive access to the processed workflows. Therefore, the process management concepts (e.g., activities and workflows) are contextually annotated within the ontology. This means, for each concept applied to enable workflow enactment in the WfMS, there exists a related concept in the ontology, which is processed by the context management component. That way, both the context management and the process management component work closely together to provide contextually extended workflow enactment. This extension provides the following advantages:

- It allows for an automated access to workflow specifications and workflow enactment (cf. R:AutoProc:ProcAcc), and thus enables other components of the CPM framework to interfere with workflows to adapt them to changing situations.
- It enables a connection between acquired contextual data and workflow enactment (cf. R:ContInt:ContProc).
- It provides the required basis for automatically applying situational workflow adaptations (cf. Section 7.6).
Figure 7-4 illustrates the contextual mirroring of process management concepts in the ontology that is the basis of the features.

Figure 7-4 shows example workflows on the right side: On the top, workflow template is used to specify the workflow. When such a workflow is enacted, a workflow instance (lower part of figure) is created using the properties of the workflow template. Both concepts are annotated and mirrored in the context management component. This enables the latter to have direct access to workflows and activities. For each workflow template, there exists a corresponding mapping in the ontology, called work unit container template. The latter contains work unit templates that correspond to the activities of the workflow template. Workflow instances are mapped by the work unit containers that contain work units corresponding with activities in a workflow instance. Work unit container templates have properties for workflow specification and work unit containers for enactment. As opposed to most common WfMS, the activities have different properties for specification and enactment, realized by the concepts of the work unit templates and work units. Each concept in the context management component has a reference to the mapped concept in the process management component (respectively the WfMS). That way, the context management component can coordinate workflow enactments in alignment with contextual data.

The Example 7-2 shows contextual extensions of the process management concepts. We use the OpenUP process [EcFo15] as running example.

**Example 7-2 (Contextual extensions to OpenUP):**

To illustrate basic extensions to workflows, we apply the workflow ‘Develop Solution’. This workflow is used to ‘design, implement, test, and integrate the solution for a requirement within a given context’ [EcFo15]. Regarding SE projects, this deals with the development of new software. Figure 7-5 shows the workflow and its contextual extensions.

The workflow favors a test-driven development (TDD) [Beck03] approach and, therefore, before developing the solution, developer tests shall be written. After that, a cycle starts, which contains activities ‘Implement Solution’ and ‘Run Developer Tests’. That cycle is repeated until the solution is developed and the related test succeeds. If the code is well designed, it can be integrated; if not, there is another iteration of the whole procedure. After integrating the newly developed solution, there may be another iteration, if there is more work to be done. Otherwise, the workflow terminates. As shown in Figure 7-5, there is a work unit for each described activity.
Subsequent examples will utilize the workflow introduced in the preceding example to illustrate different parts of the developed concept. It will be further explained how the contextual extension is actually used to achieve the required features. Before that, the basic communication between the context and process management components is illustrated. The following two procedures (i.e., Procedure 7-1 and Procedure 7-2) illustrate the communication between the two components in two basic situations. The first procedure is executed when a new workflow instance is created from a workflow template. The procedure is illustrated by Figure 7-6.

**Procedure 7-1 (Workflow instantiation):**

1. Instantiation Event Distribution: The context management component receives a trigger event for instantiation, which may stem from various sources (e.g., the event management component or GUI).
2. Template Retrieval: The context management component retrieves the work unit container template from the project context.
3. Concept Creation: The context management component creates the work unit container as well as all relating concepts (like work units).
4. Instantiation Event Distribution: The context management component distributes an event to the process management component.
5. Workflow Instantiation: The process management component accesses the WfMS to instantiate and start the relating workflow.
6. Start Event Distribution: The process management component distributes an event to the context management component.
7. State Change: The context management component marks the work unit container as started.

**Procedure 7-2** represents the basic communication between the context and process management components, enabling workflow enactment with human task distribution. The procedure shall only illustrate the basic communication between the components, and thus ignores additional concepts regarding human activity management. Figure 7-7 illustrates the procedure.
Procedure 7-2 (Human activity distribution):
1. Activity Event Distribution: As the workflow instance runs, enactment reaches one or more activities that get activated. The in-process component (cf. Chapter 6) in such an activity generates an event for the context management component.
2. GUI Event Distribution: The context management component creates an event for the enactment GUI (cf. Chapter 6).
3. GUI Display: The enactment GUI displays the new activity to the human.
4. Activity Start: The human starts the activity. The enactment GUI generates an event for the context management component.
5. State Change ‘started’: The context management component changes the state of the respective work unit to ‘started’.
6. Activity Finish: The human finishes the activity. The enactment GUI generates an event for the context management component.
7. Termination Requirement Check: The context management component checks for additional requirements regarding the termination of the respective work unit (see below).
8. State Change ‘finished’: If no additional requirement is in place, the context management component changes the status of the work unit to ‘finished’ and distributes an event to the in-process component.
9. Activity Termination: The in-process component distributes a termination event back to the context management component and terminates. Thus the workflow enactment in the WfMS continues.

The management of activity termination by the context management component enables the utilization of various contextual factors in connection to activity enactment. As opposed to classical WfMS, the requirements for terminating a work unit can be defined flexibly, relying on different concepts defined in the context management component. This is illustrated by Figure 7-8.

For example, dependencies to other work unit containers or even work units within other work unit containers can be established, as illustrated for work units ‘A2’ and ‘A3’. Other influences, e.g., stemming from the SE process models to be implemented, may be incorporated as well. This is illustrated for activity ‘A1’. This includes related activities of various granularities, like in the OpenUP process, where an activity may have a set of subordinated activity steps whose sequencing is not specified by a workflow. Another example concerns artifacts related to the processed activities, as illustrated for activity ‘A4’. Here, the processing of a development checklist is required when performing a coding task.
7.3. Software Engineering Workflow Governance

The contextual extension and enactment of workflows not only yields the possibility of tightly integrating context parameters with process enactment (cf. \textit{R:ContInt:ContProc}); it also enables a more flexible way of modeling dependencies between workflows (cf. \textit{R:DynProc:VertCon}). As opposed to this, WfMS typically support one type of vertical connection between workflows. These connections cannot be managed dynamically as they are predefined by the connections between the workflow templates. Therefore, we move the management of these connections to the context management component. The CPM framework separates horizontal and vertical workflow governance. Horizontal governance refers to the dependencies and connections between workflow activities, while vertical governance deals with the dependencies between workflows. This is illustrated by workflow instances ‘A’, ‘B’ and ‘C’ in Figure 7.9.

In Figure 7.9, the context management component governs the dependencies between the three work unit containers that are connected to the workflows. In the given example, work unit ‘A2’ depends on work unit ‘B3’; i.e., the termination of ‘A2’ requires the one of ‘B3’. The termination of ‘B3’, in turn, requires the termination of work unit container ‘C’. The governance of the sequencing of the activities contained in one workflow stays with the process management component (respectively the WfMS). This is done because WfMSs have well elaborated correctness and soundness principles that guarantee correct enactment (this does not apply for all of them, but we assume a WfMS capable of this, cf. [Reic00]). Therefore, the work units contained in one work unit container do not have mutual connections.

Figure 7-9 illustrates the different parts of this concept: Section 7.3 discusses horizontal and vertical governance. Human-centric modeling of various types of activities is discussed in Section 7.4. Section 7.5 elaborates a modeling approach to abstract from the internal workflow governance logic. Section 7.6 contributes an approach to automatically adapt running workflows to context changes. Finally, Section 7.7 discusses the conceptual framework the other features are based on. This incorporates formal definitions, logical consistency checks, algorithms, and concrete actions to practically apply the concepts.
7.3.1. Horizontal Governance

Horizontal governance, i.e., the dependencies and connections between workflow activities, has two components: First, executability of the workflows on the process management side must be guaranteed. Second, the correct alignment of the contextual extensions with their states and properties to the process management concepts must be assured since the former connect the context parameters with the high-level workflow enactment. Both are illustrated by the OpenUP process in Example 7-3.

Example 7-3 (Executability of workflows):
Reconsider workflow ‘Develop Solution Increment’. Figure 7-10 shows the basic workflow extensions as well as a set of internal (e.g., event management) and external (e.g., the humans) framework components whose communication with the workflow is managed by the context management component.

The executability of the workflow and activities in the process management component may be problematic, since the high-level governance and communication with the context (and other framework components) is accomplished by the context management component, while the actual governance of the workflows is accomplished by the process management component. The activity termination procedure has already been explained. However, other parts of the workflow can be an issue (e.g., workflow patterns like XOR-splits): They need internal workflow variables for their decisions and it must be assured that their actual values are always available. As an example consider activity ‘Design the solution’ in Figure 7-10, which is surrounded by an XOR pattern. This activity is only executed if the solution to be developed is so complex that it requires an explicit design. This information can be derived from an external tool (e.g., a project management tool), in which the activities are planned, or directly from humans.
The issue of workflow variables and their connection to the context is addressed by a set of standard workflow variables used for the WfMS. That way, the context management component knows all variables (i.e. data elements) used to govern a workflow instance. These variables are initialized with standard values before starting the workflow instance. In turn, these standard values can be set during the modeling of the workflow and thus define a standard trace the workflow instance will produce if no variable is changed. The in-process component (cf. Chapter 6), which is executed for each activity in the workflow, processes these variables and updates them if required. Updating the variable values from the context can be accomplished at workflow initialization time or by the activities (and the respective in-process component) during enactment. Therefore, the in-process component updates the values in the workflow before it terminates. This concerns values associated with activity enactment as well as contextual values whose change had been detected.

The correct alignment of the contextual extensions with the activities and workflows must assured, since the former have no information on the horizontal structure of the workflows and their governance. Thus, based on different workflow patterns, activities might be executed more than once or be omitted completely. Hence, there must be a facility to provide information on the repeated or omitted enactment of the work units to, e.g., cancel or reactivate them. However, the context management component has no access to the WfMS internals and thus, workflow enactment constitutes a black box. When an activity is started, the context management component gets control over the enactment. This happens when an in-process component (cf. Chapter 6) is activated. However, information on the actual order of activities in the workflow instance is not available to the context management component. To enable the latter with this limited information about actual workflow enactment to still maintain correct enactment states for its concepts, an automated static analysis is conducted on the workflow templates. This analysis adds markings to the work units to enable the setting of their correct states. These markings include the marking of omittable and repeatable activities and the terminator activities. The latter are utilized to determine when certain activities will not come to further enactment. This is used for activities surrounded by XOR or LOOP patterns. For example, the first activity after a loop will be the terminator activity for all looped activities. That way its enactment marks a point in the enactment of the workflow instance when the framework knows that the looped activities will not be re-executed. In fact, the procedure is more complicated and will be further elaborated in Section 7.7.3. Example 7-4 serves for a concrete illustration of these markings applied in the OpenUP process.

Example 7-4 (Workflow markings):
Again, the ‘Develop Solution Increment’ workflow of the OpenUP process is chosen for illustrating the ‘repeatable’ and ‘omittable’ markings as depicted in Figure 7-11.
7.3.2. Vertical Governance

To accommodate the flexibility requirements regarding the hierarchical connections between workflows (cf. \textit{R:DynProc:VertCon}), we have extended the modeling of the workflows utilizing the context management component. We call this vertical governance. That way, process modeling is separated: the connections between activities contained in workflows and their governance are accomplished by the process management component and the WfMS (cf. Chapter 6), while the work units in the context management component, associated with the activities of a workflow, have no mutual connections. That way, only the WfMS is in charge of the governance of their sequencing. We opted for that because WfMSs provide the necessary concepts and checks to guarantee correct enactment [Rei00]. The hierarchical connections between workflows, however, are managed by the context management component to allow for more flexibility. In traditional WfMS, only one kind of hierarchical connection is supported: An activity in one workflow may contain an associated sub-workflow and, thus, the termination of activities depends on the workflow’s termination. As shown in Section 7.1, this is insufficient in some cases. Dependencies of activities to other activities in other workflows can be crucial as well (cf. Requirement \textit{R:DynProc:VertCon} and Figure 7-2). Thus, such dependencies are extended as illustrated in Figure 7-12.
Two options exist for such dependencies:

- Depends on work unit container: This connection associates a work unit container to a work unit. When this work unit is executed, the work unit container is started. The termination of the work unit then depends on the termination work unit container. This corresponds to the standard type of dependency also realized in WfMSs.

- Depends on work unit: This connection adds a dependency to a work unit. As illustrated in Figure 7-12B, the termination of work unit ‘A2’ depends on the termination of work unit ‘B3’. If not already running, the work unit container comprising work unit ‘B3’ is started by executing work unit ‘A2’.

Realizing these two types of connections in the context management component implies that the workflows on the process management side have no mutual connections. The dependencies are then added to the termination requirements of a work unit. That way, the work unit will only terminate if the work unit (container) on which it depends terminates. This requirement may be combined with human tasks added to a work unit. All entities or events belonging to the termination requirement of a work unit actively trigger the termination check. However, the work unit will only terminate if all requirements are satisfied. As an example, consider a work unit that depends on another work unit and has an associated human task. The termination of the work unit it depends on will trigger the check for termination. However, it will only terminate if the related human task has terminated as well.

Another important aspect to be covered is the data utilized in the workflows. The data in one workflow may influence the data in another dependent workflow. In contemporary WfMS, data can be passed from a sub-workflow to its super-workflow and vice versa. This implies the issue that all data elements in all workflows must be supplied with data values. In our approach, this topic is simplified: All content-related data is not kept in the workflows, but only processed within the context management component. In the workflows of the WfMS, the only data kept is for producing the exact trace of the workflows. More precisely, only data elements that govern workflow patterns such as XOR and LOOP are contained in the workflows. To keep data accommodation simple and robust, our approach requires that all these data elements are in place when starting a workflow. They are initialized with standard values representing the standard trace of the workflow. The data transfer from one workflow to another (if they depend on each other) is kept similarly simple: If one data element has the same name in both workflows, it will be passed from the super-workflow to the sub-workflow upon instantiation of the latter and vice versa when the latter terminates.

Modeling the connections in the context management component has another advantage: In traditional process management, the connections are rigidly predefined by workflow templates and cannot be changed for actual enactment. Our way of modeling is more flexible. Besides the possibility of modeling hierarchical workflow connections as templates being instantiated as modeled, it is also possible to dynamically instantiate and connect different workflow instances as required. This depends on two aspects: The dependencies are explicitly modeled in the context management component and the work unit containers can be created without starting them – as opposed to workflow instances in a
7 Contextual Extensions for Software Engineering Processes

WFMS. That way it becomes possible to prepare the whole process specifically for a concrete project as a workflow structure prior to starting it. Further, new workflow instances (e.g., for a new development iteration in iterative development) can be added to the process on the fly.

Due to the use of workflow patterns (i.e., XOR and LOOP), activities (and the relating work units) can be executed more than once (repeatable) or can be completely omitted (omittable). If a work unit depending on another one or on a container (i.e., work unit ‘A2’ in Figure 7-12) is repeatable, the concepts on which it depends will be instantiated once for each enactment of the work unit. However, the use of the connection ‘Depends on work unit’ has implications. If a work unit depends on another one, which is repeatable or omittable, this can be problematic. Required data might not be provided that way. Even if no data exchange happens, dropping a work unit (due to an XOR pattern) might lead to a deadlock of the dependent work unit container. To cope with this issue, the static analysis described in the algorithms for omittable and repeatable activities (cf. Appendix B) is utilized: If a work unit is repeatable and data must be transferred from it to another work unit, the data is transferred and overwritten in each enactment. If a work unit is omittable, two options can be configured. It can be prohibited or allowed to use ‘Depends on work unit’ on that work unit, even if no data transfer happens. The problem of data transfer is solved with the approach presented in Section 7.5 guaranteeing initial values for all data elements in a workflow. However, it must be determined when the depending work unit can terminate if the work unit it depends on is omitted. This is then done by the terminator activities that deactivate a work unit. As part of the deactivation procedure, an event is sent to the dependent work unit allowing the latter to terminate. If a work unit is repeatable, the data is passed upon each enactment if data transfer is needed. However, it must be determined when to resolve the dependency. That might happen when executing the respective work unit for the first time; however, a more suitable time point is the last enactment of the activity. Therefore, it must be determined when an additional enactment of the work unit cannot occur.

7.4. Extended Software Engineering Activity Modeling

As addressed in the requirements, the activity modeling in traditional WFMS might not always be suitable with respect to human tasks (cf. R:ContInt:ActGran). Especially, this applies in environments where human tasks are rather complex and have many dependencies as in is the SE domain. However, complex human tasks that cannot be captured to their full extent by traditional WFMS technology exist in other domains as well, e.g., the clinical domain [WSR09, LeRe07]. In SE, there are different granularities of interconnected human activities. Some of them are managed within tools (e.g., project management tools), others are managed by humans, and some are not managed at all. To exemplify this, Figure 7-13 shows an excerpt from the OpenUP process.

![Figure 7-13: OpenUP excerpt [EcFo15]](image-url)
There exist rather complex activities executed by humans within a development iteration, like the ‘Develop Solution Increment’ being used for the development of software components. Such activities may have an attached workflow as well as more fine-grained activities (e.g., ‘Implement Solution’ or ‘Implement Developer Tests’). The latter, in turn, may have even more fine-grained parts called activity steps not governed by a workflow like ‘Write the test setup’ (cf. Figure 7-13). These are difficult to model with traditional WfMSs. However, OpenUP does not even cover all human activity information: the information what is actually done via an activity like ‘Develop Solution Increment’ (e.g., creation of a new GUI component) must be managed elsewhere. Similarly, human activities on a rather concrete level performed with SE tools (e.g., source code checkout) are completely disregarded. The process model is more a recommendation rather than a real supportive and governing instrument for process enactment.

To better integrate human tasks into process enactment, to enhance contextual relations, and to better match the properties of the SE process models, our solution extends activity modeling by utilizing the context management component. We add three concepts, namely assignment, assignment activity, and atomic task (cf. Figure 7-14). They have different properties to capture different levels of activities with different kinds of dependencies. They are utilized as explicit activity-related extensions of the work units / work unit containers being solely in charge of the workflow governance. Another extension are the areas used to group activities into certain areas, as e.g., the disciplines in OpenUP. These areas can be attributed to either assignments or assignment activities.

The assignment is used to cover high-level assignments like the development of a new software feature. This constitutes a relatively complex task with various sub-activities. Therefore, the assignment is connected to a work unit container. Assignments are usually managed on a high level in projects, meaning that they are planned and scheduled at the beginning of these projects. This implies temporal properties and a relation to project management for these assignments. To foster integration with project management, assignments can be imported from project management tools. With such tools, it can be usually planned what shall be done and when, but the ‘how’ is left to the process or the human involved. Since the assignment is connected to a work unit container, it is connected to a workflow that can be automatically executed. Additionally, time can be automatically recorded to enable better tracking possibilities on the assignments. This can relieve developers from manually using the project management software to track assignment completion.

The assignment activity covers more fine-grained activities to be executed in order to complete the assignment. In turn, these are concrete activities, like implementation of new functionality or writing developer tests, which are relevant to the involved human, but have no direct relation to project management. Therefore, the assignment activities are connected to the work units processed to
complete a work unit container, which is associated with an assignment. While the work units are used as mappings of the activities in a workflow instance, the assignment activities are used to distribute the activities to the human. To connect workflow enactment with the management of human activities, the assignment activities are used as requirements for the termination of an associated work unit. For the assignment activities, their processing time can be automatically recorded since their start and end time points are known by the framework. This information can be used to automatically complete assignments when all required assignment activities of that assignment are completed. An activity, like the implementation of new functionality, can still be a relatively complex procedure and humans might benefit from additional assistance on the different steps that are part of the activity. As example, consider activity ‘Implement Solution’ from the OpenUP process. Such an activity comprises so-called activity steps [EcFo15] like ‘Write Source Code’, ‘Evaluate the Implementation’, or ‘Communicate Significant Decisions’. These steps are not governed by a workflow, as their enactment may be rather fuzzy in practice. To enable guidance for the steps as well, the assignment activity may include various activity steps. These can be used as additional guidance for the developer, or their enactment can be required to complete an assignment activity. See Example 7-5 for an illustration.

The atomic task constitutes the most concrete level of activities, like checking out source code, debugging, or coding. This concept might overlap with the activity step contentwise. While the goal of the activity steps is to provide additional guidance on what has to be done to complete an assignment activity, the special property of the atomic tasks is the relation to the environment of the CPM framework. For example, checking source code in and out relates to source control management systems, whereas coding or debugging relate to IDEs. This relation is utilized by the event management component, which provides a set of sensors for a multitude of different tools being able to detect various activities within such tools (e.g., source control systems or IDEs). These are mapped to the atomic tasks, enabling an automated detection of such tasks. This detection can be used to detect whether atomic tasks are executed that are part of another assignment activity rather than the one being currently processed. That way, discrepancies between the real world and the planned process can be revealed. It further allows for better traceability on what is done to achieve different assignment activities. Task times can be recorded as well as other properties like the number of task switches within an activity.

Example 7-5 (Extended activity modeling in OpenUP):
Figure 7-15 illustrates the extended activity modeling possibilities of this approach utilizing the ‘Develop Solution Increment’ workflow of the OpenUP process.

The work unit container of the ‘Develop Solution Increment’ workflow has a related assignment that represents the concrete activity to be performed when using this workflow (called ‘Develop feature X’). The contained assignment activities are attributed to area ‘Development’ (called ‘discipline’ in the OpenUP). Each work unit has a related assignment activity to be displayed in the GUI. Each assignment activity, in turn, may comprise multiple activity steps (e.g., ‘Design the Solution’ has steps ‘Understand requirement details’ or ‘Identify design elements’). In addition to the steps, each assignment activity comprises one or more atomic tasks. These represent fine-grained activities that relate to certain SE tools, as for example, ‘Check out’ or ‘JUnit execution’.

7 Contextual Extensions for Software Engineering Processes
7.5. Abstraction from Internal Workflow Logic

This section gives insights into the human-related abstraction from internal workflow logic (cf. R:AutoProc:AbstrIntLogic). As the considered workflows are human-centric, their actual execution trace must be created based on the decisions of the human processing this workflow. Otherwise, the latter will not represent the real course of activities and thus become irrelevant. Such governing involves an arbitrary number of control variables from the workflow for deciding how XOR and LOOP patterns are to be executed. This can be cumbersome for the human since it might not be transparent which of the variables shall be set to which value to execute a certain sequence of activities. Example 7-6 elaborates on this.

Example 7-6 (Human-centric workflow with decisions): Shortcomings of contemporary WfMS may cause problems when trying to model human-centric workflows. As SE processes are dynamic, this domain epitomizes such workflows. We create an operational workflow based on practical experiences. The example concerns a workflow that governs activities utilized for creating a new piece of software. Figure 7-16 shows a simplified version of a workflow we created during an interview with software developers from a software company.

Figure 7-16: Example human-centric workflow (unstructured)
The workflow starts with the analysis of the work item (that corresponds to the CPM assignment) to determine what has to be done to achieve the work item goal (e.g., to create the desired functionality) followed by the design of the solution. Thereafter, the code of the system is checked out and a local build is created. Additionally included activities concern the implementation as well as testing and a review of the solution. The workflow concludes with the promotion of the developed solution, meaning the integration of the locally created source code into the mainline development branch within the shared version control system. As in most situations the described activities cannot be simply executed in a basic linear non-repeating sequence, the workflow contains loops.

Note that the workflow depicted in Figure 7-16 is not well structured according to [MRv10], since loops overlap in some cases, contradicting proper nesting of workflow elements. Unstructured workflows are poorly readable and error prone. Thus, we restructured the workflow to enable proper nesting of the elements as shown in Figure 7-17. This is done by duplicating the build activity to resolve the overlapping loops. This configuration was chosen based on domain knowledge, since a real difference between the two introduced building activities exists: The first one, ‘Initial Local Build’, is conducted once before any implementation changes are applied to verify that the checked out versions of the code files build on the local machine (a.k.a. sandbox or programmer’s directory). The second activity, ‘Build Locally’, is conducted to verify that the changed code is buildable.

The workflow shows the different decisions to be connected to variables governing the sequencing of the activities. Each decision might be connected to a question towards the human to acquire necessary and missing information. Note that this realization of the workflow shows flaws: while processing the workflow, the human must always provide values for all decisions except D2 (this one can be set internally as the initial build activity should be only executed once). Consider activity ‘Test Solution’, which has four possible successors: ‘Review Solution’, ‘Promote Solution’, ‘Design Solution’, and ‘Implement Solution’ The human must be involved in all decisions during workflow enactment. This kind of workflow governance can be confusing and cumbersome for the human, provoking mistakes in the input coming from him, which then result in workflow traces being inconsistent with reality. This type of workflow interaction will most likely be perceived as burdensome by humans, which may affect their overall impression and resulting acceptance or rejection of such an automated guidance framework. To resolve this, a more human-centric way of communication is desirable, letting the humans focus on what they are doing rather than being distracted by the supplementary information requirements of a workflow that should be assisting them.

In order to be able to gather information needed for the enactment of a human’s workflow instance without burdening him with many abstract decisions, we have developed concepts for abstracting the workflow internals. Figure 7-18 illustrates these concepts by showing a small example workflow template as well as an instance including the mapping concepts in the context management component. The approach taken to enable abstraction from internal workflow logic and variables is grounded on these concepts. The work unit container template is extended by workflow variable templates that map the workflow variables (i.e. data elements as defined in Chapter 5) defined in the workflow template. For these workflow variable templates, the workflow variable values provide possible pre-defined values. The latter can then be used to set the variables for a workflow instance during enactment. These are mapped by the workflow variables (corresponding to the data element values in the workflow instances as defined in Chapter 5). Each work unit container template has a set
of workflow variable templates that provide the initial values for the variables of a new workflow instance. To grant the human flexible and intuitive control of the workflow variables, the workflow user information and user decision alternative are applied: The former is connected to a work unit template and used to inform the human executing that activity about a decision the human has to make. The latter represents one alternative of such a decision, and is, in turn, connected to workflow variable values used when that alternative is chosen by the human.

Consider the example from Figure 7-18: WorkflowInstance1 contains four activities of which two (Act2 and Act3) are mutually exclusive. For example, these might be two different code review activities, e.g., a peer review and a code inspection, both having different properties regarding required effort, duration, and error detection rate. As the human must decide which one is appropriate in the current situation, a value for the variable used in the context of the XOR-gate in the workflow must be acquired from the human. In the example from Figure 7-18, UserInfo1 is connected to the work unit template WUT1. The latter, in turn, maps the first activity in the workflow and is used to gather information on the upcoming decision while the human executes the current activity (e.g., asking “How much time is available for reviewing your code?”). The two options for that decision are modeled by DecAltern1 and DecAltern2 that provide the values for the XOR-gate using VarValue2 and VarValue3 (e.g., sufficient time left / high schedule pressure).

To ensure executability of a workflow, some basic conditions must be met. For all variables in a workflow, mappings must exist, their association to workflow variable templates must be clear, and initial values must be provided so that all workflows can be executed without human communication. The latter should be well defined, meaning that for each decision, one or more alternatives exist from which one is set as the default, and all have variable values to be set if a choice exists, since a human decision with no value and thus no impact would be irrelevant. These conditions are formalized as part of the conceptual framework of our approach (cf. Section 7.7). When a workflow instance is created from a workflow template, all variables receive their initial values from the workflow variable values defined for the work unit container. These values shall be defined in a way to minimize the necessary human interaction required for appropriately enacting the instance. That human interaction is illustrated in Figure 7-19. As aforementioned, the context management component adds several types of information to the workflows of the process management component.

In this case, information about the humans’ activities is relevant. It comprises an assignment that represents information about the work item for which the workflow was initiated, e.g., ‘Develop (new) feature X’. The different activities to reach that goal are represented by assignment activities, (e.g., ‘Implement Solution’). To enable better operational assistance, the concept of atomic task was added, representing activities like checking in source code or running unit tests. The different steps taken by
the framework to extend workflow enactment with more human-related semantics are described in Procedure 7-3 and are illustrated in Figure 7-19.

Figure 7-19: Workflow enactment with human decision extensions

Procedure 7-3 (Workflow enactment with human decisions):
1. Workflow Instance Start: An event (by a human or the framework) causes a workflow instance to start.
2. Activity Enabling: Workflow enactment reaches one or more activities that become enabled. This information is distributed to the context management component.
3. Assignment Activity Distribution: The context management component, in turn, distributes the relating assignment activity (cf. Figure 7-14) and potential additional information to the human.
4. Activity Start: The human starts the processing of the assignment activity.
5. Decision Information Distribution: The context management component retrieves the decision information and its alternatives and distributes it to the human.
6. Alternative Selection: The human selects one decision alternative (if different from the pre-selected).
7. Activity Finish: The human finishes the processing of the assignment activity.
8. Decision Information Distribution: The context management component informs the process management component that the active activity may complete now. This information incorporates values for the workflow instances’ variables.
9. Decision Variable Provision: The process management component sets the values of the variables and then lets the activity complete and the workflow instance continue.

This concept offers flexibility for transferring workflow governance control to humans by mapping internal workflow variables to user decisions. There can be multiple ways of defining and connecting different concepts to match a project’s needs.
A simple 1:1 mapping of user decision alternatives to variables can be applied. With such a mapping, a human could directly set the value for each workflow variable. In this case, the workflow user information will be used to store an appropriate question or statement to inform the human about the decision he is about to control now. The user decision alternatives, in turn, will store textual information on decision alternatives such as ‘Is additional implementation effort still required? – Yes/No’, which is less error-prone than setting a variable like ‘AdditionalImplEffort – true/false’;

A more complex n:m mapping where each user decision alternative sets multiple variables to provide the human with support for a more abstract decision can be established as well. Each of the alternatives could even control different sets of variables, allowing for completely different traces to be produced. This kind of mapping could make complicated workflows easier to handle for humans. In particular, they have one decision with a limited set of alternatives per activity, instead of having to set multiple variables all the time;

With the abstraction from the workflow variables and the explicit modeling of user decisions, there exists the possibility of restricting certain options of a decision that would be available if there was direct access to the workflow;

Since the modeling of user decisions is human-centric and abstracts from workflow internals, it fosters hiding technical complexity. Well-structured workflows are often bigger than unstructured ones describing the same situation (as shown in Example 7-6 with an additional activity). They might be more comprehensible for the process modeler, but could be more complicated for the executing human. This can be compensated via the additional abstraction layer introduced.

Example 7-7 shows the proposed approach applied to a concrete example (i.e., Example 7-6).

Example 7-7 (Human-centric workflow with workflow logic abstraction):
Five user decisions were required in Example 7-6 to properly govern the workflow (D2 can be set internally and D3 and D4 could be combined), that had to be displayed to the human during the entire workflow enactment. To simplify this, the following mapping of the internal workflow decisions was created (cf. Figure 7-20): for each possible successor of an activity, a decision alternative was created. That way the human can directly choose which activity to process next while executing an activity. Thus, no additional information on the decision is necessary (and the information of the workflow user information can remain empty). Each of the decision alternatives then sets the set of workflow variables required to activate the chosen activity. Additionally, a set of initial workflow variables is provided to create the most likely trace to minimize the required human interactions.

Figure 7-20: Example human-centric workflow with workflow logic abstraction
7.6. Automated Software Engineering Process Adaptation

This section introduces facilities for automated process adaptations. These become necessary to support dynamic SE processes as explained in Section 7.1 (cf. R:DynProc:Adapt, R:DynProc:AutoAdapt). “Automated” means that the CPM framework is able to adapt running workflow instances by taking information on the current situation into account that was gathered with the event management component. Adaptations to running workflow instances have already been described and technically realized for a long time [Reic00]. Therefore, for the direct structural adaptations of the workflow instances, we rely on the capabilities of adaptive WfMSs supporting this. However, adaptation capabilities feasible with such technology typically only consider manual adaptations applied by humans that understand the semantic aspects. Such adaptations have been utilized, for example, in the clinical domain or the automotive industry [LPR12, LeRe07, MHHR06, MRH08]. Due to the dynamic process and various sources of information in SE, all information required to apply an effective process adaptation may not always be in place. Example 7-8 illustrates such a scenario.

Example 7-8 (Quality management scenario):

The company has decided to apply explicit process management to SE. The process model applied, however, is relatively abstract and only manages coarse-grained activities like ‘Realization of new GUI component XY’. Such activities are planned at the beginning of a project and are then distributed to an executing human. The actual realization is left to that human. Thus, the process does not really touch the developers, and their activities are difficult to plan and trace.

The quality of the source code is not monitored continuously and static code analysis tools are only used sparsely by the developers. Thus, there is no awareness about the actual quality state of the source code. During the projects, deterioration of the quality goes undetected and software quality measures are not applied. In many projects, when it comes to the final phase, severe problems with the code show up. To still be able to deliver the software in time or with only short delays, many quality measures and bug fixes are then applied quickly by the developers under high schedule pressure. The efficiency and effectiveness of such measures is mostly non-optimal, even causing delays to other planned activities. Also, many emerging problems that would have been easy to fix remained undetected until they result in severe errors.

Example 7-8 describes a situation occurring in SE projects that requires changes to the running process, since activities for applying software quality measures must be added dynamically. Such situations comprise a variety of information necessary for effectively applying software quality measures. Examples of related questions are:

- Which code metric value must be considered as problem for the current project? Which ones would be a waste of time and should therefore be disregarded?
- Which quality measures are most important in respect to the quality goals of the project?
- Which developer should apply which quality measure?
- Which developer has time for a quality measure without delaying the overall process?
- Where and when exactly should a developer apply a quality measure?
- How effective was an applied measure? Should it be applied again in the future?

This is just a selection of issues crucial for effective quality management (cf. Chapter 9 for details). Here, we present it as an example of necessary changes to a running process. The CPM framework enables automatic support for dynamic adaptations. In this section, we deal with the basis for automatically changing running workflow instances that are part of the SE process. Before detailing the adaptation approach, we refer to Example 7-9 showing the reader in what context an automated adaptation may be applied.

Example 7-9 (Quality measure distribution):

Recall Example 7-8, which deals with software quality measures dynamically distributed during the course of an SE project. To enable such distribution in an automated fashion, the CPM framework must be aware of problems in the source code and about the humans’ activities in order to insert
quality activities without delaying other activities. To enable effective quality measure application, contextual factors (e.g., the skill level of the applying human) can be taken into account as well as the quality goals of the project. The steps of the related procedure are shown as follows:

1. Problem analysis: The event management component features sensors that are integrated in static code analysis tools. In turn, the sensors periodically analyze the source code by automatically measuring it. Thus, the CPM framework automatically receives information about the status of the source code, for instance, concerning code complexity (e.g., using the ‘Cyclomatic Complexity’ [McCa76 metric]). For each metric, a threshold is defined that, if violated, indicates a problem. For all problems, predefined quality measures (e.g., specific refactorings) are assigned to the related metrics.

2. Measure prioritizing: The assigned measures have no order or priority. Since it can be assumed that in most projects there is not enough time to react on all problems, measures should be prioritized. This is aligned with the quality goals of the project by the quality management component.

3. Quality opportunity detection: To effectively distribute quality measures to humans, the CPM framework can automatically detect when it makes sense to inject a quality activity into the human’s workflow. This is accomplished by the context management component. Such an opportunity emerges, for example, when a human finishes an activity earlier than planned.

4. Measure tailoring: When an opportunity for a quality optimization has been detected and strategically prioritized quality measures are in place, a quality measure can be distributed to the respective human. To maximize the effectiveness of the applied quality measure, the context management component conducts a measure tailoring that consists of two parts: First, a measure is selected considering various properties of the current situation (e.g., the skill level of the human or the amount of time available for the measure without delaying subsequent activities). Second, a matching insertion point in one of the workflows of the human must be determined, as certain quality measures are, for example, only applicable at the end of a project.

5. Measure application: When all parameters for inserting the quality measure are determined, the context management component transfers control to the process management component. The latter then automatically integrates the quality measure into the human’s workflow.

6. Quality trend analysis: Utilizing the source code quality reports from the event management component, the context management component automatically conducts a quality trend analysis on the source code.

7. Measure assessment: To be able to optimize the quality measure distribution procedure within the organization, the context management component automatically assesses the effectiveness of the applied quality measures. To achieve this, the application of the measures is related to the quality trend analysis. The assessment computes a value for the measure effectiveness that is used in future measure distribution procedures to more heavily weight effective measures.

To enable such automated contextual adaptations to the SE process certain basic facilities and concepts must be in place. In this section, the foundations for automated process adaptations based on contextual factors are introduced. If new activities shall be automatically added to a running workflow instance, various factors have to be considered to enable a syntactically correct and semantically matching insertion that also matches the properties of the workflow, human, and situation. First, the data supply of the newly inserted activity must be assured. Note that this comprises data on the process as well as the context management side. To not endanger the workflow executability, all variables utilized for decisions in a workflow must have an initial value that may be updated during enactment. On the context management side, different factors must be considered: The data provided to the human by assignments or assignment activities (i.e., information on what has to be done) must be present as well as the data required for the termination procedure of the inserted activity. Furthermore, more complex human activities might require that more than one (workflow) activity is inserted into the running workflow. Therefore, the insertion is always done with one single activity that is connected to another workflow that contains the complex activity to be processed by the human (cf. Figure 7-21).
The figure shows a running workflow instance ‘A’ with activities ‘A1’ – ‘A4’ as well as their contextual extensions work unit container ‘A’ with the corresponding sub-work units ‘A1’ – ‘A4’. The complex activity to be inserted is represented by the workflow (and respective work unit container) ‘B’ with activities (and respective work units) ‘B1’ – ‘B4’. The activity (and respective work unit) ‘B’ is inserted into the running workflow instance (and respective work unit container) ‘A’. Since the process management component is in charge of horizontal governance (cf. Section 7.3.1), it is also responsible for inserting the activity at the correct point in the workflow instance. Being in charge of vertical governance, the context management component manages the connection between work unit ‘B’ and work unit container ‘B’. This component further provides assignments, assignment activities, and atomic tasks for the complex activity to be inserted. These three can be predefined when creating the workflow templates, as the activities to be inserted have templates they are created from. Furthermore, the context management component enables the selection of an insertion point matching the context. This is supported through extension points and extensions. The former is an extension for work units that may be defined as part of their templates. When a work unit is annotated by an extension point, the CPM framework can consider the insertion of a new activity as a direct successor of that work unit. The explicit definition of the extension points becomes necessary as the framework may automatically check issues like data availability or syntactical correctness (but not the semantic suitability of an inserted activity without additional information).

For the semantically matching integration of new activities (e.g., quality measures) into running workflow instances, the properties of the situation at hand cover the properties of the workflow instance as well. Some quality measures might, for example, only be applicable at the end of an iteration, phase, or project. To be able to provide the framework with such information, the extension point features properties (e.g., the abstraction level) the process engineer may define upon process creation. Such properties can then be matched to the ones of an extension being used to pre-define the properties of an assignment. That way the framework can autonomously select a suitable insertion point for a new activity during run-time. The data needed by the newly inserted activity can be predefined by template concepts. This concerns the termination procedure as well: The newly inserted work unit ‘B’ terminates with the termination of work unit container ‘B’ and the contained activities terminate as configured by their templates.

There are other relevant factors with respect to newly inserted activities: If the activity becomes inserted within a workflow pattern, it might not be executed or be executed more than once. Both cases are problematic on the semantic level since certain activities (e.g., software quality measures prescribed by the framework) are intended to be executed only once. Therefore, the workflow must be analyzed during activity insertion to cope with such a scenario. We provide an algorithm treating both
cases: inserted activities are marked to be executed only once, whereas other activities marking a point in the workflow, where enactment of the inserted activities shall no longer be possible, are marked similarly to the procedure in the algorithms for omittable and repeatable activities (cf. Appendix B). Being marked that way, these activities generate an event for the CPM framework, informing it about the fact that the newly inserted activity will not come to execution. This can be of great importance to other components, e.g., regarding the insertion of software quality measures: If an inserted measure does not come to execution, another one might be proposed to that human. Furthermore, an unexecuted measure should not be incorporated in the automatic measure assessment procedure.

Example 7-10 demonstrates the insertion and marking of a new activity utilizing the OpenUP process.

Example 7-10 (Marking of newly inserted activities):
We reuse the ‘Develop Solution Increment’ workflow, which was used in previous examples, as shown in Figure 7-22.

A ‘Design Review’ activity is inserted right after the ‘Design the Solution’ activity. The newly inserted activity is placed within a LOOP as well as an XOR pattern. Therefore, it is marked for being executed only once. Consequently, the situation might occur that the design is made, the design is reviewed, the solution is created and a redesign is needed. When activity ‘Design the Solution’ is executed the second time, the review is not re-conducted, since it is a planned and time-consuming quality activity that shall only be executed if explicitly planned.

7.7. Conceptual Framework

First, this section provides a formal basis of the CPM framework with definitions of all concepts involved. In Section 7.7.2, consistency checks are applied to avoid erroneous definitions. Section 7.7.3 then discusses the algorithms utilized to enable the contextually extended process enactment with the context management component. Finally, Section 7.7.4 elaborates on concrete actions applied on the concepts to enable the practical use of a CPM framework for process enactment.

7.7.1. Basic Concepts

This section introduces formal definitions of the concepts introduced. Thus, the complete basic conceptual framework of the CPM framework is discussed. Later on, this framework will be extended with additional concepts (e.g., enabling knowledge or quality management). The most important ones and their use have already been discussed in the course of this chapter. However, they have not been formally defined and not all concepts needed for automatic process implementation have been introduced. Therefore, this section discusses the conceptual framework behind the CPM framework and illustrates it along an example dealing with process implementation.
Example 7-11 (Practical issues related to process implementation):
Having realized that manual process implementation bears many issues, The Company wants to implement a process model and automatically execute it in a WfMS. They choose the lightweight OpenUP process for this. Relying on standard WfMS technology, they experience many problems with the shortcomings of existing technology: WfMS technology strongly focuses on enacting workflows and many aspects of a project and its process model cannot be modeled therein. This concerns, for example, the different types of human activities, their special properties, and their connections. That way, the OpenUP would have to be simplified to model it in a WfMS.

Moreover, the great number of artifacts and their relations to other entities cannot be captured completely with data elements in a workflow. One type of such a relation concerns the different areas of a project and the different types of artifacts. Regarding requirements management, for example, different documents related with each other are used, while the implementation area comprises a myriad of different technical artifacts (e.g., source code files). Such files are further organized, for example, in source code packages. Such elements cannot be readily modeled in the WfMS, and thus many aspects are beyond the governing technology. Other aspects to be similarly disregarded are the project itself, with its properties, milestones, or tools used for enactment. The WfMS only executes various relating workflows with flat human tasks that have no relations to the real-world operational process. An example of this can be a coding task: It is not feasible to integrate an activity being directly connected with an IDE, as the exact timing and enactment of such an activity must be done by a human using such a tool.

Another aspect separating process enactment in a WfMS from real-world enactment of the process in the project are the many events happening during enactment and the potential problems they impose requiring changes to the executed process.

We distinguish between template concepts and individual concepts. We use the former term to indicate that these concepts are used as templates for the creation of other concepts. The latter term stems from ontologies and the semantic web where concrete objects are modeled via individuals. In CPM, templates comprise all concepts used for defining a process, whereas individuals consist of all concepts applied to one particular enactment of the templates.

**Template Concepts**

Figure 7-23 gives an overview of important template concepts. For a full list and discussion of the involved concepts, we refer to Appendix B. For an excerpt of the technical implementation of these concepts, we refer to Appendix A. In the following, we show two exemplary definitions of the most basic concepts (cf. Definition 7.1 and Definition 7.2).

As discussed, the basic idea is to extend concepts of a WfMS with additional concepts and relations provided by the context management component and tightly connect them with the concepts in the WfMS (i.e., activities and workflows). Basic concepts in this context include the work unit container template and the work unit template. The former is the direct mapping of a workflow template in the WfMS, whereas the latter maps the contained activities. These are defined first, starting with the work unit container template. Two properties shared by all concepts are type and name. The former denotes the type of concept, like work unit container, whereas the latter is a unique identifier for each concept. As both are common for all concepts, they are omitted in the definitions.
Definition 7.1 (Work Unit Container Template)
A work unit container template used to map and extend a workflow template. It is represented as a tuple
\(\text{workUnitContTempl} = (\text{type}, \text{name}, \text{wfTempl}, \text{workUnitTemplSet}, \text{assignmentTempl}, \text{mandInputSet}, \text{optInputSet}, \text{outputSet}, \text{primRoleTempl}, \text{addRoleTemplSet}, \text{reqContTemplSet}, \text{varTemplSet}, \text{varValSet}, \text{noWorkflow}, \text{dependencyTemplSet}, \text{projectTempl})\) where

- \(\text{wfTempl}\) is the workflow template annotated by workUnitContTempl.
- \(\text{workUnitTemplSet}\) is a finite set of work unit templates being used to map activity templates within the WfMS.
- \(\text{assignmentTempl}\) is an assignment template or undefined.
- \(\text{mandInputSet}\) is a finite set of project component templates used to define the mandatory input for workUnitContTempl.
- \(\text{optInputSet}\) is a finite set of project component templates used to define the optional input for workUnitContTempl.
- \(\text{outputSet}\) is a finite set of project component templates used to define the output for workUnitContTempl.
- \(\text{primRoleTempl}\) is a predefined primary human role template for workUnitContTempl.
- \(\text{addRoleTemplSet}\) is a finite set of additional role templates defined for workUnitContTempl.
- \(\text{reqContTemplSet}\) is a finite set of work unit container templates that workUnitContTempl requires.
- \(\text{varTemplSet}\) is a finite set of flow variables used to control workflow enactment in the WfMS.
- \(\text{varValSet}\) is a finite set of values for the flow variables used in varTemplSet to be used as initial values for the variables upon instantiation of a work unit container.
- \(\text{noWorkflow} \in \text{BOOLEAN}\) indicates a special type of container that contains activities whose sequencing is not specified and thus not governed by a workflow.
- \(\text{dependencyTemplSet}\) is a finite set of work unit container template dependencies.
- \(\text{projectTempl}\) is the project template workUnitContTempl is associated to.

WorkUnitContTempls describes the set of all definable work unit container templates.

We explain the properties of a work unit container template. The first property to be in place is a reference to the workflow template that shall be extended (\(\text{wfTempl}\)). The second basic property is a set of references to the work unit templates it contains (\(\text{workUnitTemplSet}\)). For human-centric activity
management two things must be in place: a reference to the assignment template capturing the human’s goal \( (assignment\text{Templ}) \) and references to the concepts enabling human-centric abstraction of workflow logic \( (var\text{Templ}\text{Set}, \text{varValSet}) \). That way, the variables in place for the container can be defined, including an initial value for them. For tightly integrating artifacts and workflows, the container template has three sets to define templates for input and output artifacts \( (\text{mandInputSet, optInputSet, outputSet}) \). It further defines the roles involved in processing the workflows by defining references to role templates for primary and additional roles \( (prim\text{RoleTempl}, add\text{RoleTempl}\text{Set}) \).

The SE process is rather dynamic, and the process is often composed from different workflows rather than applying one static process model. Therefore, a process may contain various sets of workflows depending on each other. To model such a relation between workflows, the container template has a relation to other container templates it requires \( (req\text{ContTempl}\text{Set}) \). Another property of real world SE workflows is the potential absence of structure. For some of them, it cannot be planned a priori in which sequence the activities shall be executed. In order to take this into account, the container template features property noWorkflow that skips the connection to a workflow template in a WfMS and only models a set of work unit templates without structure. For managing the inter-workflow dependencies, the container template refers to a work unit container template dependency \( (dependency\text{Templ}) \). The latter realizes the connection to a work unit template in another container template depending on the current one. Finally, to be able to model the integration of the process into a project, the container template has a reference to a project template \( (project\text{Templ}) \). That way, when a new project is started, a project template can be used referring to all concepts for this project’s process.

Definition 7.2 goes into detail about the work unit templates of a container template.

**Definition 7.2 (Work Unit Template)**

A work unit template is a tuple \( \text{workUnitTempl} = (type, \text{name, actTempl, assignActTempl, workUnitContTempl, mandInputSet, optInputSet, outputSet, primRoleTempl, addRoleTempl\text{Set, workflowUserInfo, extensionPointTempl\text{Set, repeatable, omittable, milestoneTempl\text{Set, dependsOnTemplSet, dependencyTempl\text{Set}}})} \) where

- \( \text{actTempl} \) is the activity template within the integrated WfMS annotated by \( \text{workUnitTempl} \).
- \( \text{assignActTempl} \) is the activity a human has to process to complete \( \text{workUnitTempl} \) or undefined.
- \( \text{workUnitContTempl} \) is the work unit container template \( \text{workUnitTempl} \) belongs to, i.e., the mapping of the workflow template in the WfMS that contains the activity which is annotated by \( \text{workUnitTempl} \).
- \( \text{mandInputSet} \) is a finite set of project component templates used to define the mandatory input \( \text{mandInputSet for workUnitTempl} \).
- \( \text{optInputSet} \) is a finite set of project component templates used to define the optional input for \( \text{workUnitTempl} \).
- \( \text{outputSet} \) is a finite set of project component templates used to define the output for \( \text{workUnitTempl} \).
- \( \text{primRoleTempl} \) is a predefined primary human role template for \( \text{workUnitTempl} \).
- \( \text{addRoleTemplSet} \) is a finite set of additional role templates defined for \( \text{workUnitTempl} \).
- \( \text{workflowUserInfo} \) is used to provide the human with necessary information to provide values for the variables guiding the workflow instance in the WfMS or undefined.
- \( \text{extensionPointTemplSet} \) is a finite set of extension point templates. It marks a point in the workflow template where workflow instances based on that can be automatically extended.
- \( \text{repeatable} \in \text{BOOLEAN} \) indicates if the activity that is mapped by \( \text{workUnitTempl} \) could be executed more than once because of the structure of the workflow.
- \( \text{omittable} \in \text{BOOLEAN} \) indicates if the activity that is mapped by \( \text{workUnitTempl} \) could be completely omitted because of the structure of the workflow.
- \( \text{milestoneTemplSet} \) is a finite set of milestone templates.
- \( \text{dependsOnTemplSet} \) is a finite set of work unit (container) template dependencies on which \( \text{workUnitTempl} \) depends.
- \( \text{dependencyTemplSet} \) is a finite set of work unit template dependencies.
WorkUnitTempls describes the set of all definable work unit templates.

The work unit template has a set of properties similar to the container template. It allows explicitly defining its corresponding activity in a WfMS (actTempl), the container template (workUnitContTempl), input and output artifacts (mandInputSet, optInputSet, outputSet), and roles (primRoleTempl, addRoleTemplSet). Besides, it features connections to concepts for human-centric activity management: the assignment activity (assignActTempl) and a workflow user information (workflowUserInfo) applied in order to abstract from internal workflow logic (cf. Sections 7.4 and 7.5).

For managing cross-workflow connections, the work unit template has two properties connecting it with other container templates or work unit templates. Therefore, the dependency concepts are applied (cf. Section 7.3.1). Property dependsOnTemplSet covers relations to work unit template dependencies and work unit container template dependencies. Note that a work unit template can depend on another work unit container template or a work unit template within another container. The realization of the opposite direction is accomplished by property dependencyTemplSet. It enables a connection to work unit templates depending on the current template. Regarding milestones, the work unit template enables modeling a connection to one or more milestone templates (milestoneTemplSet). We have already discussed the need for extensions to running workflow instances to adhere to the real world process. Therefore, extension point templates may be connected to the work unit template (extensionPointTemplSet). With their properties, they enable the CPM framework to automatically determine respective extensions.

As discussed, the CPM framework favors a separation of control between a WfMS encapsulated by a process management component and the surrounding context management component (cf. Section 7.3). To enable the latter to correctly intervene with workflow enactment in the WfMS, it must have certain information about the workflow structure. This comprises information about which activities are omittable and which ones are repeatable (omittable and repeatable).

In Example 7-11, The Company decided to implement the OpenUP process. The latter only contains standard dependencies (one activity of one workflow is connected to another workflow) among the different workflows. As shown in Example 7-1, another dependency may be useful or even crucial: to be able to model different types of dependencies, their modeling has been moved to the context management component. As discussed in Section 7.3.2, we have implemented two kinds of dependencies. To model the latter, we add two template concepts, the work unit container template dependency and the work unit template dependency. The former connects a work unit template with a work unit container template. The latter works similarly with the difference that it connects two work unit templates in two separate containers. One of these work unit templates (i.e., the source of the dependency) depends on another one (i.e., the target of the dependency). As the target is a work unit template in that case, it might be executed more than once in a LOOP. Therefore, the work unit dependency allows configuring whether the source terminates with the first or the last execution of the target (see Appendix B for formal definitions).

With standard WfMS, the management of complex interconnected human activities as proposed by process models like the OpenUP can become difficult. Example 7-12 illustrates this.

Example 7-12 (Practical issues for process implementation regarding activities):
The Company has decided to implement the OpenUP process. This process contains different types of interconnected activities. A workflow like the ‘Develop Solution Increment’ (cf. Example 7-2) describes a set of concrete activities. However, the whole workflow is also applied to a more complex activity. Further, it has a goal and certain properties relating to the human executing it. Each of the contained activities has specific information and properties regarding the executing human. In addition, the activities comprise activity steps further describing what has to be done. WfMS only have one type of human activity that can be placed on a node in the workflow. With these modeling limitations the accurate realization of the OpenUP is difficult.
To overcome the limitations of WfMS in respect to human activity modeling (cf. Example 7-12), we have introduced concepts for explicit human activity management in Section 7.4. To be able to model these along with the workflows, we have included template concepts containing different properties. The assignment template is applied as a template for a human-related activity. It symbolizes a rather coarse-grained activity (e.g., developing a new software feature) and has references to sub-activities, the assignment activity templates. As the assignment template captures the human activity attached to one workflow template in a WfMS, the assignment activity captures the human activity attached to an activity in such a workflow template. The assignment activity template offers the option to pre-define activity steps; i.e., fine-grained sub-activities applied to help complete the activities that shall support the human and are not governed by a workflow (cf. Example 7-5). As discussed, it is important to connect the workflows in the WfMS with the real-world process executed in the project. Therefore, the assignment activity template enables the pre-definition of relations to atomic task templates. These correspond to small tasks that humans execute with an SE tool (cf. Section 7.4). Thus, the assignment activity templates are the most concrete part of the planned process in the WfMS, while the atomic task templates symbolize the real world process enactment, and both of them can be directly connected. For further discussion of activity management concepts, we refer to Appendix B.

As discussed in Example 7-11, a project and its process contain a myriad of different interconnected artifacts. The latter and their mutual connections cannot be modeled properly within a WfMS using the data elements contained in the workflows. To add facilities to model artifact structures, we consider the concept of the project component template. The latter has a set of properties for defining its type (e.g., ‘PDF file’ or ‘Java artifact’) or responsible roles for it. Another feature of the project component template is the possibility to add various relations to other project component templates. Such relations can be used to model various dependencies of artifacts as required by SE process models like the OpenUP [EcFo15]. By distinguishing between two sub-concepts of the project component template, it becomes possible to define an abstract structure for projects connected to defined area templates. The latter are used to generally specify abstract areas of a project having relations to resources, activities and artifacts. Example 7-13 illustrates this.

Example 7-13 (Project structure with project component templates):
The Company wants to enable automatically supported process enactment. Therefore, artifacts and their relations shall be modeled. For the source code of a project, The Company defines multiple areas, e.g., one area for implementing and one for testing the code (cf. Figure 7-24).

![Diagram](image-url)
Such management of areas facilitates the flexible management of responsibilities in a project: For the implementation, a fine-grained responsibility structure may be created where single developers are responsible for certain source code packages. For the testing area, in turn, more coarse-grained responsibilities might be favored where, for example, one testing team is responsible for the GUI tests of one module.

Process models like OpenUP comprise concrete workflows such as ‘Develop Solution Increment’ (cf. Example 7-2). On that level, human activities may only take a short time and be repeated or switched frequently. In turn, that makes it cumbersome for humans who must set multiple variables in the workflow instance every time they finish an activity so that they get the right activity they want to work on afterwards from the WfMS. To overcome this problem, we have developed a concept in Section 7.5 that enables an abstraction of internal workflow logic making decisions in workflows more human-centric. For defining such human decisions in a workflow, we have included a set of template concepts. The first is the workflow user information. That concept defines one human-centric decision in a workflow being attached to a work unit template. It features references to the different alternatives for that decision being explicitly modeled by the user decision alternative. These concepts allow storing information about the decision and its alternatives (e.g., to proceed with testing code or redesigning the concept behind it) to support the human during enactment. In addition, to map the decisions to variable values usable for the workflow instance in the WfMS, the decision alternative also refers to a workflow variable value concept. The workflow variable template concept defines these workflow variables. Each of these represents one workflow variable in a workflow in the PAIS.

As discussed in Example 7-11, automatic process implementation involves the problem that, due to various dynamic events happening during process enactment, the real process differs from the one enacted by the WfMS. The CPM framework integrates a set of concepts to capture such events and to enable issuing of automated adaptations for realigning the process in the WfMS with the real world process. The concepts for defining such events and reactions are discussed in the following, starting with the event template. That concept allows predefining events that may occur during process enactment in order to react to them. In turn, an event may indicate a problem, for example, when the measured complexity of a source code artifact becomes too high. Therefore, the event template allows for modeling references to a tool template as well as to a problem template. That way, during enactment, specific events can be created based on these templates.

To enable the CPM framework to react to problems automatically, Section 7.6 has introduced the extension point, indicating points where the workflow could be extended and providing information on these points. To predefine such a point, we introduce the concept of the extension point template. The latter features content- and process-related information to distinguish which extensions can be feasible. The extension point template corresponds to the marking of a change to a potentially running workflow instance. Note that there exists substantial work on patterns specifying different kinds of changes [WRR08] as well as the formal semantics [LRW08, LRW09, LRW10] and use cases [ReWe13, WZP14] of these patterns. The CPM framework enables automated changes. In order to avoid making this procedure even more complicated, we refrain from applying complicated patterns, but only choose a simple insertion into the workflow instance (i.e., Pattern AP1 from [WRR08]). For this pattern, three options exist: serial insert, parallel insert, and conditional insert. The third option is redundant, as the added activity would be contemporarily inserted into the workflow instance matching the properties of the situation. In such a case, no further condition is necessary.

To classify the extensions made to the process, we further introduce the concept of the extension template. This concept also has a set of properties classifying it to enable the selection of the right extensions for the right points upon enactment. Further, to define what has to be done by the extension, the template features a reference to an assignment template. In Chapters 9-11, we will discuss more concrete cases for the extension of a workflow.
**Individual Concepts**

When a process with its interconnected entities has been specified based on the template concepts, it can be enacted by the CPM framework. As discussed, each individual enactment of such a process is captured by the individual concepts. In the following, these individual concepts are introduced, starting with an overview of the concepts, their relations, and pointers to the following concrete definitions in Figure 7-25.

![Diagram of Individual Concepts](image)

**Figure 7-25: Individual concepts (expressed as UML class diagram)**

The individual concepts comprise many properties similar to the template concepts, applying the same connections between individuals. Therefore, in the following, we will focus on the differences compared to the template concepts as well as new properties. The basic concepts for extending workflows in a WfMS with additional and contextual properties are the work unit container as a direct mapping of the workflow instance and the work unit as a direct mapping of the activities container in the workflow instance. Recall Example 7-11 where The Company wants to implement the OpenUP process. For illustration purposes, we refer to the ‘Develop Solution Increment’ workflow that governs concrete SE activities. The basic mapping of this workflow to the two aforementioned concepts is shown in Example 7-2. In the following, we discuss these concepts. For brevity, we refrain from mentioning all formal definitions in this chapter. For more information, we refer to Appendix B. We will, however, show the definitions of the two most basic concepts starting with the work unit container in Definition 7.3.

**Definition 7.3 (Work Unit Container)**

A work unit container is a tuple $\text{workUnitCont} = (\text{type, name, wfInstance, workUnitSet, assignment, mandInputSet, optInputSet, outputSet, primRole, addRoleSet, reqContSet, workflowVarSet, basis, noWorkflow, futureExec, pastExec, state, dependencySet, project})$ where

- $\text{wfInstance}$ is the workflow instance annotated by $\text{workUnitCont}$ or undefined.
- $\text{workUnitSet}$ is a finite set of work units being used to map activities within the WfMS.
- assignment is a human assignment or undefined.
- $\text{mandInputSet}$ is a finite set of project components used to define the mandatory input for $\text{workUnitCont}$. 
- optInputSet is a finite set of project components used to define the optional input for workUnitCont.
- outputSet is a finite set of project components used to define the output for workUnitCont.
- primRole is the primary human role responsible for workUnitCont.
- addRoleSet is a finite set of additional human roles.
- reqContSet is a finite set of work unit containers that workUnitCont requires.
- workflowVarSet is a finite set of workflow variables used to control workflow enactment in the WfMS.
- basis is the work unit container template that workUnitCont is based on
- noWorkflow ∈ BOOLEAN indicates a special type of container that contains activities whose sequencing is not specified and thus not governed by a workflow.
- futureExec contains a work unit container that is to be executed as a future iteration of workUnitCont.
- pastExec contains a work unit containers that was executed as a past iteration of workUnitCont.
- state is the state of workUnitCont.
- dependencySet is a finite set of work unit container dependencies.
- project is the project workUnitCont is associated to.

WorkUnitConts describes the set of all definable work unit containers.

We refrain from re-explaining basic properties similar to the ones of the work unit container template. One of the differences to the template is property workflowVarSet. It models relations to all concrete variables used in the work unit container to be passed to the workflow instance producing the concrete enactment trace of the workflow instance. The most important difference of the container as an individual concept is, however, that it is stateful in alignment with the different situations occurring during process enactment. Therefore, we add a finite set of states with deterministic state transitions for all stateful individual concepts. Figure 7-26 shows these for the work unit container.

![Figure 7-26: Work unit container states](image)

As the context management component encapsulates the process management component, the signal to create a new workflow instance (and relating work unit container) is received by the former component. It then creates a work unit container from its template being in the state ‘Created’ (cf. Figure 7-27); i.e., the relating workflow instance has not been started yet. Thus, the container can be attributed with other relating concepts and be connected to other containers. That way, it becomes possible to build a structure of cohesive workflow instances to execute a whole process for a project before starting it. It is even possible to plan multiple iterations of certain workflow instances to be executed after another in a loop. Therefore, the relating containers can be connected to each other via the futureExec and pastExec properties creating a linked list. When the container is executed, an explicit start signal is applied. After that, in turn, a signal is distributed to the process management component to start a new workflow instance based on the related template. When the instance is started, the work unit container enters state ‘Started’. It stays in that state until the signal from the process management component is received indicating that the workflow instance is finished. The activities of such an instance are mapped by the work units (cf. Definition 7.4).

**Definition 7.4 (Work Unit)**
A work unit is a tuple

$$\text{workUnit} = (\text{type}, \text{name}, \text{actInst}, \text{assignAct}, \text{workUnitCont}, \text{mandInputSet}, \text{optInputSet}, \text{outputSet}, \text{primRole}, \text{addRoleSet}, \text{basis}, \text{state}, \text{extensionPointSet}, \text{pastExec}, \text{futureExec}, \text{finalized}, \text{milestoneSet}, \text{dependsOnSet}, \text{dependencySet}, \text{singleExec})$$

where
- actInst is the activity node in the workflow instance that is annotated by workUnit.
- assignAct is the activity a human must process to complete workUnit or undefined.
- workUnitCont is the work unit container workUnit belongs to, i.e., the mapping of the workflow instance that contains the activity annotated by workUnit.
- mandInputSet is a finite set of project components used to define the mandatory input for workUnit.
- optInputSet is a finite set of project components used to define the optional input for workUnit.
- primRole is the concrete human role responsible for workUnit.
- addRoleSet is a finite set of additional human roles.
- basis is the work unit template that workUnit is based on
- state is the state of workUnit.
- extensionPointSet is a finite set of extension points. It marks a point in the workflow where the latter can be automatically extended.
- pastExec is the work unit that has been executed in the previous loop iteration in case the workUnit is placed within a loop.
- futureExec is the work unit that has been executed in the succeeding loop iteration in case workUnit is placed within a loop.
- finalized ∈ BOOLEAN indicates that this instance of workUnit is the final one in the current execution (including no further execution in case the work unit is executed within a loop).
- milestoneSet is a finite set of milestones.
- dependsOnSet is a finite set of work unit (container) dependencies on which workUnit depends.
- dependencySet is a finite set of work unit dependencies.
- singleExec ∈ BOOLEAN indicates if the work unit is meant only for single execution.

WorkUnits describes the set of all definable work units.

An activity of a workflow instance may be executed more than once if placed within a LOOP. To be able to keep the additional and contextual data of each execution separate and traceable, one separate work unit for each execution is used. To manage these potentially multiple work units per activity, we have added several properties: pastExec and futureExec link the work units of the different iterations to each other and property finalized indicates whether or not a work unit will be able to enter another iteration. For work units that were inserted into a running container, property singleExec indicates whether it shall be executed only once. The work unit is a stateful concept as well (cf. Figure 7-27).

![Figure 7-27: Work unit states](image-url)

When a container is created, the contained work units are created as well having state ‘Created’. When the in-process component of a particular activity in the workflow instance is activated, the related work unit enters state ‘Started’ and related information (the assignment activity) is transferred to the human executing the activity. When the human finishes the assignment activity, the related work unit is finished as well. However, due to the XOR pattern, certain activities might not come to execution. As the context management component does not have direct access to the workflow instance (including information on the workflow patterns and the current execution trace), that information cannot be directly received (cf. Section 7.3.1). To still gather it, the execution of certain activities can be used as an indicator for the abortion of other activities. These are called ‘terminator activities’ (cf. Section 7.3.1). When such a terminator activity is executed, related work units enter state ‘Aborted’.

98
Most process models comprise different milestones. An example is the ‘Lifecycle Objectives’ milestone of the OpenUP process that The Company wants to implement. These are already included in the template concepts of the CPM framework. To indicate whether a milestone has been achieved, there is a milestone concept included in the individual concepts.

As discussed, the CPM framework features two different concepts for inter-workflow dependencies: work unit template and work unit container template. Both have a related stateful individual concept: the work unit container dependency allows storing the information whether its target has been executed. Storing such state is more complicated for the work unit dependency whose target is a work unit that might have multiple iterations (cf. Appendix B).

As described in Example 7-11, The Company wants to implement the OpenUP process. The latter contains multiple interconnected activities of different granularity. We have already described the template concepts for respective activities and shown how a mapping executing a particular workflow of the OpenUP process looks like (cf. Example 7-2). Example 7-5 further shows the extension of the basic mapping of the workflow instance using the activity concepts. These concepts require a particular set of runtime properties. Therefore, we integrate the assignment, the assignment activity, and the atomic task as individual concepts.

In most projects, human assignments are planned with their start and end times or duration. The assignment can store this information, as well as textual information that might support the human executing it. Further, it has references to multiple assignment activities that also have properties for time recording and content-related information. The assignment activity is the planned human activity with the finest granularity in the CPM framework. However, it has connections to the more fine-grained activities, i.e., the atomic tasks representing the process as it is actually executed involving humans and SE tools (cf. Appendix B).

Equivalent to the other areas, artifact management has runtime-specific properties as well. Therefore, the project component is added. It provides a stateful concept and models artifacts in projects that may be in various states like ‘Under Review’, ‘Completed’, or ‘Rejected’. Due to the number of possible artifact types and states, the CPM framework does not pre-define the states. Instead, it enables humans to define states for different types of project components in their templates. However, these states are outside the control of the CPM framework and must be set by humans.

We have already introduced a set of concepts enabling the abstraction from internal workflow logic in Section 7.5. However, these are template concepts for modeling that human-centric abstraction. To store the data element values of a concrete workflow instance upon enactment, we add the workflow variable.

To discuss a final area for the individual concepts, we add concepts for the dynamic change of workflows. We have already introduced the extension point and extension templates; however, for one concrete instance of a workflow, runtime properties must be saved for its adaptation. Therefore, we add the concepts of the extension point and the extension. They enable to create concrete assignments and integrate them dynamically into a running work unit container. Concrete cases for this will be discussed in the succeeding chapters (see also Appendix B).

### 7.7.2. Consistency Checks

Due to the large number of concepts, the various kinds of information processed, and the fact that much information depends on the human involved, it is not possible to guarantee correct execution in every situation with justifiable effort. Some facts are simply not under the control of the CPM framework, but are directly managed by humans; e.g., the different sets of states for different types of artifacts. However, to limit execution problems we define a set of basic consistency checks that avoid a number of obvious inconsistencies that may cause problems we encountered when testing and applying the CPM framework. In the following, we briefly discuss these checks. The checks we
consider deal with the relation of template and individual concepts, work unit containers, work units, omittable and repeatable activities, dependencies, and variables. For the sake of brevity, we only present the formal definition of one of these check sets. For an illustration of the other checks we refer to Appendix B.

The concept for mapping workflows in the context management component implies certain definitions regarding omittable and repeatable activities (cf. Section 7.3.1). Such properties might be defined in a way that prevents correct execution. One such definition concerns multiple instances of a work unit that has not been defined as ‘repeatable’. Another definition concerns work units not coming to execution at least once, but that have not been defined as ‘omittable’. A third one concerns work units that have been marked as ‘finalized’, but come to execution again. Figure 7-28 illustrates these three undesired scenarios. In turn, Definition 7.5 formalizes the checks preventing these cases.

**Definition 7.5 (Repeatable and Omittable Activities)**

Let $WU \in \text{WorkUnits}$ be a work unit and $WUT \in \text{WorkUnitTempls}$ be a work unit template within the framework. Then:

a) $\neg(WU.\text{basis} = WUT \land WUT.\text{repeatable} = \text{FALSE} \land (WU.\text{futureExec} \neq \text{NULL} \lor WU.\text{pastExec} \neq \text{NULL}))$,

i.e., there can only be multiple instances of a work unit if its relating template is defined as repeatable.

b) $\neg(WU.\text{basis} = WUT \land WUT.\text{omittable} = \text{FALSE} \land WU.\text{finalized} = \text{TRUE} \land WU.\text{state} = \text{“created”})$,

i.e., a work unit can only be finalized without having been executed if it is defined as omittable by its template.

c) $\neg(WU.\text{finalized} = \text{TRUE} \land WU.\text{futureExec} \neq \text{NULL})$,

i.e., a finalized work unit cannot be executed a second time (with a new instance).

### 7.7.3. Algorithms for Marking Workflows

The CPM framework provides comprehensive support for SE projects and, in particular, for their processes. As described, this involves a set of components for various tasks like gathering contextual data, dynamically processing a multitude of data sets, or enacting the workflows being parts of the SE process models. Therefore, we have decided to rely on a set of technologies already available (cf. Chapter 6) and to build the CPM framework as a framework uniting these technologies. The context management and process management components constitute the core for enacting the processes in alignment with the contextual data of the SE project. As the context management component
coordinates all other components, it needs comprehensive access to their data. In the first place, this concerns the process management component, and, in turn, the workflows managed by it. The concepts described in this chapter extend these workflows in several aspects and thus require exact information about the enactment properties and workflow state. This raises the following issue: The workflows are managed within a WfMS integrated into the process management component. In turn, this WfMS provides crucial capabilities like correctness and adaptability of these workflows. However, getting exact information about the enactment of all workflows and the contained activities is problematic. Activities might be executed multiple times or be omitted completely. This has impact on the concepts extending these activities, like work units or assignment activities, in that activities might be erroneously presented to humans or might not reach them despite their enactment in the WfMS.

To overcome this issue, this section introduces the algorithms we created for analyzing workflows. That way, the context management component is provided with information about them. Workflows are managed by the WfMS being in charge of the sequencing of the activities. When an activity is executed, the control is passed to the context management component, which then can execute the activities, contextually utilizing all concepts introduced in the previous sections. However, to the context management component, the enactment of the workflows looks like a black box. This can be problematic at presence of activities executed multiple times or omitted completely. When an activity is omitted, the status of the corresponding concepts in the context management component must be set to ‘Aborted’. When an activity is executed multiple times, new instances of the corresponding concepts must be created. To solve this problem, a set of algorithms analyzes workflow templates when these templates are created. In this context, they assume block-structured workflows that are sound. This must be guaranteed by the WfMS. These algorithms add a number of markings to the work unit templates. To simplify and accelerate processing, the algorithms work on a specific representation of the workflow that consists of simple lists and entities.

To provide this structure of entities and lists, we create a workflow decomposition algorithm that takes a workflow template as input. We refrain from explaining that algorithm in detail as there has been other work in that area that discusses workflow decomposition in detail. As example consider the process structure tree [VVK08, VVK09]. The latter enables the decomposition of a workflow graph into a hierarchy of sub-workflows having a single entry and a single exit of control. It is a general-purpose approach and introduces a fair amount of complexity. Our algorithm can be compared to the latter but is much simpler and tailored to output exactly the data structures we need for the CPM markings of the workflows. However, to support understanding of other algorithms of this work that utilize the output of this algorithm, we will give a brief introduction to the algorithm in the following.

This algorithm traverses the entire workflow template to generate a nested structure of blocks. This is illustrated in Figure 7-29. The output of the algorithm is an ordered list of activities representing the workflow template. This list only contains sequentially connected activities and no workflow patterns. To allow for a mapping of the structure of the workflow template, the contained activities may be simple activities or other structures, like a part of a workflow template surrounded by an AND split and AND join (cf. Activities ‘3’ and ‘4’ in Figure 7-29A). That way, the algorithm recursively decomposes the workflow template into blocks. Therefore, an object called block is used. The latter contains the type of the patterns, like XOR or AND, and a collection of branches that correspond to the outgoing arcs of a pattern. Each of these branches is realized as an ordered list of activities. As input, the algorithm needs the workflow template represented as a list of nodes and arcs, a stack used for the recursive operations, an initialized empty list used for storing the output, and the current position in the workflow template represented by an arc. The latter is needed as the algorithm is recursive, so that, for each call, a marker for the current position in the workflow template is available. As such a marker, we use an arc in the workflow. Thus, the algorithm can be called recursively for each branch of a pattern like XOR.

Example 7-14 illustrates the outcome of the workflow decomposition algorithm.
Example 7-14 (Workflow decomposition):
The workflow template below uses all basic workflow patterns and shows the decomposition into blocks by means of these patterns.

![Workflow diagram]

Initial Activity Marking

The context management component should know whether an activity may be executed more than once or might be omitted completely. Therefore, the decomposed workflow templates are analyzed to be able to mark such activities for the context management component. Note that the repeated or omitted execution of an activity corresponds to XOR and LOOP patterns in the workflow templates. An example of such activities is illustrated in Figure 7-30.

![Terminator diagram]

Figure 7-29: Workflow decomposition

Figure 7-30: Terminator activities
Activity ‘1’ is placed within a LOOP pattern and can thus be executed multiple times, while Activity ‘2’ is placed within a XOR pattern with an empty branch and may thus be omitted completely. Activities ‘3’ and ‘4’ are both repeatable and omittable due to the surrounding patterns. The algorithms that mark omittable and repeatable activities using the shown decomposed workflow template lists are relatively simple (cf. Appendix B). However, the marking of such activities itself does not suffice for the context management component. First, the latter must be aware of points during workflow enactment where it is clear that omittable activities are actually omitted in the context of a particular workflow instance. This is crucial because the concepts in the context management component relating to that activity (e.g., work units) must be deactivated. Furthermore, if an activity in another workflow depends on that activity by the ‘depends on work unit’ connection, that activity must be allowed to terminate to not create a deadlock. In addition, the CPM framework must be aware of the time points when it is clear that repeatable activities will not be executed another time. This, in turn, is crucial for activities in other workflows depending on the repeatable activity by the ‘depends on work unit’ connection. The latter can be configured to be satisfied when the repeatable activity is executed the first or the last time. In the latter case, the execution of the terminator activity marks the time point when the dependency is satisfied.

As aforementioned, for the context management component, workflow enactment is like a black box and control is only transferred to it when an activity becomes activated. Therefore, the execution of certain activities is taken as such a point in time. Respective points are illustrated in Figure 7-30: The first point is the execution of Activity ‘2’ that indicates that the repeatable Activity ‘1’ will not be repeated any more. In turn, Activity ‘2’ is surrounded by an XOR pattern and thus may be omitted. Therefore, the execution of the mutually exclusive Activities ‘3’ and ‘4’ both mark such a point. Furthermore, as they are mutually exclusive, each of them also marks a point when the other one will not come to execution. Both of these activities are additionally surrounded by a LOOP pattern and could thus be executed more than once. Since there are no succeeding activities in place, the termination of the workflow instance itself marks the point in its execution where no further execution of the activities will occur. As illustrated, there are different kinds of these points in a workflow. For both repeatable and omittable activities this is denoted as activity deactivation. In the following, the generic cases are briefly described:

1. Mutually executed activities deactivate each other: In the simplest case, an XOR has two mutually executed branches that mutually deactivate their activities. If the branches do not contain simple activities, but other patterns, their activities deactivate the other branch of the XOR pattern.
2. Omittable activities are deactivated by succeeding activities: In this case, one branch of the XOR pattern is empty, making the activities in the other branch optional. These optional activities are deactivated when the first activity after the XOR pattern is executed. As in the first case, multiple activities may be candidates for this if the considered XOR pattern is not succeeded by a simple activity but by another pattern. It is further possible that a mixture of the first case with the second case occurs: An XOR pattern has more than two branches of which one contains no activities. In that case, the set of activities whose activation can deactivate activities of one branch in the XOR pattern must be extended: In particular, it must contain the first activity of each other branch as well as the first activity executed after the considered XOR pattern.
3. Omittable activities are deactivated upon workflow termination: This is a specialization of case 2. When no succeeding activities are in place between the considered branch and the end of the workflow template, deactivation happens upon workflow termination.
4. Repeatable activities are deactivated by succeeding activities. This is similar to case 2.
5. Repeatable activities are deactivated upon workflow termination. This is similar to case 3.

Algorithm 7-1 presents algorithm getTerminatorActivities that determines activities whose execution marks a point in the enactment of the workflow instance where certain other activities will no longer be executed. These activities are called terminator activities.
Algorithm 7-1: getTerminatorActivities (Pseudo Code for determination of terminator activities)

Require: Decomposed Workflow list P {Blocks, Activities}, List targetBranch, List terminatorActivities, Boolean getActivities, Boolean topLevel
Return: Boolean foundTerminatorActivity

1: for all elements in P do
2:     if element ∈ activities and getActivities
3:         terminatorActivities.add(element)
4:         return true
5:     else if element ∈ blocks
6:         Boolean thisBlockComplete ← true
7:         if element ∈ ands
8:             thisBlockComplete ← false
9:     end if
10:    Boolean changedGetActivities ← false
11:    Boolean oldValueGetActivities ← getActivities
12:    if element.contains targetBranch
13:        getActivities ← true
14:    end if
15:    Boolean emptyBranch ← false
16:    for all element.branches do
17:        if branch.isEmpty() and element ∈ xors
18:            emptyBranch ← true
19:        end if
20:        if branch ≠ targetBranch
21:            if element ∈ xors and not emptyBranch
22:                thisBlockComplete← thisBlockComplete and
23:                getTerminatorActivities(branch, targetBranch, terminatorActivities, getActivities, false)
24:            end if
25:            if element ∈ ands and not changedGetActivities
26:                thisBlockComplete← thisBlockComplete or
27:                getTerminatorActivities(branch, targetBranch, terminatorActivities, getActivities, false)
28:            end if
29:            if oldValueGetActivities ≠ getActivities
30:                changedGetActivities ← true
31:            end if
32:        end if
33:        if element ∈ loops
34:            thisBlockComplete← getTerminatorActivities(branch, targetBranch, terminatorActivities, getActivities, false)
35:        end if
36:    end for
37:    if oldValueGetActivities ≠ getActivities
38:        changedGetActivities ← true
39:    end if
40:    if emptyBranch
41:        thisBlockComplete ← false
42:    end if
43:    if thisBlockComplete and getActivities and not changedGetActivities
44:        return true
45:    end if
46: end if
47: end for
48: if topLevel
49:    terminatorActivities.add(workflowElement)
50: end if
51: return false
Algorithm 7-1 takes as input the decomposed workflow represented as a list we have discussed as well as a list representing a branch (target branch) for which the terminator activities should be determined. It further expects a list of these terminator activities (empty at the beginning and passed on as a reference) and a Boolean variable getActivities to indicate if, for the current execution of the algorithm, the detection of terminator activities should be executed. The last parameter is a Boolean variable topLevel indicating whether the algorithm is called from the outside or recursively by itself. Note that the whole workflow template has to be processed, since it is possible that the treated branch is nested within other patterns that, in turn, might contain terminator activities. Therefore, all activities are processed even if they are predecessors of the considered branch, and the detection of terminator activities is started when the target branch is encountered. When the algorithm encounters a simple activity and getActivities is true, it adds that activity to the terminator activities and returns a true Boolean value (Lines 2-4). This value is used when the algorithm recursively calls itself to determine if the currently called recursion detected the desired terminator activities.

If a block is encountered, it has to be determined, if for all branches in that block, terminator activities can be determined using another Boolean variable thisBlockComplete (Lines 5-9). To ensure correct processing of different blocks (Lines 21-39), this variable is initialized with different values according to the current block. If the processed block contains the target branch, the detection is enabled (Lines 12-14). After that, all branches except the target branch (if it is in place in that block) are processed (Lines 16 and 20). However, if the current block is a XOR and one of the branches is empty, the information is stored in variable ‘emptyBranch’ (Lines 17-19). For each branch of the block the algorithm recursively calls itself and uses the Boolean output value to determine if all of the branches of the current block contain terminator activities (Lines 21-39). This is done in different ways for XOR, AND, and LOOP blocks since for a XOR block each branch has to contain at least one terminator activity, whilst for an AND block one branch with a terminator node suffices, and a LOOP block only has one branch. If the currently processed block has terminator activities in all branches and activity detection is activated, the algorithm terminates and returns a true Boolean value (Lines 47-48).

However, this involves a set of special cases: If the current block is a XOR with an empty branch, succeeding blocks or activities must also be considered and the algorithm may not yet terminate. Therefore, the information stored in variable ‘emptyBranch’ is utilized (Lines 44-46). In an XOR block, branches mutually deactivate each other. If an empty branch is present, also succeeding activities must be taken into account. Lines 21-39 of the algorithm can be used to 1) determine the terminator activities of mutually exclusive XOR branches if one is the target branch, and 2) to process the blocks that are successors of the target branch block. For these, the algorithm can terminate only if all relevant branches contain a terminator activity. A counter example is an XOR block with an empty branch. For such a block, the algorithm would have to continue the search for succeeding terminator activities. The AND block has to be treated in a different way: If one of its branches contains the target branch (via nested blocks), the activities in other branches shall not be used as terminator activities. To correctly process all three blocks and take into account target branches that are contained in nested blocks within a block of a block, we have added the following: In Line 10 and 11 variables are initialized to indicate, whether the getActivities variable has changed while processing the block. This occurs when the target branch is encountered in a block contained in the currently processed block. Therefore, we check this in Line 40-42. The algorithm may only terminate and return true, if all branches contain terminator activities (thisBlockComplete) and the target branch was not found in the current block (getActivities is true and its value was not changed in the current block, Line 47). To prevent the addition of terminator activities from AND branches when the AND block contains the target branch, we also add the check for a change in getActivities to the AND processing (Line 26 and Lines 30-32).

If the traversal through the current workflow list is completed and the list represents the entire workflow template (not a single branch), that means that for at least one branch of one block a terminator activity might not be determined, and thus the workflow template itself is added to the terminator activities.
Example 7-15 illustrates the getTerminatorActivities algorithm:

Example 7-15 (Terminator activity acquisition steps): Consider the workflow shown in Figure 7-31 as an example and assume that the branch inside LOOP1 is selected as the target branch for which terminator activities shall be found. As illustrated in Figure 7-31, the algorithm determines Activity ‘5’ as the terminator activity for that branch.

If activities shall be repeated, the corresponding concepts in the context management component (work units, assignment activities, etc.) have to be executed more than once. In such a case, the CPM framework resets these concepts and uses them multiple times. To be able to have the data of each execution of an activity available after execution, new instances of the respective work units (and associated concepts) are created during execution when the respective activity is executed more than once. When the execution reaches a completed but repeatable work unit, a new instance of the latter is created and connected with the previously completed instance.

For omittable activities the situation is more complicated: Their work units have to be deactivated during runtime so that it is clear after execution which activities (and related concepts in the context management component) have been executed. Therefore, the points in the workflow have to be
determined when it is clear that such activities will not come to execution anymore. These points correspond with the execution of terminator activities as described in Algorithm 7-1. Consider, for example, an XOR pattern that contains two mutually executed activities: If the first activity is executed, the second will not be executed and vice versa (cf. Appendix B). Algorithm B-1 is utilized to mark omittable activities and connect them with other activities upon their execution it is clear that the marked activity has been omitted.

The workflow template from Example 7-14 is taken to demonstrate the combination of omittable and repeatable activities as well as the application of the respective algorithms to a workflow template. Example 7-16 illustrates the extensions of that workflow template.

Example 7-16 (Execution properties of activities):
Figure 7-32 shows a workflow template including markings for repeatable and omittable activities.

![Workflow Template Diagram](image)

**Figure 7-32: Execution properties of activities**

Activities 7-10 are located within a LOOP pattern and are thus marked as repeatable. The simplest case for omittable activities in which two activities are mutually exclusive applies to Activities ‘5’ and ‘6’ as well as to Activities ‘9’ and ‘10’. Therefore, these activities are marked to mutually deactivate each other upon execution. All activities in the workflow template are surrounded by an XOR making them optional. That XOR pattern has no activities contained in the alternative branch and no succeeding activities in that workflow template that might deactivate all the activities. Thus, the workflow instance will deactivate the activities upon termination. Activities ‘9’ and ‘10’ demonstrate the combination of the properties ‘repeatable’ and ‘omittable’ that do not interfere. If one of these activities is executed, it will deactivate the other one. If workflow enactment reaches one of those activities more than once, a new instance will be generated and connected to the formerly executed. In the first LOOP iteration, instances of all contained activities are in place (7-10) and, thus, e.g., the execution of Activity ‘10’ deactivates the instance of Activity ‘9’. If the LOOP is executed more than once, the new instances are only generated as needed and, if Activity 10 is executed, no new instance for Activity ‘9’ will be generated. In that case, the deactivation procedure only checks whether an active instance is in place. Concerning LOOP termination, the figure shows one LOOP. Since no successor activities are in place, the termination of the workflow instance is taken as the point in the execution where no further execution of the looped activities occurs.
Adaptation Markings

There are other issues to be considered when workflow instances are adapted; i.e., when activities are added to the workflow instance. If the activity becomes inserted within an XOR pattern, it may eventually not be executed at all and, if placed within a LOOP pattern, it can be executed more than once. In case of an adaptation, the workflow instance is already running and not in the state as defined by its template, as activities might have been added. However, these new activities must be marked as all the other ones in the workflow instance. Therefore, the workflow list created from the workflow template is kept and adapted for each workflow instance. This involves the marking of a new activity, the potential connection to its terminator activities, and connections to other activities for which it might be the terminator activity. For example, for activity ‘2’ in Figure 7-33, this includes a ‘repeatable’ and an ‘omittable’ marking, the fact that it is terminated by the execution of activity ‘6’ and ‘4’, and that its execution terminates activity ‘1’.

To apply the markings for a new activity, the template is not reanalyzed. This would entail the redundant execution of all algorithms for marking, which would be inefficient, especially during execution. Instead, the workflow list created by the workflow decomposition algorithm is reused as well as the markings and connections already in place. When an adaptation occurs, the workflow list is also adapted, so that it represents the current state of the workflow instance. Therefore, we create a separate algorithm applying the markings for a new activity based on its surrounding activities at runtime. This is far more efficient than re-running all other algorithms for marking again. For a thorough discussion of the algorithm and the computational complexity of all algorithms, see Appendix B.

7.7.4. Basic Actions for Software Engineering Process Enactment

In the preceding sections, we have presented the concepts for contextually extending process management in order to enable various features like extended activity modeling, human-centric abstraction of internal workflow logic, and automated process adaptations. Further, we have introduced the formal framework, specifying the concepts for the CPM framework and a set of created algorithms enabling contextual process enactment with that approach. In addition, we provide descriptions of concrete actions needed for executing an SE process based on the CPM framework, such as creating individual concepts from the template concepts, checking whether a work unit may terminate, or adding new dependencies between different containers. These actions are defined, including their different steps, their input and output, and preconditions. Table 7-2 gives an overview of the different actions including a short explanation.
Table 7-2: Basic actions for enactment

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Project</td>
<td>Creates the individual concepts for a project from a project template.</td>
</tr>
<tr>
<td>Create Work Unit Container</td>
<td>Creates a new work unit container from a work unit container template.</td>
</tr>
<tr>
<td>Check Work Unit Termination</td>
<td>Checks if a work unit may terminate.</td>
</tr>
<tr>
<td>Create new Work Unit Instance</td>
<td>Creates a new Work Unit instance for a looped activity.</td>
</tr>
<tr>
<td>Add Work Unit to Work Unit Dependency</td>
<td>Creates a new dependency between a source work unit and a target work unit.</td>
</tr>
<tr>
<td>Add Work Unit to Container Dependency</td>
<td>Creates a new dependency between a source work unit and a target work unit container.</td>
</tr>
<tr>
<td>Add Container Dependency</td>
<td>Creates new dependencies between the work units in a source container and the work units in a target container.</td>
</tr>
<tr>
<td>Remove Work Unit Dependency</td>
<td>Removes a dependency between a source work unit and a target work unit or a container.</td>
</tr>
<tr>
<td>Remove Container Dependency</td>
<td>Removes mutual dependencies between work units in a source work unit container and a target work unit container.</td>
</tr>
<tr>
<td>Move Work Unit Dependency</td>
<td>Moves a dependency between an old source work unit and a target work unit or a container to a new source work unit.</td>
</tr>
<tr>
<td>Move Container Dependency</td>
<td>Moves mutual dependencies between work units in a source work unit container and a target container to work units in another source work unit container.</td>
</tr>
<tr>
<td>Start Work Unit Container</td>
<td>Starts a work unit container (includes workflow start in PAIS).</td>
</tr>
<tr>
<td>Distribute Activity</td>
<td>Changes the executing person of an assignment activity.</td>
</tr>
<tr>
<td>Distribute Activities</td>
<td>Changes the executing person of multiple assignment activities belonging to one assignment.</td>
</tr>
<tr>
<td>Distribute Assignment</td>
<td>Changes the executing person of an assignment and all of his related assignment activities belonging to that assignment.</td>
</tr>
</tbody>
</table>

For the sake of brevity, we refrain from discussing all actions in detail here. We rather show one of these actions as an example in the following and refer the reader to the Appendix C for the other ones.

**Create Work Unit Container**

This action is applied to create a new work unit container from a work unit container template. As opposed to a WfMS where workflow instances are directly created and started based on their templates, the containers in the CPM framework are created without starting them (or the relating WfMS workflow instances). Thus, a workflow structure for the complete process of a project can be created without having to start each of the future workflow instances.

**Preconditions:** -
**Input:** work unit container template ∧ values for roles, project components, and tools

**Actions:**
- Create work units as defined in the template and assign it to the work unit container.
- Create assignment as defined in the template and assign it to the work unit container.
- Create assignment activities as defined in the template and assign it to the assignment.
- Create atomic tasks as defined in the template and assign it to the assignment activities.
- Assign concrete tools to atomic tasks as defined in the template.
- Set process variables as defined in the template.
- Assign concrete humans for the container roles.
- Assign concrete inputs/outputs for the container (including structure of project components as defined in super/subCompsSet properties).
Assign the main human with the main role also to the assignment. Distribute the humans filling the roles of the container to the work units. Add the responsible party of each work unit to the relating assignment activity.

For all defined dependencies defined by the template for work units, apply the action ‘Create work unit container’ to create the containers (and work units) that are the targets of the dependencies and then connect them via the ‘Add work unit to work unit dependency’ and ‘Add work unit to container dependency’ actions.

Output: work unit container in state ‘Created’.

After a concrete work unit container has been created, it remains in the state created and does not automatically initiate the start of its relating workflow instance. As an advantage, for a project, its entire process can be prepared with a workflow structure without having to start one or more of the involved work unit containers or workflow instances. When all concepts and information is in place, a work unit container can be explicitly started including the creation / start of its relating workflow instance. Thus, to start a project, its top-level container will have to be started.

7.8. Discussion

Processes are beneficial in various domains and automatic process enactment support is desirable. This chapter has proposed a contextually-extended process enactment approach for SE projects that enables more holistic support of the project and its process. To the best of our knowledge, no directly comparable approach exists in related work. However, many approaches enable distinct functions comparable to the ones of the CPM framework. This section will briefly discuss basic workflow enactment approaches before elaborating on dynamic process approaches enabling workflow configuration, as well as manual and automated adaptation. Finally, this section goes into detail about contextually-integrated and semantically-extended processes before providing a short summary.

7.8.1. Process Enactment Support

Process enactment support deals with the automated enactment of the modeled process or its parts. It is mostly realized based on workflow management. However, traditional WFMSs lack expressive power in modeling so, in most cases, an entire process model cannot be implemented using workflow management technology. Most of these systems strongly focus on the sequencing of the activities and the governance of these while neglecting other aspects discussed in this chapter. This concerns, for example, complex artifact and related activity hierarchies or various types of additional information for humans like e.g., checklists. In addition, the connection of hierarchically dependent workflows is strictly limited to the connection of the sub-workflow to an activity of the super-workflow. As workflow management is only a very basic technology in relation to providing holistic project and process support, this section only briefly introduces an example of three different kinds of available WFMSs. The first example, YAWL (Yet Another Workflow Language) [vdtH05] features facilities for complex data transformations and an integration of organizational resources. Further, it is based on a formal foundation that allows for unambiguous specification and automated verification of the workflows. An example stemming from the open source community is the WFMS jBPM [Cumb07]. As opposed to scientific approaches, it lacks formal grounding and sophisticated verification facilities. However, its focus lies more in the programmatic access, extensibility, and easy integration into Java enterprise applications. The third example, Intalio [Inta15], is a commercial product with an open source foundation. It enables workflow specification and execution based on BPEL [OASI07]. Its main feature is enabling a zero code approach for workflow modeling and execution.

7.8.2. Dynamic Processes

As discussed, SE processes are dynamic. Consequently, a tool aiming to provide holistic support for an SE project and its process must be able to incorporate dynamic workflows. This section covers four different types of approaches in this area: configurable processes, processes that can be aligned to
products or artifacts, dynamically adaptable processes, and approaches that enable the latter adaptations automatically. Another way to incorporate dynamicity into enactable workflows by declaratively specifying them will be covered in Chapter 8.

**Process Configurations**

An important area of related work are configurable processes, wherein the workflows used for enactment can be configured. These permit specifying one reference workflow and specific configurations to it that make it applicable for different situations. On one hand, that makes the workflow applicable for different variants of the same problem; on the other, it keeps the number of modeled workflows and associated complexity low. There exist different approaches to process configuration that are briefly introduced in the following.

One way to enable configurable processes is to incorporate configurable elements into the workflows [LDTM11, RoAa05]. An example of this is a configurable activity. Such an activity can be integrated or omitted or even optionally integrated surrounded by XOR gates. Another approach enabling process model configuration is ADOM [RSS09, RSS10], which builds on software engineering principles and also allows for the specification of guidelines and constraints with the process model. A different approach to process configuration is called behavior-based configuration. It allows the modeler to specify predefined adaptations to the execution workflow’s behavior. An option for realizing this is hiding and blocking [GAJL08, Gott09]. By blocking, this approach supports disabling the occurrence of a single activity/event and thus prevents one edge from becoming active. The other option supported by this approach is hiding, which allows a single activity to be hidden. That way, the intended activity is executed silently but succeeding activities in that path are still accessible.

Another way for enabling process configuration is called structural configuration. It is grounded on the observation that process variants are often created by humans by simply copying a process model and then applying adaptations to it. A sophisticated approach dealing with such cases is Provop [HBR08a, HBR10, RHB15], which enables variants of execution workflows by storing a base workflows and pre-configured adaptations to it. The latter can also be related to context variables to enable the application of changes matching different situations [HBR08b].

After enabling workflow variant and configuration management, the configuration for a specific situation must be applied. This is mostly done manually by a human and should be supported. One approach to this is to abstract the variant configuration for the human by providing a questionnaire whose answers are mapped to certain workflow configuration steps [RLS07, LDH09]. Another way to abstract and facilitate variant management is provided by feature diagrams [SHT06] that originated from product line management. These diagrams offer a structured way to describe the common and the varying parts of an item. The aforementioned Provop approach also aims to assist the human by utilizing context knowledge and thus only provide the changes that match the parameters of the current situation [HBR08b].

The above approaches for process configuration enable the alignment of one base process to various different situations. This is an important factor for making them applicable in domains with high dynamicity like the SE domain. For more information on this topic, we refer to [ATW+14]. However, most of them only enable configuration prior to execution and exclude dynamic variability during runtime. Furthermore, they only support manual configurations applied by humans. In complicated projects, e.g., in the SE domain, this can be problematic, as one human might not have all necessary information at hand to apply such a configuration effectively.

**Artifact-centric Processes**

Artifact-centric processes interpret complex data structures representing a product in order to derive related workflow structures. Corepro, for example, allows product engineers to define complex data structures and to semi-automatically derive workflow structures from them [MHHR06, MRH08]. The
latter comprise the concrete workflows for engineering a particular product component (i.e., part) as well as the required synchronization between them. In particular, dynamic changes of a product structure are automatically compiled into respective adaptations of the workflow structure (on condition that certain correctness constraints are met, cf. [MRH08]). Corepro uses object life cycles and their dependencies in order to represent product components and their relations. The Business Artefacts approach [LBW07, BHS09] is a data-driven methodology that focuses on business artefacts rather than activities. These artefacts hold the information about the current situation and thus determine how the process shall be executed. In particular, all executed activities are tied to the life cycle of the business artefacts. Recently, more generic approaches aiming at a tighter integration of process and data have emerged (see [KWR11, KüRe11b] for an overview). These are particularly interesting for enabling artifact-based processes as in SE. For example, PHILharmonicFlows enables object-aware processes, which consider object behavior (i.e., the behavior of single objects and artifacts respectively) as well as object interactions (i.e., the coordinated processing of a collection of objects) [KüRe11a]. Consequently, object-aware processes are based on two levels of granularity. In particular, data-driven process enactment is enabled as well as integrated access to processes and data [KüRe11c, Künz13].

Artifact-centric process approaches manage the important aspect of integrating the influence of artifacts on processes well. Compared to the approach presented in this work, however, they lack other features for enabling comprehensive support, such as modeling additional contextual factors or enabling automated reactions to them.

Process Adaptations

During the course of a project, various situations might occur in which the parameters for process enactment change. This means that the running workflows do not conform to the current situation. While a number of such situations might be incorporated into the workflow models from the beginning [RDB03], this would bloat the models and make them difficult to understand and maintain. In light of these facts, adaptive WfMS have been developed that incorporate the ability to change a running workflow instance to conform to a changing situation. Examples for such systems are Breeze [SMO00], WASA [Wesk00, Wesk01], SPADE [BFGL94], and ADEPT [ReDa98, DaRe09, ReDa09]. However, these only permit manual adaptation carried out by a human. Furthermore, the workflow instances have to be manually suspended from execution and loaded into a process editor to apply adaptations. This can be cumbersome and, in some situations, a human might not have all necessary information available to apply a semantically suitable adaptation. This applies especially for SE projects where many loosely connected areas with their own tools exist. For example, integrating a software quality measure into a human’s workflow depends on knowledge from different and unconnected tools like task management systems, IDEs, and static code analysis tools.

An important issue for workflow instance adaptations is that the exceptional situations leading to the adaptation can occur more than once. In that case, knowledge about the previous changes should be exploited to extend effectiveness and efficiency of the current change [DRK00, LeRe07, MTB07]. In case a human shall apply the adaptations, approaches like ProCycle [WRWR09, WRW05]), [WWB04]) or CAKE2 [MTS08] aim to support him with that knowledge. In these approaches, changes are annotated with additional contextual information utilizing Case-Based Reasoning (CBR) [Kolo92].

However, even if the human is supported by such a system that potentially improves his choices, manual adaptations are still cumbersome. Furthermore, in SE this would often involve situations in which one human from one project area (e.g., the quality manager) changes the running workflow of a human of another area (e.g., a developer). This can be problematic as this action might interfere with the intentions of the human whose workflow has been changed. In addition, the question remains if the human applying the change has all necessary information from the other project areas.
Automated Process Adaptation

There exist several approaches supporting automated and dynamic adaptations of workflows during run-time [WSR09]. As in our approach, their aim is to reduce error-prone and costly manual workflow adaptations during run-time and thus to relieve humans from this task. As opposed to this work, the focus of those approaches is only on automated exception handling. For this, the process-aware information system must be able to automatically detect exceptional situations, derive the dynamic change necessary to handle them, identify the workflows to be adapted, correctly apply the dynamic change to these workflows, and notify respective humans. Existing approaches can be classified according to the basic method used for automatic exception detection and workflow adaptation:

Rule-based approaches are one area of approaches enabling automated workflow adaptations. ECA-based (Event-Condition-Action) models are suggested for automatically detecting exceptional situations and determining the actions (i.e., workflow adaptations) required to handle them. In many ECA approaches, however, adaptations are restricted to currently enabled and running activities (e.g., to abort, redo, or skip activity execution) [CCPP99]. By contrast, AgentWork [MGR04] further enables automated adaptations of the yet not entered regions of a running workflow (e.g., to add or delete activities). Basic to this is a temporal ECA rule model that allows specifying process adaptations at an abstract level and independent from a particular process model. When an ECA rule fires during run-time, temporal estimates are made to determine which parts of a running process instance are affected by the identified exception. These parts are then adapted immediately (predictive change) or, if this is not possible due to temporal uncertainty, at the time they are entered (reactive change).

Goal-based approaches formalize process goals (e.g., process outputs) and automatically derive the process model (i.e., the activities to be performed and their execution order) based on which of these goals can be achieved. Further, if an exception (e.g., an activity failure) occurs during run-time that violates the formal goals, the process instance model is adapted accordingly. In ACT [BeKi99] for example, certain workflow adaptations (e.g., replacing a failed activity by an alternative one) are automatically performed if an activity failure leads to a goal violation. EPOS [LiCo93] rewrites software engineering workflows when process goals themselves change. Both approaches apply planning techniques to automatically derive and repair workflows in such cases. However, current planning methods do not cover all relevant process scenarios like our approach, since important aspects (e.g., treatment of loops, appropriate handling of data flow) are not adequately considered.

A recent area for dynamic processes are so-called smart processes and the relating systems. An example of this category is SmartPM [deLe09, MMS14], a process management system that incorporates various techniques for reacting to unforeseen situations. That way it can recover from exceptions by applying automated adaptations. Another approach that enables advanced process exception handling is presented in [ElLu10], where recovery strategies are proposed for process fragments that still preserve their flexibility. Finally, the approach presented in [FFM10] enables advanced exception handling: it applies a model-based approach enabling the repair of the process and its activities. The approach can even assess reparability by analyzing the process structure and defined repair actions.

The approaches discussed in this section enable automated process adaptations that are in some way similar to the adaptations of the approach presented in this work. However, they strongly focus on this feature and utilize it only for a relatively narrow area: exception handling. As opposed to this, our approach enables comprehensive support for an entire SE project and its process and seamlessly integrates automatic adaptations to support.

7.8.3. Contextual Process Support / Integration

This section deals with approaches targeting at a contextual support for processes. Primarily, this comprises approaches providing concrete extensions to processes enabling contextual integration during process enactment. To enable contextual process extensions, many approaches exist, addressing different aspects of integration with different technologies. For example, issues include the reuse of process models across different languages, the connection of processes of different organizations, or
the connection of business and IT views on processes. Technically, there have been various approaches utilizing various technologies like web services, Petri nets, or process modeling notations.

**Context Integration**

Adapting application services to contextual changes is a major research area in areas like pervasive computing. A number of context-aware frameworks have been suggested to facilitate the implementation of application services that can adapt their behavior to changing context. Frameworks like Context Management [KMK+03], CASS [FaCl04], SOCAM [GPZ04], and CORTEX [BiCa04] provide support for gathering and processing context data similar to our approach. However, they leave the reaction to context changes to the application or use hard-to-maintain rule-based approaches for dealing with respective changes.

Only a few approaches like inContext [DoDu07] combine workflows with context-awareness as described in this work. Regarding inContext, contextual information plays a central role similar to our approach; inContext strongly focuses on the teamwork domain, while our approach delivers a more generic technology enabling the development of context-aware, adaptive workflows.

**Semantic Web Services Extensions**

Although the following techniques are limited to web services, they allow for a semantic or contextual integration. Many approaches dealing with contextual integration of web services build upon technologies semantically describing the services, mostly the Web Service Modelling Language (WSML) [dLPF06] and the Web Services Modelling Ontology (WSMO) [RKL+05]. The WSMO features four top-level elements:

- **Ontologies** are used to describe all relevant aspects of the domains of discourse in a syntactical and semantical way.
- **Web Services** provide the concrete functionalities described abstractly in the ontologies.
- **Goals** describe the outcome or functionalities the human desires.
- **Mediators** are utilized for managing different aspects of interoperability like incompatibilities of data.

The WSMO provides an approach for semantically and thus also contextually extending web services technology. It is however limited to web services.

**Context-based Service Selection**

In [DGD07] an approach was proposed that seeks to enable a paradigm shift from the manual selection of services at design-time to the automatically aided selection of those at run-time. Technically, these services are realized by semantic web services. The approach covers both business process management and learning management. For services and goal description, WSMO is utilized and for business process management the Upper Process Ontology (UPO) that was developed as part of the SUPER project, was used. For learning management, a novel ontology, the Learning Process Modeling Ontology (LPMO), is developed. To enable goal-oriented service selection, multiple levels of abstraction are introduced:

- **Data Layer**: This is the lowest level containing the data the services use and provide.
- **Web Service Layer**: On this level the data of the data layer is used for the functionalities the services provide.
- **Semantic Web Service Layer**: This layer enables an abstraction from the functionalities of the services and thus, their semantic selection, invocation, and composition.
- **Semantic Process Model Layer**: This layer contains semantic descriptions of the processes.
- **Semantic Process Domain Model Layer**: This layer contains semantic descriptions of the processes according to their domain mapped to the semantic process model layer and thus makes the lower levels available to different domains.

Utilizing the described concepts and ontologies, a system can select appropriate services to achieve certain goals that are selected by humans matching their current situations.
automatic selection of services, this approach is not directly comparable to the approach developed in this work.

Compliance

Nowadays, enterprises are often required to implement internal control mechanisms in the context of regulatory compliance rules, such as the Sarbanes Oxley Act 2002 (SOX). These include, for example, achievements in efficiency and effectiveness of operations, compliance with laws or reliability of financial reporting. The approach presented in [NaSt07] aims at introducing an abstraction layer above business processes to enable abstract modeling and evaluation of these to enable automatic checking of the processes and application of formal verification techniques. Therefore, a connection of business process management and internal controls management is established: Significant accounts within the enterprise are identified as well as their relation to business processes. For each process, control objectives are defined and risks assessed to implement a set of controls on them. The approach features a three-phased procedure. First, a representation of the process is stored in the semantic mirror, which holds an ontological representation of all controlled processes. After that, control statements are defined for the automatic evaluation of the application controls. Finally, the processes are executed. To enable control, a bidirectional connection between the processes and the controls management is established, informing the controls management about the state of the process instances. In case of violations, recovery actions can be executed.

Two other approaches in this context are SeaFlows [LRKD11, LRD10a, Ly13] and C3Pro [KRFR13, KRL+13, KRM+12, FIRR15]. SeaFlows provides a general framework for supporting business process compliance along the complete process lifecycle. It includes a graphical modeling language to capture process-related compliance rules and execution mechanisms for checking process models and running process instances against modeled rule graphs. The results of such checks can be assessed and aggregated by provided compliance notions. Furthermore, SeaFlows enables a broad application by a general trace model as a formal foundation for the formal and operational semantics of the rule graphs. C3Pro uses methods for compliance checking of processes and extends them to be applicable for collaborative cross-organizational processes. In this context, not only consistency checks are provided, but also change propagation mechanisms between business partners.

These approaches provide a set of governance-related contextual extensions to processes. However, these extensions do not relate to the automated enactment support of contextually adaptive SE processes.

Process Interoperability

Process interoperability can involve different aspects. On one hand, it can deal with the abstract process captured within a process model because, for example, different organizations may specify processes in different ways that may not semantically match. On the other, interoperability can deal with the technical realization of the processes, thus aiming at unifying different process languages semantically. The approach described in the following deals with the former of the two mentioned aspects and concerns processes specified as Petri nets. Petri nets [ReRo98] have been used to describe processes technically. Extensions of Petri nets enabling contextual integration of these have been described as well. The approach described in [KoOb05] provides semantic annotations for so called Pr/T nets (predicate-transition nets [GeLa81]) that utilize OWL-DL classes and taxonomies. The ontology categorizes the Petri net elements into nodes and arcs. Nodes comprise the places and transitions of the Petri net. Since they have different meanings, arcs are separated into 'fromPlace' arcs that are directed from a place to a transition and 'toPlace' arcs that are directed from a transition to a place. To represent different types of markings for the places, concepts 'Number', 'Indistinguishable', and 'IndividualDataItem' are used.

This ontology is further used to enable semantic alignment of processes [BEK+06]. Therefore, a background ontology is added that is modeled using UML and then translated into OWL. The goal of
the alignment is to find semantically matching entities in two or more process models, which are then aligned. The procedure takes the ontological descriptions of the processes as input and compares each element of one process with all elements of the second process. Afterwards, similarities are counted and, if exceeding a threshold, interpreted as aligned.

The work presented in [LiSt05, LSH+06] focuses on the second aforementioned aspect of process interoperability, providing a means to annotate process models of different languages semantically to make them interoperable and their semantics machine-readable in order to support knowledge transfer across enterprises. The proposed semantic annotation is separated into three aspects. The model profile annotation describes context semantics of a process model, as e.g., the problem domain, name, date of creation. The model content annotation describes the domain related content of a process, meaning the objects that are participating in the process. The meta model annotation abstracts the process model constructs semantically to a set of morphemes. To commonly describe different process models, the General Process Ontology (GPO) is introduced. It incorporates descriptions of standard process modeling concepts: activities, agents, artifacts, workflow patterns, inputs and outputs, preconditions and post conditions, exceptions, and goals. As example consider the sequence workflow pattern that is described with an id, a name, a reference to the element in the original process model, an alternative name, and predecessor and successor activities.

As in our approach, the described approaches utilize process annotations. However, they are targeted at the interoperability of different modeling languages and do not cover contextual enactment support as our approach does.

**Enterprise Integration**

The approach presented in [GAF00] aims for better integration of the process into enterprise engineering by using ontologies. To enable this, a computational representation of the enterprise (enterprise model) is necessary. Therefore, the work uses the ontology framework that was developed as part of the Toronto Virtual Enterprise (TOVE) project [Fox92] as well as the enterprise ontology developed by the University of Edinburgh [UKMZ98]. The TOVE project provides a set of generic ontologies that cover different aspects of the enterprise and provide extensions to cover cost and quality. Examples include an activity ontology, a product ontology, and an organization ontology.

The enterprise ontology features different levels to cover various aspects of the enterprise including the meta, organization, or strategy levels. The ontologies are developed in the Knowledge Interchange Format (KIF) [GeFi92]. The presented work uses constraints to characterize process integration within the enterprise and introduces new concepts as an initial foundation for a business process ontology. Thereby, automated workflow support is not the intended goal, but the high-level representation of business processes in the enterprise. It focuses on processes that relate to customers and defines a set of process concepts, including customer interaction process, information flow, or evaluation processes.

The approach just presented uses ontologies as does the approach presented in this work. However, the ontologies above are not used to extend process modeling for enactment, but rather for better integration into an enterprise.

**Business-IT Alignment**

Business-IT alignment concerning processes has been the goal of numerous works. A prominent example is the European project SUPER (Semantics Utilised for Process management within and between Enterprises) [SUPE09] that unites many approaches. In particular, it aims to semantically annotate business processes to make them machine readable, and primarily to unite the different views on processes, like the business view and the technical view. The notion of SBPM (Semantic Business Process Management) [HLD+05] was created within the SUPER project. SBPM includes different ontologies and semantic annotations for different process languages like Event-driven Process Chains
Due to the size of the project, many aspects of SBPM were considered and researched by many different researchers in different countries. Although the active enactment and adaptation of semantic processes was not the focus of SUPER, it does have similarities to our work. In the following, we will give an overview about SUPER and SBPM, discussing selected publications from the project covering different areas.

According to SUPER, BPM lacks automated support and therefore no smooth transition between the business world and the IT world is possible. On major factor hindering that is complicated business process design. Business process issues are addressed rather specifically on different levels of abstraction [PDB+08], including the business level (strategic), the business level (operational), the technical level (processes), the technical level (services), and the technical level (implementation). To enable a holistic view on the different layers of the business process, SUPER offers a stack of ontologies for various purposes that build upon the WSMO.

An Upper Level Process Ontology (UPO) provides the integration of the different conceptualizations and contains high-level concepts that are also used in the other ontologies as, such Process or Activity. Organizational Ontologies capture high-level business aspects such as the organizational structure, resources, or business functions. A Business Process Modeling Ontology (BPMO) resides on a more concrete level of abstraction and unifies specific sEPC [ThFe06a, ThFe06b] and the sBPMN [AFKK07] ontologies, which capture the semantic enhancements to EPCs and BPMN. To support the transformation to executable workflows, the BPMO is also connected to the sBPEL ontology, capturing the semantic enhancements to WS-BPEL. These enhancements are created using the formalisms of WSML [NWV07].

To also capture log information and semantically link it to the business to enable Business Process Analysis (BPA), the Core Business Process Analysis Ontology (which is also called Core Ontology for Business Process Analysis, COBRA [PDA08]) is also connected to the UPO. BPA seeks to provide answers to questions about processes, activities, and resources. The SUPER project seeks to cover all phases of the business process lifecycle. The first phase is the modeling phase, in which analysts create a first process model. Then, the process model is translated and enriched into a model executable by workflow engines in the implementation phase, which thereon executes them in the execution phase. The analysis phase deals with process monitoring and mining. In [WMF+07], requirements are elicited for semantic business process modeling, semantic business process implementation, semantic business process execution, and semantic business process analysis.

The architecture to satisfy the various requirements of SBPM is described in [KVL+08]. A SBPM system is built around a Semantic Service Bus (SSB) that takes the role of an integration middleware. The process modeling environment comprises a modeling tool, a composer, a discoverer, and a SBP repository. The SBP repository [MWA+07] supports semantic querying for SBPs, check-in and check-out operations, and fine grained locking possibilities. The execution environment supports goal based execution of services using WSMO goals and BPEL extensions that support the execution of SWS. The execution history component enables monitoring and analysis, storing all events in a globally shared persistent storage.

The project SUPER provides a rich set of ontologies aiming at contextually extending many aspects of processes applied in companies. However, its primary focus is bridging the gap between business and IT views on processes. It does have aspects covering execution of these processes, but this is limited to a semantic integration of running web services into the system of ontologies. SUPER does neither achieve nor aim at active contextual support of workflows to aid the executing humans as our work does.
Enterprise Collaboration

There exist various approaches dealing with enterprise collaboration. An example is the FUSION project [FUS15] that aims at supporting SMEs in collaboration with international partners via the semantic fusion of heterogeneous business applications. As in SUPER, FUSION also has a set of similarities with our work while not having the same goal. We will highlight these in the following.

In [BGM07], an ontology for enterprise application integration (ENIO) was presented to overcome the lack of formal semantics that impedes easy connection of heterogeneous applications. ENIO is based on the DOLCE [MBG+02] and SUMO [PNL02] ontologies and extends them. It was developed with four primary goals. First, inputs and outputs of services should have a formal definition to resolve message level heterogeneities. Second, the functionalities of service operators should also be formally represented to enable effective discovery of those. Third, reusable process templates should be provided to facilitate manual process composition. Fourth, semi-automated process composition should also be supported. The ontology is utilized to enable semantic assistance in collaborative process design as described in [ABB+07]. Therein, two types of process design are described. The first is the semantically-assisted manual process design where most tasks are executed manually by the modeler. The second type of process model creation is semi-automatic and supported by a composition goal that generates executable processes out of a list of component web services and a composition goal.

Another interesting approach for the composition of business processes that emerged out of the FUSION project is described in [LeFr07]. The composition problem is solved via abstract state machines (ASMs) based on a mathematical model. A web service is defined by input and output variables, a set of states, and a transition function, which is restricted to a bipartite directed tree. The states of the web service are covered by a taxonomy. For communication of the web services, globally unique variables are defined that are organized as tuples combining the output variable of one web service and the input variable of another. The orchestration of the web services is encapsulated in an alternating invocation of the different ASMs. The correctness of the composition is defined using a composition goal that is described using primary goals and recovery goals. Since the execution relies on potential nondeterministic web service behavior, a verifying procedure ensures that there exists a successful composition also for non-deterministically deviated paths.

The FUSION project proposes ontological models to extend various tasks and systems utilized in companies. However, the focus lies more on describing the systems and enabling better integration of the companies, and not in better workflow enactment support as in this work.

7.8.4. Related Work Summary

To the best of our knowledge, no approach exists that has the same goals and set of features as the approach presented in this chapter. The related work can be grouped into two areas: enhancement of processes regarding dynamic enactment aspects and contextual integration aspects.

Regarding the dynamicity aspects of process enactment, different areas of related work have been discussed: configurable processes, artifact-centric processes, dynamic processes, and automated process adaptations. As shown in this section, none of them has a feature set comparable to our approach. The area most closely related is automated process adaptations. However, the approaches in this area focus rather narrowly on the aspect of exception handling and do not enable process-based comprehensive project support.

The other aspect discussed in this section is the contextual integration of processes. Therefore, several approaches have been discussed. The SUPER project pursues a holistic approach for semantically enhancing process management. It offers solutions for all areas of the process lifecycle from modeling to analysis of the processes. Its primary focus is on bridging the gap between the business and the IT world, utilizing the semantic information to enable the SBPM system to provide a view on the process that comprises business and organizational facts as well as all information needed for execution.
However, it does not exploit the machine-readable information to enable the system to automatically influence the execution of the processes to, e.g., match changing situations. Further, it is relatively heavyweight and implies a huge infrastructure and much additional effort. This infrastructure may not be suitable for smaller projects or SMEs. The FUSION project, in turn, uses semantic enhancements primarily to support collaboration of different companies. Its focus is not on the comprehensive support of a project process including enactment aspects. The other approaches discussed in this section propose different aspects of contextual process integration used for compliance, enterprise integration, or interoperability. However, none of them considers enhancements supporting and enriching the enactment of processes, and in particular SE processes.

7.9. Summary

Process management has been proven to be beneficial in various areas [MHHR06, MRH08, LeRe07]. Nonetheless, process management and, in particular, process automation faces adoption challenges in SE. This has various causes: The SE process is a dynamic process and SE is a relatively immature discipline that mainly involves the development of new products. Thus, process execution largely depends on humans and on various contextual factors. These factors have a greater impact on the actual execution of the process and deviations from a defined plan. Broadly accepted standard WfMS technology is, however, inadequate to accommodate these requirements. Workflows are modeled rigidly and have to ignore vast parts of the actual SE process (e.g., specialized human activity information). Furthermore, holistic automation attempts are impeded by WfMS because they are not well integrated with other information sources. Thus, SE processes are not very well integrated into SE projects and are manually implemented to a great extent.

This chapter presented the basic SE workflow extension approach that enables process management to incorporate several factors. Automated support is one of them. Using the contextual extensions, the context management component is able to better access the specifications of both workflow templates and instances. The approach developed is flexible concerning connections between different workflows: On one hand, an arbitrary number of sub-workflows are supported. On the other, these connections are unburdened from rigidly integrating the sub-workflows of traditional WfMS. Instead, more flexible connections to activities are supported.

Another advantage of this approach is its extension of workflows with new properties and data to enable better SE context integration as well as the ability to better model the properties of the SE process. Acquired contextual data can be directly used in the process automatically since the specification supports this. Concerning SE process models as well as actual SE process implementation, the extended human activity modeling is of value. SE activities can be connected on multiple levels, have various different properties, and can be grouped as the SE process requires it. Finally, the CPM framework also allows for human-centric decision modeling in the workflows, abstracting from the internal workflow logic and relieving the human from cumbersome decisions.

The developed approach enables better process flexibility by supporting changes even to running workflow instances. Based on the capabilities of the integrated dynamic WfMS, these change options are enabled. Yet they are extended by the contextual extension of the process: By this, the framework is able to apply situational adaptations automatically. Thus, the process can react to real process situations and avoid a ‘plan-reality-divide’.

Finally, by incorporating all these possibilities via the extended process specification, it is possible to model SE processes to a far larger extent versus a standard WfMS. Thus, abstract areas can be integrated as well as operational ones, and they can be connected. This contributes to an approach for project execution in which the abstract processes finally reach the executing humans [Wall07].
8. Extended Software Engineering Process Coverage

This chapter describes those aspects of the CPM framework that not only cover workflows being part of SE process models, but also consider unplanned workflows executed out of scope of the latter. We denote these workflows as **extrinsic** (cf. Chapter 5). This part of the CPM framework targets at covering a greater amount of the SE process by also integrating these extrinsic workflows. Therefore, we call it extended process coverage as introduced in Chapter 4 (cf. Requirement R:ProcCoverage).

Business process management (BPM) and automated guidance of human-centric processes have proven to be useful in a multitude of domains [MHHR06, MRH08, LeRe07, WRMR11]. However, existing BPM technology is often based on rigid process models turning its application in dynamic and evolving domains with diverse workflows such as software engineering (SE) [JaCo93] into a challenging task. In fact, reality often deviates from pre-defined processes [McCo01, CNGM95] in that domain. While automated workflows could assist overburdened software engineers by providing direct orientation and activity guidance, the latter must coincide with the actual situation or it will be ignored causing the entire CPM framework to be mistrusted or ignored. To further adopt automated workflow guidance in SE environments (SEEs), adaptation and pertinence to the dynamic and diverse SE situations is crucial.

SE process models aim to support development efficiency [GGK06]. However, they do not cover all activities executed in an SE project. In particular, humans execute various workflows apart from the models and without support or guidance. We denote such workflows as extrinsic (cf. Chapter 5). Extrinsic workflows cover issues frequently recurring in SE projects. Examples include bug fixing and refactoring. Thus, they are often neither explicitly governed nor supported (cf. Example 8-1). We denote such extrinsic workflows as issue workflows in this chapter. Typically, they are not as foreseeable as intrinsic workflows, which constitute an explicit part of the SE process models. Thus, extrinsic workflows may interfere with intrinsic ones. Furthermore, their enactment often depends on different project parameters like time constraints or risks. In turn, this makes traditional workflow modeling for these SE issues difficult since numerous workflow fragments matching different situations would have to be integrated in one big workflow template.

The resulting workflow problems for SE environments are as follows: First, the high effort required for modeling extrinsic workflows results in the absence of both extrinsic workflow models and, subsequently, automated guidance for these types of workflows, yet these special use cases are often the ones where guidance is especially helpful and desirable. Second, pre-specified workflow templates are limited in their adaptability; thus, such workflows become situationally irrelevant and are therefore ignored [RRD09]. Third, the complex modeling of situational property influences (e.g., risk or urgency) on workflows would have to be integrated into them. This incorporates an implicit modeling with many alternative paths that unduly increases their complexity and aggravates maintenance. The cognitive effort required to create and maintain large process models syntactically [WRMR11] can lower the attention towards the incorporated semantic problem-oriented content.

This chapter focuses on integrating facilities for the appropriate modeling and enactment of extrinsic workflows into the CPM framework. In particular, the following features are integrated:

- Problem-oriented modeling of extrinsic workflows facilitating their systematic creation;

---

1 This chapter is partially based on the publications [GOR10a], [GOR11b], and [GOR12b].

121
• Support for the easy modeling and reuse of both the activities and fragments of extrinsic workflows;
• Automatic detection of situational properties influencing extrinsic workflows;
• Automatic selection of activities matching the current situation;
• Automated generation of workflows using the selected activities.

The remainder of this chapter is organized as follows: Section 8.1 elicits the requirements for the approach enabling extrinsic workflows. In turn, Section 8.2 describes this approach. Related work is discussed in Section 8.3, followed by a summary in Section 8.4.

8.1. Requirements

This section presents detailed requirements that need to be met to enable extended SE process coverage support. In particular, it details the already discussed requirement \( R_{:ProcCoverageage} \) (cf. Chapter 4). These requirements have been elicited based on practical experiences supported by a literature study. The following example illustrates practical problems that led to the definition of the requirements:

Example 8-1 (Extrinsic workflows):
The Company uses a standard process model to capture development activities (cf. Example 4-2). However, this model does not cover many of the daily activities like bug fixing, refactoring, technology swapping, or infrastructural issues. As these activities lack proper process support and remain untraced, there have been efforts in The Company to create workflows for supporting them. However, since there are many different kinds of issues with ambiguous and subjective delineation, it is difficult and burdensome to universally and correctly model them in advance. Many activities may appear in multiple issues, but are not necessarily required, bloating different SE issue workflows with many conditional activities if pre-modeled. Figure 8-1 shows an extrinsic workflow for bug fixing. It contains about 30 activities of which many are executed conditionally in order to accomplish various tasks like testing or documentation. An example considers activities for static source code analysis that may be omitted for urgent use cases. Furthermore, there are various reviewing activities with different parameters (e.g., effectiveness or efficiency), where the choice can be based on specific project parameters (e.g., risk or urgency). The same applies to different testing activities. Moreover, it has to be determined whether a bug fix shall be merged into other branches of the source control system applied.

As the decisions in the workflow mainly rely on contextual properties, many activities could be already excluded prior to enactment, as each situation requires another workflow marking a subset of the workflow shown in Figure 8-1. However, the contextual information for making such decisions is not always in place, and gathering it would mean additional efforts for the humans. Another option, modeling many smaller workflows for different situations is also problematic, as the best fitting workflow for each situation would have to be determined manually. Additionally, that solution would result in a high number of modeled workflows making the choice among them difficult. Finally, many of the activities and even entire fragments of the workflows would appear in multiple workflows, resulting in redundant modeling.
To counteract problems as described in Example 8-1, we elicit a set of requirements for supporting activities and workflows being extrinsic to SE process models as extension of requirement R: R:ProcCoverageage (cf. Chapter 4). These requirements are specific and assume that basic requirements concerning automated process support (cf. R:AutoProc) and context awareness (cf. R:ContInt) have already been satisfied (cf. Chapters 4 and 7).

To enable comprehensive SE process support, a framework for SE process enactment should cover the activities actually performed as precisely as possible. Particularly, this includes extrinsic activities, which are mostly unaddressed by standard SE process models.

- **Requirement R:ProcCoverageage:CovInEx (Intrinsic / extrinsic support):** There should be a facility to support both intrinsic and extrinsic activities with an SE framework.
• Requirement R:ProcCoverage:CovU (Uniform workflow realization): Both intrinsic and extrinsic activities should be executable in a uniform way to support assistance for the human as well as to enable easy tracking and analysis of executed workflows.

To not only support humans in enacting the workflows, but also in creating them, an intuitive way of modeling needs to be provided that accommodates the properties of the extrinsic workflows as well. Our practical experiences with workflows like the one depicted in Example 8-1 have shown that imperative modeling styles as offered by common WfMS, do not suit such situations. On one hand, they differ in their properties (e.g., dynamicity and context-dependency). On the other, they do not represent pre-planned processes to be rigidly followed. Rather, they represent best practices often discovered or refined during everyday work. Thus, they are not pre-planned by process engineers but often captured on the fly by the humans enacting them. Having to model complicated structures like shown in Figure 8-1 will be an obstacle in this dynamic process and even deter humans completely from it.

• Requirement R:ProcCoverage:ModDy (dynamic modeling): Compared to intrinsic workflows, extrinsic workflows are more dynamic and less predictable. Hence, their modeling should enable the coverage of various situations, but without bloating process models or making them too complex.

• Requirement R:ProcCoverage:ModRe (modeling for reuse): Workflow modeling itself should remain easy and foster the reuse of modeled workflows or the parts thereof.

• Requirement R:ProcCoverage:ModHi (hide complexity): Workflow modeling should hide the inherent complexity of the workflow templates to assist the human with their creation. The human shall be able to concentrate on the content-related problems rather than modeling complexity. We call this problem-oriented modeling. Simplicity is crucial for modeling extrinsic workflows to not deter humans from doing it.

To be able to generate workflows matching various situations, a method for modeling contextual influences and connecting them to the workflow templates is required. Facilities to gather contextual information also become necessary.

• Requirement R:ProcCoverage:CtxGet (Gather contextual information): It should be possible to automatically gather situational information from humans or the environment.

• Requirement R:ProcCoverage:CtxInf (Model contextual influences): It should be possible to model contextual influences on the workflows to be able to use situational information directly and automatically.

• Requirement R:ProcCoverage:CtxCon (Connect workflow and context): It should be possible to connect workflow activities with contextual properties to enable their automated selection in alignment with the situation.

The elicited requirements are summarized in Table 8-1:

<table>
<thead>
<tr>
<th>Req ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R:ProcCoverage</td>
<td>Process coverage</td>
<td>The CPM framework shall cover as many activities and workflows as possible.</td>
</tr>
<tr>
<td>R:ProcCoverage:CovEx</td>
<td>Intrinsic / extrinsic support</td>
<td>The CPM framework shall support extrinsic as well as intrinsic workflows.</td>
</tr>
<tr>
<td>R:ProcCoverage:CovU</td>
<td>Uniform workflow realization</td>
<td>Intrinsic and extrinsic workflows shall be realized in a uniform way.</td>
</tr>
<tr>
<td>R:ProcCoverage:ModDy</td>
<td>Dynamic modeling</td>
<td>The CPM framework shall enable the dynamic modeling of extrinsic workflows.</td>
</tr>
<tr>
<td>R:ProcCoverage:ModRe</td>
<td>Modeling for reuse</td>
<td>The CPM framework shall enable the modeling that fosters reuse.</td>
</tr>
<tr>
<td>R:ProcCoverage:ModHi</td>
<td>Modeling hiding complexity</td>
<td>The CPM framework should hide complexity from humans.</td>
</tr>
</tbody>
</table>
8.2. Hybrid Workflow Approach

This section discusses the different parts of the approach for modeling and enacting extrinsic SE workflows. This incorporates a discussion of different types of activities in an SE project. Following this, the approach to modeling and enacting extrinsic activities is presented as well as the approach for exploiting context data for extrinsic workflows.

8.2.1. Different Activity Types of Software Engineering Workflows

Extrinsic workflows differ from intrinsic ones. On one hand, they are extraneous to the SE process. Thus, they cannot be modeled as part of the SE process. Furthermore, they are difficult to trace as their flow is neither predictable nor transparent. While certain extrinsic workflows may be automatically or semi-automatically initiated, others rely on manual activation by humans. On the other hand, their internal governance is more difficult. The concrete set of activities may largely depend on situational properties like deadlines or quality goals. Therefore, the imperative way of modeling as favored by traditional process management may not always be suitable.

In the following we present a separation of different workflows occurring in SE projects using three different dimensions and three concrete use cases: The first is their affiliation to the SE process (process affiliation), i.e., intrinsic workflows are tightly integrated into the SE process while the extrinsic ones are enacted without explicit relations to it. The second one is the type of workflow modeling. We assume that for less predictive extrinsic workflows, the imperative way of modeling is less suitable. The third concerns the automation level of their initiation (i.e., automatic vs. manual). Figure 8-2 illustrates this by concrete use cases the CPM framework will enable, situated in a three-dimensional space: The x-axis denotes the process affiliation, the y-axis illustrates the type of modeling, and the z-axis depicts the automation level for the triggering of workflow enactment.

![Workflow modeling dimensions](image)

**Figure 8-2: Workflow modeling dimensions.**
The first use case (standard SE process enactment) deals with standard process enactment. This implies intrinsic workflows of the SE process whose activity sequencing is known a priori (i.e., imperative modeling is possible). By contrast, issues occurring during SE projects (issue processing) are ad-hoc, do not belong to the process, and are dynamic, relying on situational properties. Such issues frequently occur during SE projects. One of them that has been inspired by our practical experiences is illustrated by Example 8-2.

Example 8-2 (SE issue):
A requirements’ analyst prepares a special build of the software produced for a customer demonstration. He notices that some crucial function does not work in that build and, because of time pressure, directly contacts a developer about this issue. The developer immediately starts working on the issue and, within an hour, delivers a fix directly to the analyst, enabling him to hold a successful customer presentation.

Another use case (automatic follow-up activities) is illustrated by follow-up activities, which are extrinsic, but might be required due to the enactment of an intrinsic activity. For example, if a developer changes code belonging to an interface component, not only unit tests may have to be adapted, but these changes shall be reflected in the architecture specification and the integration tests as well. However, these activities may have to be processed by other humans in other teams, like architects or the test team. This chapter deals primarily with issue processing while follow-up activities will be covered in Chapter 10.

8.2.2. Extrinsic Workflow Modeling and Enactment

This section elaborates on how we aim at fulfilling the requirements just elicited for extrinsic workflows. Roughly, this concerns three main areas (cf. Section 8.1): First, the modeling, which should be as simple as possible, while at the same time allow modeling specifics of extrinsic workflows. Second, facilities must be in place allowing the CPM framework to automatically select the exact set of needed activities is not known prior to workflow enactment. To achieve this, workflows consist of activities and constraints. Upon enactment of such a workflow, every activity can be executed as long as no constraint is violated. Figure 8-3 shows an example of a declarative workflow model with the tool DECLARE [PSSA07, PSA07, Pesi08, HBZ+14]. It contains three different constraints. For example, ‘precedence’ imposes that, if activity ‘Run Developer Test’ is executed, activity ‘Implements Developer Test’ must have been executed before. Another example is ‘chain response’ requiring that after activity ‘Design Solution’ the next executed activity is always ‘Implement Developer Test’.

Declarative approaches allow for the simple modeling of small activity sets for workflows that are less foreseeable and structured than imperative ones. However, when involving a greater number of activities and constraints, declarative modeling can be difficult to comprehend [ZPW11a, HBZ+14, ZSH+15] and may produce models that are hard to maintain [ZPW11b].

Our basic idea was to combine elements of declarative and imperative modeling to enable modeling being simple enough to be done during everyday work, but still having enough expressive power to capture all necessary properties of extrinsic workflows. Furthermore, we wanted to make enactment fit into the CPM framework seamlessly to be able to exploit the CPM concepts and features being in
place for extrinsic workflow enactment as well. Figure 8-4 illustrates the different parts of the CPM framework for extrinsic workflows.

Figure 8-3: Declarative modeling with DECLARE [Pesi08]

The CPM framework’s approach for modeling extrinsic workflows (cf. Figure 8-4C) consists of three main parts. First, we integrate constraints that are similar to the ones utilized in declarative modeling (cf. Figure 8-4A). However, our focus on these constraints was not an approach as comprehensive as, for example, DECLARE [PSSA07, PSA07, Pesi08] because, that way, modeling can get complicated and difficult to understand. Our focus was a very simple way of modeling (cf. R:ProcCoverage:ModHi). Therefore, we integrated only basic constraints sacrificing the high expressive power provided by approaches like DECLARE. To overcome this limitation, we included constructs similar to the blocks in block-structured imperative modeling (cf. Figure 8-4B). We denote these blocks as building blocks (cf. Figure 8-4C). On one hand, building blocks enable our approach to capture complex structures. On the other, modeling with them is designed to be hierarchical, separating workflow templates into nestable blocks. These blocks may be modularized and can be treated like simple activities, fostering their reuse in various workflow templates (cf. R:ProcCoverage:ModRe).

Constraints and building blocks are further extended by a third modeling concept: the context properties (cf. Figure 8-4C). The latter enable the modeler to express relations of activities or building blocks to contextual properties (cf. R:ProcCoverage:CtxInf); e.g., a certain review activity may only be chosen if risk is high in that situation. This way, the specific properties of extrinsic workflows can be included into the models (cf. R:ProcCoverage:ModDy) connecting them directly to the context (cf. R:ProcCoverage:CtxCon). However, by connecting activities to properties, the former become candidate activities that only come to execution when specific contextual properties apply. Therefore, we create an approach for automatically selecting matching activities for every situation (cf. Figure 8-4E). This approach utilizes the sensors of the CPM framework (cf. Figure 8-4D and Chapter 7) that allow for the automatic gathering of contextual information (e.g., state transitions of certain SE tools or SE artifacts recognized as situational properties; cf. R:ProcCoverage:CtxGet). In turn, these properties have values that may be derived from various sources, e.g., the skill level of a human executing an activity or the measured code complexity of a source code artifact.
When the set of applicable activities has been selected, an imperative workflow instance is automatically generated out of these activities (cf. Figure 8-4E). This is possible by combining basic constraints with the building blocks since the latter are derived from the constructs of block-structured workflows. Utilizing this modeling method, extrinsic workflows can be modeled with reasonable effort and thus be integrated into the CPM framework (cf. R:ProcCoverage:CovInEx). To unite this with traditional imperative process modeling, which remains useful for more predictable processes [ReWe12], the approach unites both ways of modeling under a common process management concept (cf. R:ProcCoverage:CovU). The following sections provide details and introduce the different parts of the concept: contextual extensions to SE process models, modeling of contextual influences, gathering of contextual information, and modeling SE processes in a declarative way.

8.2.3. Applying Situational Method Engineering

In order to incorporate contextual influences into the extrinsic workflows we integrate a methodology called situational method engineering. The latter adapts generic methods to the actual situation of a project [RBH07]. This is done based on two different influence factors, i.e., process properties, which capture the impact of the current situation, and product properties that realize the impact of the product currently being processed. In this context, the product relates to the type of component, e.g., a GUI or database component being developed. To strike a balance between rigidly pre-specified workflows and the absence of process guidance, we apply a basic workflow for each SE issue, which is then dynamically extended with activities matching the current situation. The SE issues correspond to the term case used in situational method engineering. That way, an occurring issue will be handled by such a case. The CPM framework incorporates a case base as well as a method repository for the
construction of the workflows. The case base contains a workflow skeleton for each of the SE issues. The workflow skeleton belonging to a case only contains the fundamental activities for that case. The method repository contains all other activities whose enactment is possible according to the case. To be able to choose the appropriate activities for the current artifact and situation, the activities are connected to properties that realize product and process properties of situational method engineering.

Each SE issue, such as refactoring or bug fixing, is mapped to exactly one case relating to exactly one workflow skeleton. To realize a pre-selection of activities (e.g., Create Branch or Code Review), which semantically match a case, the case concept is connected to an activity concept via an n-to-m relation. Activities are connected to properties used to model a particular situation. Therefore, the properties have a concrete value representing how well a property applies in a situation (e.g., urgency ‘++’, complexity ‘-‘). The selection of an activity may depend on various process as well as product properties. To model the characteristic of a case leading to the selection of concrete activities, the case is connected to various properties as well. The latter have a computed value indicating the degree in which they apply to the current situation. Utilizing the connection of activity and property, selection rules for activities based on the values of the properties can be specified. Example 8-3 illustrates these concepts by means of a simplified bug fixing workflow.

Example 8-3 (Situational workflow extension):
Figure 8-5 shows different parts of the concept for extrinsic workflows applied to a bug fixing issue. On the left side, the related case and skeleton workflow are shown. This skeleton workflow is then extended with activities that match the values of the properties: Activity ‘B’ (e.g., ‘Run Regression Tests’) is added due to property ‘Criticality’ and activity ‘C’ (e.g., ‘Validation to Requirements’) is added due to property ‘Complexity’.

8.2.4. Information Gathering
To leverage automatic support for extrinsic workflows, computing the values of the properties constitutes a key factor. The approach unifies process and product properties in the property, which can be influenced by various factors. On one hand, tool integration can provide meaningful information about the artifact being processed in the current case. For example, if the artifact is a source code file, static code analysis tools (e.g., PMD [Cope05]) can be used to execute various measurements on that file, revealing potential problems. This part of the concept may be executed automatically without human involvement. Therefore, we call it implicit information gathering. Since neither all aspects of a case are covered by implicit information nor all options for gaining knowledge about the case are always present, the CPM framework utilizes explicit information gathering from the human processing the case. To enable and encourage the human to provide meaningful information, we apply a simple response mechanism that can be integrated into the user interface of a CPM framework. With this mechanism, the human may directly influence process as well as product properties. To enable the CPM framework to utilize explicitly gathered information for workflow generation, the workflow skeletons of the cases always start with an activity ‘Analyze Issue’. The latter prompts the human to gain awareness about the issue and the current situation, and lets him set the properties accordingly. To keep the number of adjustable parameters small and to not burden the
human, we introduce *product category*. The latter unites the product properties in a pre-specified way. The influence of the product categories on the different properties is specified at build time and can be adapted to fit various projects. Example 8-4 illustrates both implicit and explicit information gathering.

**Example 8-4 (Information gathering):**

Figure 8-6 shows a simplified excerpt from a bug fixing issue. It shows a related workflow with different activities, like the application of the bug fix, the explicit checking of dependencies to determine the impact the bug fix might have, a GUI testing activity to determine whether user functions still work, and two review activities with different effectiveness and time consumption. A sensor provides information on the source code from a static analysis tool (e.g., PMD) and a junior engineer that processes the issue. If, for example, a high coupling factor in the source code is detected by the sensor (implicit information gathering), this will raise product property ‘risk’ associated with that file. The integration of various project areas like resource planning entails contextual knowledge about the entire SE process. An example is elevating process property ‘risk’ if the human processing the current case is a junior engineer. As aforementioned, product and process properties are unified as simple properties that both influence the selection of activities similarly. The human can also directly provide information to the CPM framework (explicit information gathering), for example, by specifying a product category for the files he will process, e.g., a database or GUI component. The database component is likely to have more dependencies, whereas the GUI component presumably has more direct user impact. Having concrete values for the properties in place, activities can be automatically selected to generate a workflow fitting the current situation.

![Diagram](image_url)

**Figure 8-6: Information gathering example**

### 8.2.5. Declarative Workflow Modeling

After computing the property values, activities must be selected and correctly sequenced to enable dynamic generation of the SE issue workflow. For this purpose, the connection between properties and activities is utilized. In general, an activity may depend on a number of properties. Examples include selection rules such as:

- Choose activity ‘*code inspection*’ if risk is very high, criticality is high, and urgency is low.
- Choose activity ‘*code review*’ if risk and criticality are both high.

Declarative workflow modeling approaches incorporate a certain degree of flexibility in the workflow templates [PWZ+11], which may be applied to a variety of situations. Since the declarative way of modeling might be difficult to understand [ZPW11a, HBZ+14, ZSH+15] and produce hard to maintain models [ZPW11b], we introduce several simplifications: As stated in Section 8.2.2, the declarative
Workflow modeling approach chosen in this thesis is based on very simple constraints utilizing building blocks that allow for the further structuring of workflows and structural nesting of the contained elements.

The CPM declarative workflow modeling approach is illustrated by and compared to classical workflow modeling in Figure 8-7. The latter shows the modeling of both imperative and declarative work unit containers in the upper section and the derived workflow instances for enactment in the lower one. ‘Work unit container 1’ shows a simple, imperatively modeled workflow and its counterpart enacted in the WfMS (‘Workflow 1’). ‘Work unit container 2’, in turn, illustrates declarative modeling of the same workflow. Here, the exact structure of the workflow is not rigidly pre-specified. Further, there are only simple constraints connecting activities in the workflow. Examples include ‘Requires’, expressing that one activity requires the presence of another one, and ‘Parallel’, expressing that two activities shall be executed in parallel. The workflow instance generated for these constraints looks exactly like the imperatively modeled ‘work unit container 1’. Activities in the declarative approach further have relations to contextual properties in order to enable the CPM framework to select a subset of the pre-specified activities for the workflow instance generated from it. Finally, ‘work unit container 3’ demonstrates the use of building blocks for further structuring the workflow. Three building blocks are shown for sequential, parallel, and repeated enactment of the contained elements (cf. Figure 8-7). ‘Workflow 3’ shows how a workflow instance is built based on constraints and the building blocks. Furthermore, it demonstrates contextual relations, in this case assuming that the contextual properties of the situation led the CPM framework to the selection of activities ‘1’, ‘2’, ‘3’, and ‘5’, while omitting activities ‘4’ and ‘6’.

Building blocks enable the hierarchical structuring of activities contained in workflows and can be reused in different work unit containers easily, where they are treated like simple activities hiding the complexity of the contained activity structure. That way, basic modeling remains quite simple while retaining the ability to model complex structures as well. Examples include activities related to software creation like coding, testing, or documenting. These can be structured by the mentioned building blocks. For example, a loop may be used to enable multiple iterations of coding, documenting and testing new code combined in one building block. The latter may, for example, be called ‘Software Development Loop’ and be easily reused as a single activity. In conjunction with the simple basic constraints, this allows for simple and understandable workflow templates. Finally, the advantages of imperative and declarative modeling approaches are united: The imperative workflow instances generated for enactment ensure that humans follow the predefined procedures and also aid the humans with workflow guidance. However, by declaratively specifying various candidate activities for these workflows and connecting them to situational properties, the CPM framework retains the ability to choose the appropriate activities for the humans’ respective situation.

Figure 8-7: Declarative workflow modeling.
In the following, we describe the constraints and building blocks integrated into the CPM framework, as well as conditions to be met for declarative modeling such that they may later be verified automatically.

**Basic Constraints**

As stated in Section 8.2.2, it is crucial for extrinsic workflows to be simple to model, otherwise humans will be deterred from it. Therefore, the kind of constraints supported were kept simple. In particular, they facilitate workflow modeling sacrificing a high expressive power. More complex structures shall be expressible using building blocks; i.e., we want to create a minimal set of constraints, omitting more complex constraints as known from languages like DECLARE [PSA07]. Note that complexity might substantially impact understandability [PWZ+11]. More precisely, the constraints are categorized into **sequencing constraints** and **existence constraints**. The latter govern which activities shall be present in a workflow instance, while sequencing constraints govern how the activities shall be arranged. The available constraints are defined in Table 8-2:

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Meaning</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>X succeeds Y</td>
<td>if X and Y are present: X must appear directly before Y.</td>
<td>sequencing</td>
</tr>
<tr>
<td>X isParallelTo Y</td>
<td>if X and Y are present: they must appear parallel; all predecessors of one of the parallel activities must be finished to let them start; all parallel activities must be completed to let successors of one or more of them start.</td>
<td>sequencing</td>
</tr>
<tr>
<td>X requires Y</td>
<td>if X is present, Y must also be present</td>
<td>existence</td>
</tr>
<tr>
<td>X mutuallyExcludes Y</td>
<td>if X is present, the presence of Y is prohibited and vice versa</td>
<td>existence</td>
</tr>
</tbody>
</table>

The constraints shown in Table 8-2 are not to be compared directly to declarative workflow modeling approaches like DECLARE. The number of constraints is rather low and they are not meant to enable the modeling of complex workflows. In fact, they have two main purposes, to make basic extrinsic workflow modeling extremely simple and to foster generation of an imperative workflow out of them. To support the latter, we apply a high cohesiveness to parallel activities as expressed in Table 8-2. In the following, we first show how basic workflow instances are generated out of the constraints. Then, we discuss the building blocks in detail to show how more complex structures can be built. The combination of simple constraints and building blocks has another major advantage for quick on-the-fly modeling of SE issues: once modeled, building blocks are stored in a library and can be used like simple activities. Assuming that such a library is in place, humans can model new SE issues quickly as they occur during their everyday work. They can rely on the simple constraints and the CPM framework hides the complexity contained in the building blocks and the context-based activity selection from them.

The existence constraints can be checked prior to workflow generation to ensure that the set of chosen activities is sound according to the specification. Note that existence constraints always apply to all activities in a container while sequencing constraints are only valid on one level of abstraction; i.e., they are validated for one container or one building block and are not checked recursively for other building blocks that might be contained in them. Existence constraints are checked recursively for all contained building blocks. The sequencing constraints must be transformed to workflow patterns to enable the generation of an imperatively modeled workflow instance. This is shown by four basic examples in Figure 8-8 and explained hereafter.
Figure 8-8: Constraint implementation.

Figure 8-8a shows the ‘succeeds’ constraint, below a visualization of it, and below this how it is implemented as a simple sequence in a workflow. Figure 8-8b deals with the ‘isParallelTo’ constraint. It is implemented using an AND pattern in the workflow. Figure 8-8c shows a simple combination of the two sequencing constraints, ‘all predecessors of one of the parallel activities must be finished to let them start’ and ‘all parallel activities must be completed to let successors of one or more of them start’. Figure 8-8d, including two ‘isParallelTo’ constraints, shows that both of these are implemented independent from each other by specifying that activity ‘2’ must be executed in parallel to activity ‘4’, and activity ‘3’ must be executed in parallel to activity ‘5’. If the two ‘isParallelTo’ constraints were united and implemented with an AND pattern in the workflow, that could not be guaranteed. For example, if activity ‘2’ takes much longer than activity ‘4’, the latter would be executed in parallel to activity ‘3’ as well. This is prohibited by the implementation shown in Figure 8-8.

Building Blocks

The building blocks that enable complex structures have been developed to mirror standard workflow patterns for block-structured workflows [Reic00]. This way of structuring enables easy separation of the workflow into nested blocks. More specifically, these blocks may be activities, patterns, or the workflow itself. Each block must have a unique start and end point [RRKD05, VVK09, KHB00]. The blocks can be regularly nested, meaning that they must not overlap [ReDa98, VVK09, KHB00]. For unstructured workflows, in most cases a transformation to a block structured model can be applied [VVK09, MRv10, KHB00]. For control flow modeling, the basic patterns are sequence, AND, XOR, and LOOP [vtKB03]. Based on these patterns, most models in practice can be covered since they constitute the basis of any process specification language [Mend08, zuRe08, LRW11]. They can be easily transformed to formal languages like Petri Nets [vdAa98] and to other widespread process languages like WS-BPEL [BPEL07, ReRi06]. There are other control flow patterns like the Multi-Choice / OR-split [vtKB03]. However, this work presumes the sole usage of the basic control flow patterns, since the use of other patterns complicates the process model and promote error-proneness [MRv10, Kind06, MNA10]. Furthermore, it is possible to construct other control flow patterns using the basic ones. For example, composing an OR-split with XOR- and AND-splits is possible [Reic00, MDA08]. The available building blocks and their implementation with control flow patterns is shown in Table 8-3.
Table 8-3: Building blocks

<table>
<thead>
<tr>
<th>Building Block</th>
<th>Control Pattern(s)</th>
<th>Flow Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>building block</td>
<td>-</td>
<td>The building block represents an abstract super class for all available building block types.</td>
</tr>
<tr>
<td>activity</td>
<td>-</td>
<td>The activity represents exactly one activity.</td>
</tr>
<tr>
<td>sequence</td>
<td>sequence</td>
<td>The sequence represents a sequence of other building blocks. It is used to enable easy encapsulation and reuse of multiple sequentially executed activities.</td>
</tr>
<tr>
<td>parallel</td>
<td>AND-split, AND-join</td>
<td>The parallel represents the parallel execution of other building blocks.</td>
</tr>
<tr>
<td>loop</td>
<td>loop</td>
<td>The loop represents the repeated execution of other building blocks, allowing for the specification of cyclic structures in a consistent way.</td>
</tr>
<tr>
<td>conditional</td>
<td>XOR-split, XOR-join</td>
<td>The conditional represents the conditional execution of other building blocks.</td>
</tr>
</tbody>
</table>

The **conditional** implies a deferred decision regarding the executed activities being transformed into a XOR pattern. For the decision of the XOR pattern, the value range of the variable used for the decision should be completely covered to avoid deadlocks in execution [MDA08, LRD10b]. This, combined with the fact that building blocks contain candidate activities from which a subset is to be chosen, makes it error-prone. The value range can become only partially covered, and it is possible that two or more activities (from which a selection was intended by the modeler) are omitted due to context properties, leaving no valid choice at run-time. In light of these problems, the alternatives are implemented with user decision modeling as introduced in Chapter 7. That way, even if the whole value range is not covered, the human can only choose options connected to a covered value range. Two options are supported in modeling a ‘conditional’ building block: the first one contains no empty branch. That way run-time choices between mutually exclusive activities independent of context properties (but dependent on run time human decisions) can be modeled. The second variant contains an empty branch. That way it becomes possible to model a deferred decision that incorporates contextual factors including the case that none of the activities comes to execution.

‘Work unit container 3 / Execution Workflow 3’ (cf. Figure 8-7) has demonstrated how nested building blocks are transformed into the control-flow structure of a workflow. Figure 8-9 shows the concrete implementation of each single building block followed by an explanation.

![Figure 8-9: Building block implementation](image-url)

The **loop** is transformed into a LOOP pattern in a workflow as illustrated in Figure 8-9. The **parallel** is implemented by an AND split and the respective join, while the contained parallel activities are all put on a separate branch of the AND pattern. The **sequence** is simply turned into sequentially connected activities. Finally, the **conditional** is transformed into an XOR split and join with the contained building blocks stored as activities in the XOR branches. For the **conditional**, there exist two options, one incorporating an empty branch besides that branches containing activities as shown in Figure 8-9 and one without that branch.
The usage of building blocks not only enables the modeling of workflows containing all basic patterns, but also simplifies modeling since it fosters the reuse of building blocks: in traditional process management, reuse is limited to workflows or activities. In contrast, our declarative modeling approach supports the reuse of fragments of the workflows. These fragments, captured as building blocks, are encapsulated as simple activities, and thus simplify the workflow structure hiding its inherent complexity. Another factor supporting reuse is the relation to context properties: each simple activity and building block may have such context connections. That way, a building block can be used in different workflows for various situations. Example 8-5 illustrates this.

Example 8-5 (Building block template): A building block template for different code review activities can be defined. These are, for example, ‘Peer Review’, ‘Code Review’, ‘Walkthrough’, and ‘Code Inspection’. Utilizing connections to context properties like ‘Urgency’ or ‘Risk’, these activities “know” the situations to which they apply, and the surrounding building block template can thus be easily used for all of these situations without additional effort.

Conceptual Framework

The CPM framework shall be capable of supporting modeling and enactment of extrinsic activities. Furthermore, it shall automatically incorporate and utilize contextual data to automatically choose the right activity subset for every situation. This involves a huge amount of data that must be well organized. Furthermore, various conditions must be implemented to ensure the correct enactment of extrinsic SE workflows. In this section, therefore, we discuss the conceptual framework that enables the modeling and enactment of SE issue workflows in the CPM framework. In particular, this is the concrete realization of the concepts discussed in this chapter in the CPM framework. Figure 8-10 gives an abstract overview of the connections and interplay of the different concepts.

Figure 8-10: Concepts utilized for declarative workflow generation

SE issues are modeled by the case template concept and executed by the case concept. Each case template is connected to a work unit container template storing the information about the concrete workflow template in the WfMS. However, SE issues are modeled declaratively and have no imperative workflow template prior to enactment. However, they do have a workflow skeleton that will be extended based on the declaratively modeled building block templates and the properties of the
situation. Therefore, the work unit container template has a sub-concept: the declarative container template contains building block templates in addition to the work unit templates that capture the activities contained in the workflow skeleton. Similarly, for the concepts introduced in Chapter 7, we have made a distinction between template concepts for modeling the workflows and individual concepts for their concrete executions. The declarative container used for the concrete enactment of an SE issue has building blocks in addition to the work units derived from its super-concept, the work unit container. Thus, it covers the declarative parts from which the workflow instance is generated and the imperative parts generated for enactment.

Building block templates are used to model candidate activities for declarative workflows. In general, such a template has various subclasses. These incorporate the different building block template types as the sequence template or the loop template for modeling. As aforementioned, simple activities and complex building blocks are treated equivalently from the outside; therefore, the activity template is also a sub-concept of the building block template. As the declarative container template is a sub-concept of the work unit container template, it also has a connection to a workflow template in the WfMS. In this case, the latter is used for the workflow skeleton that is the basis for the workflow instance to be enacted. The activities of that workflow are mapped by the work unit templates belonging to the declarative container template. The latter, in turn, has a property that stores the information, at which point the workflow skeleton shall be extended with the activities that have been declaratively modeled with the building block templates.

The case template is connected to one or more property templates, yielding the capability to specify a unique set of properties with a unique relation to the activities contained in the case template. The property templates, in turn, are connected to various other concepts. Thus, it can be defined for each case what factors shall influence the properties. In Figure 8-10, a selection of possible property influences is shown (only with relevant connections and concepts): A skill template is used to model various special skills a human may possess, like, for example, ‘Database Modeling’ for humans that are experienced with data base modeling. The skill level describes the general skill level of a human, like ‘Junior Engineer’. The product category describes the category to which the processed code belongs. The problem type describes a specific type of problem, e.g., with the code, like ‘High complexity’. All of these concepts can be used to model influences on the situational properties of the case. As example, the case is more risky when the code is complex and when a junior engineer processes it.

As shown in Figure 8-10, in addition to the template concepts for modeling SE issues, there are individual concepts used to capture one concrete enactment of an SE issue: A case holds the information for the issue, like the concrete person that processes it. In turn, the human has a concrete skill level and special skills. Furthermore, there are concrete artifacts to be processed, which may have concrete problems. Based on these concepts relating to a concrete situation and by the connections to the templates, concrete values for the properties of the case are computed. By these values, in turn, a set of concrete building blocks is generated for the case. This set is a subset of the building block templates that has been defined for the relating case template. Finally, that set of building blocks is utilized for the generation of a concrete imperative workflow instance that will be used to guide the human in the concrete situation.

The most complicated part of this concept is the specification of the structure for the declarative container templates and its verification so that for every subset of every set of specified activities, a sound workflow instance can be generated. Therefore, we have developed concrete definitions for the template concepts, as well as for the verification and completion of them. We provide an excerpt of these definitions in the following. Definition 8.1 describes the concept of the building block template contained in the declarative container template:

**Definition 8.1** (Building Block Template)
A building block template is a tuple buildingBlockTemplate = (type, name, info, parallelBBset, inferredParallelBBset, successorBBset, inferredSuccessorBBset, predecessorBBset, inferredPredecessorBBset, requiredBBset, mutexBBset, dependsOnSet, stronglyDependsOnSet,
**weaklyDependsOnSet**, **inverselyDependsOnSet**, **inverselyStronglyDependsOnSet**, **inverselyWeaklyDependsOnSet**) where

- info ∈ STRING describes buildingBlock for modeling and reuse issues
- parallelBBset is a finite set of building block templates used to describe building block templates to be executed in parallel to buildingBlockTempl
- inferredParallelBBset is a finite set of building block templates containing all building block templates contained in parallelBBset and automatically added parallel constraints for workflow generation
- successorBBset is a finite set of building block templates used to describe building block templates that are direct successors of buildingBlockTempl
- inferredSuccessorBBset is a finite set of building block templates containing all building block templates contained in successorBBset and automatically added successor constraints for workflow generation
- predecessorBBset is a finite set of building block templates used to describe building block templates that are direct predecessors of buildingBlockTempl
- inferredPredecessorBBset is a finite set of building block templates containing all building block templates contained in predecessorBBset and automatically added predecessor constraints for workflow generation
- requiredBBset is a finite set of building block templates used to describe building block templates whose presence is required in a workflow template that contains buildingBlockTempl
- mutexBBset is a finite set of building block templates used to describe building block templates that are mutually exclusive with buildingBlockTempl
- dependsOnSet, stronglyDependsOnSet, weaklyDependsOnSet, inverselyDependsOnSet, inverselyStronglyDependsOnSet, inverselyWeaklyDependsOnSet are finite sets of property templates used to specify the relations and dependencies of buildingBlockTempl to different situational properties to enable contextual selection of building blocks for automatic generation of situationally matching workflows (see Section 8.2.7 for more details on that procedure)

**BuildingBlockTempl** describes the set of all definable building block templates.

The building block template has a property for storing information about it for reuse. Furthermore, it comprises five sets of other building block templates that implement the four constraints from in Table 8.2. The **hasSuccessor** constraint is implemented using the successorBBset and the predecessorBBset so that each building block template not only has explicitly specified successors but also predecessors. The three sets inferredSuccessorBBset, inferredPredecessorBBset, and inferredParallelBBset contain the transitive closure of building block templates connected to a building block template in different ways. As aforementioned, a building block template is an abstract concept never directly used, but having a set of more concrete specializations. These are used in declarative workflow modeling.

As aforementioned, for the concrete processing of an SE issue based on the template concepts, we also apply a set of individual concepts like a declarative container, situational properties, and different building blocks. In the first place, these are used to store the subset of activities that has been chosen for the concrete issue based on the situational properties. Therefore, the building blocks have the same connections to other building blocks as their templates. As they only have a subset of the properties of the relating template concepts, for brevity, we refrain from discussing all of them. We only show one exemplary definition, which is the building block as shown in Definition 8.2. It shows different sets to capture sequential and parallel connections to other building blocks.

**Definition 8.2 (Building Block)**

A building block is a tuple buildingBlockTempl = (type, name, parallelBBset, successorBBset, inferredSuccessorBBset, predecessorBBset, inferredPredecessorBBset,) where

- parallelBBset is a finite set of building blocks containing the chosen building blocks to be executed in parallel to buildingBlockTempl
- successorBBset is a finite set of building blocks containing the chosen building blocks to be executed as direct successors of buildingBlockTempl
Building blocks are used to build block structures. As the building block template describes the set of all definable building blocks, the conditions are formally defined based on the properties of the building blocks. The conditions are applied to ensure that a sound workflow instance can be built from the building blocks. As the CPM framework relies on block-structured workflows, the conditions enforce the same properties on the building blocks. These properties have been extensively discussed for workflows (e.g. in [MRv10]). Therefore, we will not go into detail regarding the conditions but rather summarize them in Table 8-4. For a thorough discussion, see Appendix B.

### Modeling Conditions

Declarative modeling is used for dynamic workflows where a complete imperative prescription of activities is not suitable. Therefore, all possible candidate activities are specified and connected to situational properties to enable the CPM framework to choose situationally matching activities and build an imperative workflow instance with them. That fact adds requirements to the modeling approach that exceed the ones described in the beginning of this chapter (R:ProcCoverage:ModDy, R:ProcCoverage:ModRe, R:ProcCoverage:ModHi). To be able to build a block-structured workflow instance out of a subset of declaratively specified candidate activities, it must be assured that this subset is convertible to a sound workflow instance. This requires that the initially specified set of activities enables the generation of a sound workflow instance as well. To achieve this, we specified conditions that will be explained in the following. The latter enable the CPM framework to do correctness checks on the different building block templates and the whole specified workflow. These checks are not carried out recursively, meaning that the checks for a workflow apply only for the activities (or building block templates) directly used in that workflow. The internal structure of the used building block templates is checked when creating the latter with separate checks. That way they can be easily reused and a workflow utilizing them has only to be checked on the top level.

The conditions are applied to ensure that a sound workflow instance can be built from the building blocks. As the CPM framework relies on block-structured workflows, the conditions enforce the same properties on the building blocks. These properties have been extensively discussed for workflows (e.g. in [MRv10]). Therefore, we will not go into detail regarding the conditions but rather summarize them in Table 8-4. For a thorough discussion, see Appendix B.

**Table 8-4: Modeling conditions**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>A workflow must not have multiple start or end points.</td>
</tr>
<tr>
<td>C2</td>
<td>Each activity must have at least one connection to other activities.</td>
</tr>
<tr>
<td>C3</td>
<td>No cyclic sequencing shall be specified.</td>
</tr>
<tr>
<td>C4</td>
<td>An activity must have only one successor and one predecessor.</td>
</tr>
<tr>
<td>C5</td>
<td>A building block template must not be sequentially connected to another building block template to which it is also connected in parallel.</td>
</tr>
<tr>
<td>C6</td>
<td>The constraints must not be specified in a way that violates hierarchical modeling enabled by the building blocks.</td>
</tr>
<tr>
<td>C7</td>
<td>A loop template must only contain one building block template.</td>
</tr>
<tr>
<td>C8</td>
<td>A parallel template must contain at least two building blocks.</td>
</tr>
<tr>
<td>C9</td>
<td>A parallel template must contain only building blocks that are connected in parallel.</td>
</tr>
<tr>
<td>C10</td>
<td>A sequence template must contain at least two building blocks.</td>
</tr>
<tr>
<td>C11</td>
<td>A sequence template must contain only sequentially connected building blocks.</td>
</tr>
<tr>
<td>C12</td>
<td>A sequence template must contain a clear start and end point.</td>
</tr>
<tr>
<td>C13</td>
<td>A conditional template must only contain unconnected activities or building block templates.</td>
</tr>
<tr>
<td>C14</td>
<td>A conditional template must contain a minimal number of activities / building block templates.</td>
</tr>
<tr>
<td>C15</td>
<td>One activity must not both require and mutually exclude the same activity.</td>
</tr>
<tr>
<td>C16</td>
<td>If an activity requires another activity, the latter must also be part of that container.</td>
</tr>
</tbody>
</table>

To enable the CPM framework to automatically apply correctness checks based on these conditions for the building block templates and the container templates, the conditions are formally defined based
on the concepts. Based on these conditions, each concept can be automatically checked when it is modeled. For the sake of brevity, we only show one definition of these properties for the building block template in Definition 8.3.

**Definition 8.3 (Building Block Template Properties)**

Let $BB$ and $BB_x \in BuildingBlocks$ be building block templates. Let further $seq$ be a sequence template, $par$ be a parallel template, $loop$ be a loop template, and $cond$ be a conditional template. Then:

1. $\neg(|BB.successorBBset| > 1)$, i.e., a building block template cannot have more than one specified successor.
2. $\neg(|BB.predecessorBBset| > 1)$, i.e., a building block template cannot have more than one specified predecessor.
3. $\neg(BB_x \in (BB.inferredSuccessorBBset \lor BB.inferredPredecessorBBset)) \land BB_x \in BB.inferredParallelBBset)$, i.e., a building block template cannot be connected to the same other building block template sequentially and in parallel.
4. $\neg(BB \in seq.sequentialBBset) + |BB \in par.parallelExecBBset| + |BB \in loop.repeatableBB| + |BB \in cond.conditionBBset| > 1)$, i.e., a building block template cannot be contained in more than one other building block template.

With scheme a) and b), a building block template will be classified as inconsistent if it has more than one successor or predecessor (cf. C4). Scheme c) is applied to classify the building block template as inconsistent if it has a sequential and a parallel connection to the same building block template (cf. C5). Finally, to implement the first part of C6, scheme d) is applied to classify a building block template as inconsistent if it is contained in more than one other building block template (i.e., *sequence template, parallel template, conditional template, or loop template*). This is achieved utilizing the specific properties of these building block templates like the *sequentialBBset* that contains all building block templates comprised in a sequence template.

**Auto Completion**

The two sequencing constraints hasSuccessor and hasParallel allow for the easy modeling of simple activity sets to support humans in quickly capturing extrinsic workflows on the fly. These workflows can be enhanced with more complexity utilizing the different building block template types that, in turn, hide complexity from the humans. However, these constraints enable the modeling of sequences that endanger the ability to convert them to block-structured workflows. Furthermore, as the declarative activity specification only contains candidate activities which are then automatically selected by the CPM framework, several of them might be omitted for a workflow instance. This could result in sequences not convertible to a block-structured workflow. Some sequences might not be convertible into a workflow instance at all because these activity subsets might contain activities having no relation to other activities anymore. The activities once connected to them might have been omitted in a specific situation.

Therefore, we introduce an auto completion feature, which enables the automatic extension of the specified declarative workflows through additional connections between the activities. Based on the sequencing constraints in place, additional sequencing constraints can then be added such that each subset of the activities can be transformed into an imperative workflow instance. This is illustrated by Example 8-6.

**Example 8-6 (Auto completion example):**

Figure 8-11 shows two workflows. On the left, the initially specified constraints are shown. In the middle, the automatically inferred constraints and on the right examples of generated workflow instances are depicted. The upper left workflow contains four candidate activities. However, for a concrete situation, a subset of them could be chosen for enactment. Omitting activities ‘C’ or ‘B’, for
example, would result in two unconnected parts. This would prohibit the building of a workflow instance from it as it would not be clear, in which order they should be executed. Therefore, the auto completion feature adds a set of constraints that allow building a correct imperative workflow instance as shown in the upper right.

For the second example, the initial specification is already problematic as it cannot be determined with which activities the workflow shall be started: ‘A’ and ‘B’? Or ‘E’? Or all of them? By adding constraints, the third possibility is chosen. Note that for this example, transitive successor relationships (e.g., between ‘A’ and ‘D’ in the first example) are omitted for the sake of readability.

As shown in Example 8-6, the approach for converting all declaratively specified activities to imperative workflow instances is grounded on a particular concept: partitioning the activities of the declarative specification into different consecutive sections. All parallel activities build one of these sections (e.g., ‘G’, ‘H’, ‘I’, and ‘D’ in Figure 8-11) and all activities being direct predecessors of this section build another one (e.g., ‘F’ and ‘C’). This can be done recursively and applies the successor activities as well. This is a simplification we apply deliberately to enable completion and conversion of arbitrary activity structures based on the constraints we have defined.

To enable the addition of constraints, further sets contained in building block templates are used: inferredSuccessorBBset, predecessorBBTransitiveClosureSet, and inferredPredecessorBBset. That way, additional constraints may be stored for each building block template. However, the set of initially specified constraints remains untouched. Definition 8.4 shows what constraints are added and how this is accomplished while Figure 8-12 illustrates them.

**Definition 8.4 (Workflow Auto-Completion)**

Let declarativeTemp $\in$ DeclarativeContainerTemplates be a declarative container template containing the building block templates $\{BB_1, BB_2, ..., BB_n\} \in$ BuildingBlockTemps. Then additional constraints are added between the building block templates based on the following rules:

a) $BB_{n+1} \in BB_n$.successorBBset $\implies BB_{n+1} \in BB_n$.inferredSuccessorBBset, i.e., all successors of a building block template are contained in its inferred successor set as well

b) $BB_{n+1} \in BB_n$.predecessorBBset $\implies BB_{n+1} \in BB_n$.inferredPredecessorBBset, i.e., all predecessors of a building block template are also contained in its inferred predecessor set

c) $BB_{n+1} \in BB_n$.parallelBBset $\implies BB_{n+1} \in BB_n$.inferredParallelBBset, i.e., all parallel building blocks of a building block template are contained in its inferred parallel set as well

d) $BB_{n+1} \in BB_n$.inferredSuccessorBBset $\land BB_{n+2} \in BB_{n+1}.inferredParallelBBset \implies BB_{n+2} \in BB_n$.inferredSuccessorBBset, i.e., if there are two or more parallel building block templates and one of them succeeds another building block template, all other parallel building block templates are successors of that building block template as well.
To infer constraints while leaving the sets for the initially specified constraints untouched, at first, the initially specified constraints are added to the sets of the inferred constraints (cf. Definition 8.4a-c). The succeeding operations are executed on the latter. If a building block template has a successor that, in turn, has a parallel building block template, this building block template will also be a successor of the first building block template (cf. Figure 8-12d and Definition 8.4d). If there are two or more parallel activities and one of them has a successor, this building block template will be the successor of all other parallel building block templates (cf. Figure 8-12e and Definition 8.4e). If two building block templates exist in parallel and both of them have a predecessor, the two preceding building block templates will be considered as parallel as well (cf. Figure 8-12f and Definition 8.4f). The same applies if two parallel building block templates both have a successor (cf. Figure 8-12g and Definition 8.4g).

Based on all specified and inferred constraints, the transitive closures are built for sets inferredParallelBBset, inferredSuccessorBBset, and inferredPredecessorBBset and added to them. Note that these actions are to be executed recursively to be able to add connections like the ones between activity ‘A’, ‘B’ and ‘E’ in Figure 8-11. The added constraints combined with their transitive closure allows building imperative workflow instances out of arbitrary subsets of the candidate activities as they are categorized in sections and all of the latter have mutual successor constraints now.

8.2.6. Treatment of Different Workflow Types

There are different combinations of intrinsic and extrinsic workflows that are modeled imperatively or declaratively (cf. Figure 8-2). This section briefly explains how different combinations are enabled. As both declarative and imperative workflows are realized by the work unit container including a sub-concept, it is possible to use both types for intrinsic as well as extrinsic workflows.
There are different levels of automation concerning workflow initiation: intrinsic workflow instances are automatically started, as they are part of the running SE process. By contrast, extrinsic workflow instances can be started out of different situations: First, they may be started manually by a human. Second, they may be started semi-automatically, e.g., when an activity is assigned to a human in a bug tracking system monitored by a sensor. The sensor then generates an event triggering the instantiation and start of a new workflow instance for the respective human. The third case is the follow-up activities required by other activities (cf. Section 8.2.1). These are automatically initiated by the CPM framework. The latter case is illustrated by Example 8-7.

Example 8-7 (Follow-up activities):
Consider a source code modification conducted by an intrinsic activity. This modification was applied to an artifact belonging to the interface of a component. Thus, the change not only impacts the component itself as well as its implementation, but other areas as well. The areas ‘testing’ and ‘architecture’ might be impacted, since the integration tests or the architecture specification has to be adapted eventually. Details on how to determine the impacts one area has on another and to govern the follow-up activities are described in Chapter 10.

8.2.7. Concrete Procedure for Extrinsic Workflow Enactment

The concrete procedure for handling SE issues is as follows: First, the workflow representing the case is modeled declaratively as illustrated in Figure 8-13A. This procedure includes the composition of the workflow out of various building block templates, their connection to situational properties, and the connection of both to a case. After modeling is completed, the CPM framework validates it and applies the auto completion feature (cf. Figure 8-13B). An event indicating that an SE issue is assigned to a human serves as an entry point for enacting a workflow instance. In turn, this event may stem from various sources. Examples include the assignment of an SE issue to a human in a bug tracker system or the manual triggering by a human. The next step is to determine a case for that issue, like ‘Bug fixing’ or ‘Refactoring’. Depending on the origin of the event, this can be done implicitly or explicitly by the human.

Figure 8-13: Concrete procedure
When creating an SE issue, it derives the work unit container and the properties from its associated case template. Usually, the work unit container only contains one activity for analyzing the case (cf. Figure 8-13C). Each property holds a value indicating how much this property applies to the current situation. These values may be influenced by various factors defined by the property template (cf. Figure 8-10). There are four possible relations between entities affecting the properties that capture impacts: strong negative, weak negative, weak positive, and strong positive. These are used to compute the values of the properties. The values are initialized with ‘0 (neutral)’ and are incremented / decremented based on the relations to the different influences. For simplicity, the values are limited to a range from ‘-2 (very low)’ to ‘2 (very high)’, thus representing five possible states of the degree to which the property applies to the current situation.

To be able to select appropriate building blocks according to the current properties, six connections between the building block templates and the property templates may be utilized. These include the sets ‘weaklyDependsOnSet’, ‘stronglyDependsOnSet’, and ‘dependsOnSet’, which constitute properties of the building block template. These properties indicate that the latter is suitable if the value of the property is ‘1 (high)’ or ‘2 (very high)’, or in both situations. For negative values, there are also three of these sets (e.g., ‘ inverselyDependsOnSet’, cf. Definition 8.1). Each building block can be connected to multiple properties.

The first step of an extrinsic workflow is to gather contextual information (cf. Figure 8-13C-D). This information may stem from various sensors that provide information on the state transitions of SE tools or directly from the human via the GUI. After having determined the properties of the case, the additional activities matching the current situation and product must be selected as depicted in Figure 8-13E. This is done by generating a set of concrete building blocks for the case. In particular, this is a subset of the building block templates assigned to the relating case template. The properties of the building blocks are initialized using the concrete properties of the case and the building block connections specified in the case template.

After having determined the set of building blocks, the concrete workflow skeleton of the case may be adapted to incorporate all activities applicable in this situation (cf. Figure 8-13F). Therefore, checks must be applied: First, we check the existence constraints (mutual exclusion and requirement) for the chosen subset of activities to ensure that required activities are not omitted in the chosen subset or that mutually exclusive ones appear. In the latter case, two alternatives can be considered: required activities are added to the subset or the workflow generation is aborted. Example 8-8 illustrates one of the existence constraints.

<table>
<thead>
<tr>
<th>Example 8-8 (Activity subset consistency check):</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 8-14 illustrates the ‘requirement’ constraint by an excerpt from a bug fixing workflow. Two building blocks belonging to the latter are shown. The first one ‘Prepare Bug Fixing’ shall be applied at the beginning of the workflow. It contains activities for creating a separate branch for the bug fix, reproducing the error, and checking dependencies of the concerned artifacts. The other building block, ‘Integration’ shall be placed at the end of the workflow. It contains two activities relating to the integration of the developed code, both to be inserted conditionally. The requirement constraint concerns the separated branch used for the bug fix: If, due to situational properties, the creation of a separate branch is chosen, there also has to be an activity for integrating that separate branch.</td>
</tr>
</tbody>
</table>
After checking correctness, workflow generation is started. Example 8-9 illustrates how a workflow instance is generated out of a list of building blocks.

Example 8-9 (Workflow generation):
Figure 8-15 shows a building block structure that is converted into a workflow instance. The building blocks and activities with dashed red lines have been chosen on account of properties representing the current situation. The first parallel has been chosen as well as two of its contained activities. Therefore, the AND pattern is integrated into the workflow instance containing these two activities. The second structure, a loop containing a sequence of two activities is completely integrated into the workflow instance as all of them have been chosen. The conditional also has been chosen. However, since the contained activity has not been chosen, the conditional is omitted as it would only be an empty XOR pattern in the workflow instance. The last structure, a loop containing a parallel with two activities, is also omitted as the loop has not been chosen.

The automatic workflow generation illustrated in Figure 8-15 is executed by Algorithm 8-1, which is applied on the list of activities to create an executable workflow instance out of these. For the sake of simplicity, the functionalities for creating the context management and process management concepts are abstracted to simple function calls like, e.g., insertParSplit() that would insert a parallel AND split into a workflow instance.
Algorithm 8-1: ConstructWorkflow
(Pseudo Code for constructing a workflow instance out of a specified set of building blocks)

Require: List allBBs [Building Blocks BB], work unit container skeleton, activity extensionAct

1: String errorCode ← empty String
2: if allBBs is empty
3: errorCode ← ‘emptySet’
4: return errorCode
5: end if
6: Arc marker ← extensionAct.getOutgoingArc()
7: Node afterInsertion ← marker.getTarget()
8: marker.setTarget(NULL)
9: while allBBs not empty do
10: List nextBBs ← determineBBsWithNoPredecessor(allBBs)
11: if nextBBs is empty
12: errorCode ← ‘structureInconsistent’
13: return errorCode
14: end if
15: if nextBBs.size < 2
16: errorCode ← BBtreatment(nextBBs.firstElement, skeleton, marker)
17: else
18: AndSplit split ← insertParSplit(marker, skeleton)
19: List branches ← new List()
20: for all nextBBs do
21: insertBranch(split, marker)
22: errorCode ← BBtreatment(nextBB, skeleton, marker)
23: branches.add(marker)
24: end for
25: insertParJoin(marker, split, skeleton, branches)
26: end if
27: if allBBs.isEmpty()
28: errorCode ← emptySet
29: end if
30: end while
31: return errorCode

Algorithm 8-1 expects a list of building blocks, a work unit container, and the activity after which the new activities shall be inserted (extension activity) as input. In that case, the container corresponds to the workflow skeleton that will be extended with the newly inserted activities. First, the algorithm defines an error code variable to be used as return code of the algorithm (Line 1). In case of an empty building block list, the algorithm returns with an error code (Lines 2 - 5). Otherwise, it creates a marker for the insertion of new activities. For this purpose, the arc coming from the extension activity will be used. Further, the activity that will be the successor of the inserted activities is stored (Line 6). After that, the algorithm iterates through the list of building blocks until it is empty (Line 9). From the list, all building blocks having no predecessors that are also in the list are removed from the list and used for workflow generation. This selection is done by a call to the algorithm determineBBsWithNoPredecessor(), a simple algorithm that determines building blocks with no predecessors in the given list and removes them from it. If no such building blocks can be found, the algorithm terminates with an error (Lines 11 -14). If such a building block is found, it is passed to algorithm BBtreatment() (cf. Appendix B), which is in charge of integrating the activity into the workflow instance (Lines 15 -16). In turn, this algorithm uses several other algorithms for the different building block types. For a discussion of these, we refer to Appendix B. If multiple parallel building blocks are found with no predecessor, they are also included by the BBtreatment() algorithm. However, each of them is integrated in a separate branch of a parallel AND pattern into the workflow instance (Lines 17 -26). Finally, the last one of the newly inserted nodes is connected with the activity of the skeleton that must succeed them (Lines 27 -28).
8.2.8. Modeling Effort

While the approach presented in this chapter consists of various components and may appear to involve a fair amount of complexity, it does not impose complicated modeling or undue workflow enactment effort for humans. The required elements for modeling are discussed in the following:

- Context properties: The CPM framework needs explicitly modeled context properties for selecting appropriate activities. These properties must be connected to other facts to be automatically computed; e.g., ‘If the skill level of the applying human is low, the risk is increased’. These properties can be reused for all cases and must be modeled only once.
- Activities: The workflows consist of activities that must be specified and connected to context properties to enable the CPM framework to know when they apply. Like the properties, the activities only must be modeled once and can be reused.
- Building blocks: building blocks are used to group activities and to govern their sequencing. They are further connected to context properties and may be reused. Building blocks offer great potential for reuse and for simplifying modeling: They are encapsulated as simple activities and thus simplify the structure of the containing work unit container. Consider the four code review activities from Example 8-1, which may be grouped together, e.g., in a parallel building block called ‘Review Activities’. For other workflows modeled in the future, the latter can be used instead of incorporating multiple activities and choices, leaving the CPM framework responsible for selecting the matching activities for the current situation during run-time.
- Cases: For each concrete issue like ‘Bug Fixing’, one case template is defined. Such definition is simple since all defined activities, context properties, and building blocks can be reused. The structure of the cases is simple as there are only four constraints needed for connecting the activities or building blocks. More complex control flow modeling is handled and encapsulated by the building blocks.

Example 8-10 illustrates simplicity by showing the top level of the declaratively specified activities for the bug fixing issue (cf. Example 8-1). The whole issue including a building block library is presented in Chapter 14.

Example 8-10 (Bug fixing issue modeled declaratively):
Figure 8-16 shows how a bug fixing workflow can be modeled simply yet dynamically using the present approach: The depicted workflow contains various elements out of a pre-configured set of building blocks (a building block library). As example, consider the ‘development cycle’ building block: It contains various other activities like implementation, developer test, or review activities that are executed in a loop. It constitutes a standard set of activities that may be reused in various workflows. To match different situations, the building blocks are connected to situational properties, enabling the CPM framework to automatically tailor them to various situations. As example, consider the review activities being part of the development cycle: According to the situation, different activities could be chosen, like code review or code inspection, or even no review activity at all. Thereby, complexity is hidden from humans and managed by the CPM framework, leaving a relatively simple sequential workflow for the bug fixing issue.

8.3. Discussion

This section discusses work from different areas related to the presented concept. In particular, we cover the two main topics of this concept: declarative process specification and (automated) configuration or generation of processes.
8.3.1. Declarative Process Models

Declarative (i.e., constraint-based) process modeling is an emerging paradigm receiving increasing attention [DHM+96, Mont10, MPA+10, PSSA07, PSA07, Pesi08, SSO01, WBB04, WPZW10, ZPW12]. An example of a system enabling declarative modeling is DECLARE [PSSA07, PSA07, Pesi08]. In particular, DECLARE features a declarative approach where workflows are described only by certain constraints governing the relations of different activities. These constraints are based on Linear-time Temporal Logic (LTL). Utilizing them, both permitted and prohibited relations can be described. Examples include existence constraints like ‘co-existence(A,B)’, which prescribes the presence of B when A is present and vice versa. Sequencing constraints are also covered, as e.g., ‘response(A,B)’, which implies that B must appear after A. Different workflows are organized as cases containing a set of constraints.

Researchers agree that there are certain problems associated with declarative process models [Mont10, Pesi08, PWZ+11, HBZ+14, ZSH+15]. The specification of large workflows utilizing constraints creates a great amount of unstructured constraints, making maintenance and evolution burdensome because they severely affect understandability [PWZ+11]. Therefore, it is often recommended to use declarative approaches only for small workflows [Pesi08]. In addition to that, if modelers focus on desired behavior, models might become over-specified [Mont10]. As opposed to the approach presented in this chapter, there exists another drawback with these approaches: Specified elements cannot be used multiple times in one specified workflow, and reuse is not as simple as in our approach with the concept of the building block.

To overcome the limitations of declarative approaches, there have been attempts to combine them with imperative processes similar to our approach [SSO01, WPZW10, ZPW12]. An example is the Pockets of Flexibility approach [SSO01]. It allows modeling loosely-specified processes. The latter are based on an imperative pre-specified process model where certain activities (pockets of flexibility) are only concretized when they are instantiated. For such activities, diverse activities and process fragments can be modeled including constraints that govern how they should be composed. Another approach allowing for such combination is the Alaska Simulator [WPZW10, ZPW12]. It allows for iterative refinements of processes. Therefore, a set of activities and constraints as well as imperative processes can be modeled, which is further refined while being executed. In addition, a schedule provides a forecast on the scheduled but not yet executed activities. Finally, DECLARE also offers such a combination utilizing the imperative YAWL tool [vdH05]. That way, DECLARE models can be used within YAWL as sub processes.

While there are similarities to the described approaches, our approach provides a unique set of features overcoming certain problems and limitations inherent to them. Pure declarative approaches suffer from the problems of understandability as discussed. Our approach, therefore, limits the complexity of the constraints. This further limits expressiveness. However, as such approaches are intended for small and dynamic operational workflows, which are combined with other imperative ones, this is not significant. Furthermore, our approach does not combine declarative and imperative parts being independent from each other. It rather uses simple declarative modeling in combination with contextual configuration to automatically create small, understandable, and situational workflows. None of the discussed approaches has such an automatic contextual component that in fact reduces the number of involved activities for enactment. Another major advantage of our approach is its focus on reusability: the building block concept supports easy reuse of arbitrary activity structures whose complexity is hidden.

8.3.2. Process Model Configuration

As the approach we introduced in this chapter deals with automatically configuring run-time workflows out of pre-specified candidate activities, in principle, other approaches for process configuration are also applicable. We do not go into detail about the different approaches since we already discussed them in Chapter 7. Instead, we summarize the differences of our approach compared to them: Our approach features a simplified declarative way of modeling hiding significant
complexity. This is simpler than defining configuration options for a pre-specified model in a new notation [ATW+14]. Furthermore, the concept of building blocks allows producing much smaller and better understandable models compared to configuration approaches that incorporate all possible activities in one model. The building blocks, however, add the advantage of easily reusable arbitrary activity structures. Finally, our approach focuses on the enactment and, thus, the automatic selection of appropriate activities. The contextual integration enables the CPM framework to automatically apply the entire configuration. In turn, this relieves humans from that task and provides them with situationally tailored workflows.

8.4. Summary

The SE domain epitomizes the challenge faced by adaptive WiMS. Since SE is a relatively young discipline, automated process enactment in real projects is often immature. One of the open issues in this domain is the gap between the top-down abstract archetype SE process models lacking automated support for real enactment, and exactly the actual execution with its bottom-up nature. An important factor affecting this problem are activities belonging to specialized issues such as bug fixing or refactoring. These are not covered by archetype SE processes and, as they are on often so variegated, their pre-modeling is not feasible and not cost-effective.

On one hand, the significance of our approach is that workflows for SE issues, which are extrinsic to archetype SE processes, are not only explicitly modeled, but may be dynamically adapted to the current issue and situation. This is done based on various properties derived from the current product, process, context, and human. Thus, it is possible to provide situational and tailored support as well as guidance for the software engineers that process SE issues. On the other hand, the proposed approach shows promising perspectives for improving and simplifying process definition for extrinsic workflows. The initial effort to define all the activities, cases, properties, and workflow skeletons may not be less than predefining huge workflows for the SE issues, but the reuse of the different concepts is fostered. Thereafter, the creation of new issues is simplified since they only need to be connected to activities they should contain. The latter are automatically inserted later in order to match the current situation. Yet the main advantage is of semantic nature: the process of case creation is much more problem-oriented compared to the creation of large process models. The process engineer can concentrate on activities matching the properties of different situations rather than investing cognitive efforts in the creation of huge rigid process models matching every possible situation.

To be precise, this approach satisfies three categories of requirements: Generally, it enables the unification of intrinsic and extrinsic workflows, supports and simplifies extrinsic workflow modeling, and offers automation and execution support for them in alignment with properties of various situations. Our approach is not only able to model and execute intrinsic as well as extrinsic workflows, but also to uniformly realize them during enactment.

The higher level of dynamicity inherent to extrinsic workflows is accommodated by a declarative, problem-oriented method of modeling. The latter allows defining a dynamic set of candidate activities rather than modeling large rigid workflow templates. Furthermore, the hierarchical structure of the declarative modeling approach featuring the concept of the building blocks supports modeling: complexity is hidden at build- as well as at run-time. Reuse is fostered as process models can be separated not only by sub-processes, but also by logical blocks.

Finally, effective as well as efficient execution of extrinsic workflows is fostered by automated contextual governance. The properties of various situations are automatically detected by the CPM framework and are used to select a subset of activities that matches that situation. This is enabled by the explicit modeling of these situational properties and their influences on the workflow templates.

This chapter discusses how the CPM framework enables automated integration of software quality management with SE processes\(^1\). SE projects have faced difficulties in meeting budget, schedule, functionality, and quality targets [Broo87, Glas98, NaRa68]. A more recent study by the National Institute of Standards and Technology, for example, revealed that most delivered software products are still stricken by bugs and defects [Jone10]. While some of these difficulties might be ascribed to a misaligned planning environment in certain organizations [EvVe09], the project pressures and resulting issues will likely linger due to global competition and other influences [Your03]. Other difficulties can be attributed to the adolescence of SE as a discipline and certain unique product properties that affect the SE process, such as the complexity, conformity, changeability, and invisibility of software [Broo87]. Additionally, the extent (too little or too much) and timeliness of Software Quality Assurance (SQA) significantly impacts overall project costs [Abde88], making effective and efficient SQA vital. Yet, it remains laborious to manage and apply the appropriate low-level SQA measures (actions) in a timely fashion during SE process enactment. In order to achieve software quality goals, these must be defined and concretely and contemporaneously measured [Kan02]; yet this is often challenging for various SE organizations [STT06]. Especially, small and medium sized companies often struggle to achieve high quality levels. This often results from the increased complexity of their growing organizational structures, the lack of process maturity, and the lack of dedicated quality management personnel.

While SE process models foster development efficiency [GGK06], they are often defined rather abstractly and thus fail to provide specific detailed guidance for the activities actually executed at the operational level. Furthermore, processes are often defined rigidly beforehand. According to [McCo01], however, during their enactment, reality often diverges from the planned process.

Automated guidance for combining SQA with SE process management is not prevalent yet. Challenges in SE projects are presented at both the product and process levels based on the nature of software artifacts and manually driven processes. For example, product intangibility hinders effective retrieval of timely information about its quality status. Additionally, the combination of abstract process definitions and concrete operational workflows make targeted process guidance for developers irrelevant, complex, or costly. Thus, issues with software artifacts often cannot be detected promptly and, even if they are, the contemporaneous integration of software quality measures into the humans’ workflows to solve the issues is not possible. Quality measures might come into focus and are applied close to release, or, when the project is behind schedule, they may be jettisoned altogether. Generally, it is acknowledged that their application in earlier development stages saves time as well as costs [Abde88, SHK98]. The proper application of quality measures is problematic, since their effectiveness and efficiency depend on many factors, such as the applicability of the measure, the project timing, worker competency, and correct fulfillment [Abde88]. For the sake of clarity, quality measure in this chapter is meant in the sense of a specific action intended to produce some effect on the quality of software artifacts.

Automated support for and governance of the coordinated integration of SQA in SE process management offers promising perspectives for addressing shortcomings in current SQA approaches. However, automatically supporting SQA remains a challenge, which can only be achieved with the

---

\(^1\) This chapter is partially based on the publications [GrOb10], [GOR10b] and [GOR11b].
basic features we have elaborated in the preceding chapters: access to and automatic processing of contextual data, fine-grained human activity management, integration of process enactment with contextual information, and abilities for automatic process adaptations. In particular, this chapter provides a description of an approach for integrating process management and SQA, elucidating the following areas:

- Automatic detection and management of source code related problems in a SE project;
- Automatic assignment of quality measures to detected quality problems;
- Automatic strategic prioritization and alignment of quality measures to project quality goals;
- Tailoring of quality measure (action) proposals to the situation;
- Automatic integration of quality measures into the workflow of software engineers.

The remainder of this chapter is organized as follows: Section 9.1 elicits requirements detailing the superordinate requirement of quality management support. Section 9.2 describes the various aspects of the solution approach. Finally, Section 9.3 discusses related work and Section 9.4 concludes the chapter.

9.1. Requirements

This section details the elicited requirement (R:Qual) that deals with the automated integration of quality and process management in SE. Potential problems concerning SE quality measure distribution are first illustrated along Example 9-1:

Example 9-1 (Process and quality management):

The Company uses a process model for SE. This SE process features descriptions of activities, roles, milestones, and artifacts. Its application is based on the use of various documents without automated governance. In the Company, activities for developers are planned and scheduled on a coarse-grained level, leaving the coordination of what has to be done to the developers. Therefore, the SE process does not really “touch” the developers, and their actual activities are difficult to trace. The quality of the source code is not monitored continuously and static code analysis tools are only used sparsely by the developers. Thus, there is a lack of awareness about the actual quality state of the source code. Thus, deterioration of the quality goes undetected and quality measures are only taken at the end of projects if there is time left or concrete bugs exist. There is no explicit source code quality management and, thus, there are no quality goals for the source code. Therefore, it is not possible to support certain quality properties of the source code proactively (e.g., maintainability or performance). Measures are mostly taken only in reaction to occurring problems. These unplanned measures are applied under time pressure by the developers, thus further confusing and delaying the activities planned. Finally, there is no monitoring or assessment of the applied measures, and it cannot be determined which ones were really affective.

In response to situations as described by Example 9-1, the following requirements are defined. Based on solutions to basic requirements (R:ContInt and R:AutoProc, cf. Chapter 4) covered in Chapter 7, we now elicit more specific ones related to SQA as extension to requirement R:Qual (cf. Chapter 4). To enable automated decisions on quality measure assignments, any framework support should be aware of its environment and the context of the situation at hand.

- **Requirement R:Qual:ProbAw (Problem awareness):** To be aware of problems in SE projects, the framework must be able to integrate information on SE process or SE product problems from various sources (e.g., external tools measuring the state of the source code or bug tracking systems).
- **Requirement R:Qual:QualAw (Quality opportunity awareness):** To enable automated integration of quality measures at run-time, the framework must be aware of quality opportunities, meaning time points when a human can cope with a quality measure. This
requires knowledge about schedules, meaning the abstract activities that have been scheduled and estimated for humans.

The selection of appropriate quality measures matching project needs and the humans’ situations during SE process enactment constitutes another challenge. Various factors must be taken into account for ensuring effectiveness and efficiency.

- **Requirement R:Qual:StratSel (Strategic measure selection):** Applied quality measures should be automatically chosen during run-time in alignment to project goals in order to match the defined strategy of the project.

- **Requirement R:Qual:ProacMeas (Proactive measures):** Quality measures should not only rely on detected problems, but consider common quality enhancement as well. Thus, proactive and reactive measures should be available.

- **Requirement R:Qual:SitTail (Situational measure tailoring):** Context-sensitive tailoring of proposed measures is desirable taking different factors of the actual situation into account, e.g., properties of the human applying it and the time point of the application.

- **Requirement R:Qual:MeAss (Measure assessment):** The selection of measures should utilize their proven effectiveness to optimally match with specific environments or situations in different companies. Therefore, continuous monitoring of the source code quality is essential to detect potential impacts of applied measures on overall quality. In particular, a relation between the application of SQA measures and the evolution of source code quality should be established to assess the effectiveness of the measures.

To be able to effectively and efficiently distribute quality measures in an automated fashion their application should not be cumbersome or cause problems with the process.

- **Requirement R:Qual:SeamProc (Seamless process integration):** The automated distribution of quality measures should not interfere with standard SE process enactment. Rather, it should be seamlessly integrated with other workflows being part of the process. This fosters enhanced traceability of quality measure application.

- **Requirement R:Qual:UsrDist (User disturbance):** Same as standard SE process enactment, quality measure integration should also not interfere with the humans; i.e., humans should not be disturbed by quality measure proposal.

The requirements are summarized in Table 9-1:

<table>
<thead>
<tr>
<th>Req ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R:Qual</td>
<td>Automated Quality</td>
<td>The CPM framework shall provide automated support for integrating process</td>
</tr>
<tr>
<td></td>
<td>integration</td>
<td>management and SQA.</td>
</tr>
<tr>
<td>R:Qual:SeamProc</td>
<td>Seamless process</td>
<td>Software quality measures shall be integrated seamlessly with the SE</td>
</tr>
<tr>
<td></td>
<td>integration</td>
<td>process.</td>
</tr>
<tr>
<td>R:Qual:UsrDist</td>
<td>User disturbance</td>
<td>Humans shall not be disturbed by automated SQA integration.</td>
</tr>
<tr>
<td>R:Qual:ProbAw</td>
<td>Problem awareness</td>
<td>The CPM framework shall have a facility to generate an awareness of problems that are existent in a project.</td>
</tr>
<tr>
<td>R:Qual:QualAw</td>
<td>Quality opportunity</td>
<td>The CPM framework shall be able to generate an awareness of opportunities</td>
</tr>
<tr>
<td></td>
<td>awareness</td>
<td>where a human can be given an additional activity to counteract problems.</td>
</tr>
<tr>
<td>R:Qual:StratSel</td>
<td>Strategic measure</td>
<td>The CPM framework should automatically select quality measures matching the strategic quality goals of a project.</td>
</tr>
<tr>
<td></td>
<td>selection</td>
<td></td>
</tr>
<tr>
<td>R:Qual:ProacMeas</td>
<td>Proactive measures</td>
<td>Quality measures should not only be usable for existent problems, but also enable proactive problem prevention.</td>
</tr>
<tr>
<td>R:Qual:SitTail</td>
<td>Situational measure</td>
<td>The distribution of measures to humans should incorporate various contextual factors to match the human's situation.</td>
</tr>
<tr>
<td></td>
<td>tailoring</td>
<td></td>
</tr>
<tr>
<td>R:Qual:MeAss</td>
<td>Measure assessment</td>
<td>Applied measures should be automatically assessed by the framework to improve future measure distribution procedures.</td>
</tr>
</tbody>
</table>
9.2. Quality Management Integration Approach

This section describes our approach to address the elicited requirements. Section 9.2.1 introduces the abstract procedure behind it. The approach is detailed thereafter: Section 9.2.2 describes context detection features, Section 9.2.3 the processing of the quality measures, and Section 9.2.4 the post-processing after measures were applied. Finally, Section 9.2.5 introduces the formal framework the approach is based on.

9.2.1. Solution Procedure

The solution procedure involves three fundamental phases: Detection, Processing, and Post-processing (cf. Figure 9-2). These phases as well as their different steps will be explained in the following.

The Detection Phase continuously enables an awareness of the current project situation to meet the requirements related to context-awareness (cf. Table 9-1). For integrating quality measures, two factors are of particular interest: the presence of problems (cf. R:Qual:ProbAw) – recognized via the ‘Problem Detection’ (cf. Figure 9-2) – and the availability of opportunities for quality measures in the schedule of humans (cf. R:Qual:QualAw) – recognized via ‘Quality Opportunity Detection’ (cf. Figure 9-2).

The Processing Phase deals with the selection and proposal of the quality measures and involves four steps. Utilizing the GQM technique [vBCR02], quality measures (actions) are initially proposed in alignment with project goals to satisfy requirement R:Qual:StratSel. This phase adds proactive measures to the measure proposal process (cf. R:Qual:ProacMeas). In turn, to prepare these measures for their automated application, ‘Measure Tailoring’ incorporates information about the humans applying the measures and the possible points in their schedules in which to apply the measure (cf. R:Qual:SitTail). This leads to a selection of appropriate points (denoted as q-slots) as well as to an automated integration of the quality measures into the human’s concrete workflow instance. We denote such points in the workflow instances as extension points. The application of measures can be accomplished automatically utilizing the additions made to the SE process specifications (cf. Chapter 7). These enable the framework to be aware of matching extension points (e.g., in the workflows) as illustrated in Example 9-2.

Example 9-2 (Extension points for quality measure integration):

Extension points can be applied at different positions in the process having specific properties. Examples include the end of an iteration or a whole project. These could be used for quality measures that cannot be applied during standard development. This example illustrates the use of extension points utilizing the ‘Develop Solution Increment’ workflow of the OpenUP process (cf. Figure 9-1).

![Figure 9-1: Workflow with extension points](image-url)
The extension points are on a rather concrete level; one is attributed to the coding and one to the test coding activity. In both cases, quality measures can be applied directly after or parallel to these activities. As they are rather concrete, the extension points can consider the processed artifacts or packages to enable the efficient integration of quality measures. An example would be the integration of activity ‘Increase code coverage’ to add more developer tests, which concerns a source code package for which the developer has just added a developer test with activity ‘Implement Developer Test’.

‘Measure Application’ (cf. Figure 9-2) is capable of automatically adapting running workflow instances. That way, quality measures can be seamlessly integrated with the standard development workflows (cf. R:Qual:SeamProc). Furthermore, as future workflows of humans are automatically and contextually extended, there is no distraction or extra effort for the humans to receive quality measure assignments (cf. R:Qual:UsrDist).

Finally, to be able to track the quality of the project continuously, in the Post-Processing Phase (cf. Example 9-1), activity ‘Measure Assessment’ (cf. Figure 9-2) is performed via a quality trend analysis. This analysis enables an awareness and automatic assessment of the potential utility of the applied measures, fostering quality (cf. R:Qual:MeAss).

Since each project is unique, the applicability and effectiveness of measures can vary regarding different projects. Therefore, the CPM framework executes an assessment phase to rate the applied measures as well as to incorporate their impact in the given project.

The following procedures (Procedure 9-1, Procedure 9-2, and Procedure 9-3) give a short outline about the temporal coordination of different framework components. The entire procedure can be roughly decomposed into three smaller procedures partly depending on each other. These procedures are fine-grained and, therefore, they contain the main actions shown in Figure 9-2 as well as additional ones to better explain the course of action. The first procedure deals with the different steps executed for processing occurring problems.

**Procedure 9-1 (Problem processing):**
1. **Static Analysis (cf. Figure 9-2 4):** The quality of the source code is measured by static analysis tools.
2. **Report Event:** The event management component recognizes the creation of a new static analysis tool report and notifies the quality management component.
3. **Report transformation:** The quality management component uses the new report to create a unified one. The resulting report contains only metrics whose values violated predefined thresholds.
4. **Rules Processing (5):** For each metric documented in the report, a quality measure is automatically assigned via pre-defined rules.
5. **Quality Trend Analysis (6)**: The unified report is utilized by the context management component to calculate values for predefined KPIs to obtain aggregated and more meaningful values indicating the condition of the source code.

6. **Measure prioritization (7)**: The quality management component prioritizes the measures in the unified report according to the defined quality goals of the current project with an automated goal-question-metric approach (AGQM, cf. Section 9.2.3).

The second procedure shows the different steps required for detecting and processing opportunities for quality activities.

**Procedure 9-2 (Quality opportunity processing):**
When a human finishes an activity, the q-slot detection is started within the context management component. A q-slot defines an opportunity to propose a quality measure to a human. If a q-slot is available, the quality management component is triggered by the context management component to generate an ordered list of proposed measures. This list is then used by the tailoring process in the context management component to select a measure that is then integrated into a workflow by the process management component. The concrete steps are shown as follows:

1. **Workflow estimation (cf. Figure 9-2 1)**: The different human assignments of a project must be estimated in respect to time consumption. This can be accomplished with an external project management tool.
2. **Assignment import**: The event management component imports the assignments into the CPM framework. For each assignment there are predefined workflows initiated by the context and the process management components.
3. **Workflow enactment (2)**: During workflow enactment, different assignments are completed by humans.
4. **Q-Slot detection (3)**: Assignment completion triggers the context management component to execute the q-slot detection.
5. **Measure request**: If a q-slot is available, the context management component requests matching quality measures from the quality management component.
6. **Measure list creation**: Upon the request, the quality management component creates an ordered list of proactive or reactive measures that match the quality goals of the project.
7. **Extension Point Determination (8)**: The context management component selects an extension point related to a running or planned workflow of the human with the detected q-slot.
8. **Measure selection (9)**: The context management component selects a measure out of the list that matches the current situation.
9. **Measure application (10)**: The context management component utilizes the process management component to seamlessly integrate the quality measure into the humans’ workflow instance.

The third procedure is utilized for assessing applied quality measures.

**Procedure 9-3 (Quality measure assessment):**

1. **Assessment point configuration**: A human decides at which points of the process a quality measure assessment should be conducted. Examples include the end points of an iteration or project.
2. **Quality Trend Analysis (6)**: The quality management component continuously calculates values for defined KPIs utilizing reports of analysis tools.
3. **Measure assessment (11)**: At the configured time points, the context management component is triggered to assess the applied measures, considering the calculated KPIs.

### 9.2.2. Context Detection

This section describes the detection phase and its steps. In particular, it discusses how the CPM framework gains awareness about problems in the source code and about humans’ activities including opportunities for software quality measures.
Quality Opportunity Detection

To enable the automated detection of quality opportunities, an awareness of the activities that have been planned and scheduled is required. These activities are modeled by the assignment, which has properties to capture estimated durations (i.e., planned start and end times). These assignments can be created, estimated and scheduled in the CPM framework or be imported from other tools. This is illustrated by Figure 9-3.

Figure 9-3 shows an example of a simplified schedule containing assignments ‘As1’ – ‘As4’, which are estimated to take three days each. The scheduled assignments are then taken for enactment. Figure 9-3 illustrates the connection of assignment ‘As1’ to four assignment activities ‘Aa1’ – ‘Aa4’ for enactment. Optionally, the schedule can be imported from an external tool. That way, the activities can be estimated and scheduled from the business side, e.g., utilizing a process management tool like microTOOL in-Step [Micr15]. Then, they can be automatically used for enactment in the CPM framework.

Our approach features two different triggers for quality opportunities: The first one is early assignment completion. If a human finishes an assignment earlier than necessary, a quality measure can be assigned to him without delaying forthcoming activities. However, it cannot be assumed that this situation will occur frequently. Therefore, we integrated another trigger called quality overhead factor. It enables the a-priori specification of a certain percentage of the project workload that should be reserved for quality activities. If the human has not reached that amount during process enactment yet, a quality measure may be applied. This can be combined with a quality function indicating how much time for quality should be spent in which project phase. Since it can be beneficial to adjust the work allocation for quality measures based on the stage of an SE project [Abde88, SHK98], this might improve both the effectiveness and efficiency of the quality efforts taken.

Procedure 9-4 shows the exact steps taken to enable quality opportunity detection including a quality overhead factor.

Procedure 9-4 (Quality opportunity calculation):
1. Overhead factor specification: A human specifies quality overhead factor as a number between 0 and 100 to be used as a percentage of work hours for quality measures.
2. Quality function specification: A human specifies a quality distribution function to govern the distribution of quality measures in a given period. Therefore, a strictly monotonically increasing function must be defined and an interval in which the values of that function will be used for

---

2 See [LPCR15, LPCR13] for techniques managing such temporal constraints.
quality distribution. That way, the function values (i.e., f(x)) indicate the percentage of quality measure time being available for the input value (i.e., x) that represents the time point in the given period. As standard values, an interval [0, 1] is taken as well as f(x) = x as function. We assume these values for a steady distribution of the available quality overhead. This is shown in more detail in Example 9-3.

3. Period specification: A human defines a period for the quality opportunity detection, e.g., a project or a phase of it or a development iteration in an agile project.

4. Workflow enactment: The defined period begins. The assignments of a human for the given period are concatenated in an ordered list that abstracts from concrete times, but represents the order of the assignments. If no quality overhead factor is specified, there is only time for quality measures if a human finished early on an assignment and no further assignments are delayed. If a quality overhead factor is specified, the CPM framework may insert quality measures according to that factor and thus to postpone assignments in the list, but only within the given period. This approach can be also applied to a multi-project environment utilizing multiple lists of assignments for a particular human, which are then treated separately.

5. Assignment completion: An assignment for a human is finished, triggering the quality effort calculation.

6. Period normalization: The chosen period is normalized to the previously defined interval.

7. Quality effort determination: The absolute number of hours available for quality work in the period is calculated. It is a percentage, defined as quality overhead factor, of the absolute number of planned work hours of a human for this iteration.

8. Current quality effort calculation: Based on the quality distribution function, the number of hours for quality measures at the current time point is calculated.

9. Applied quality overhead subtraction: The effort already spent in quality measures is subtracted.

10. Early completion subtraction: If the human has finished one or multiple assignments early, the remaining time is also separately added.

11. Measure request: A measure is requested from the quality management component.

Example 9-3 illustrates the detection with and without the quality overhead factor.

**Example 9-3 (Quality opportunity detection):**

To illustrate how automated quality opportunity detection can be applied, Figure 9-4 shows a simple schedule of the Company symbolizing a small development iteration of a project. This example, which demonstrates early human assignment completion, assumes that the Company has created fine-grained assignments, not taking weeks but days, to support relatively fine-grained quality measure planning. The schedule comprises four humans having five assignments each. Each of these assignments, in turn, is estimated to take three workdays, each consisting of eight work hours. Every human finishes early on one assignment, triggering the creation of a q-slot filling the hole in the schedule.

![Figure 9-4: Schedule with q-slots and early assignment completion](image)

The second part of this example considers the same iteration, but with a predefined quality overhead factor of 25% instead of a fifth assignment. It is assumed that the same assignments are finished early as in the first part of this example. The defined quality function is shown in Figure 9-5. It uses an interval of [0, 1], with 1 being the maximum time. This overhead relates to the time estimated for the planned assignments (as shown in Figure 9-5 by the integrated assignments). It does not relate to the time estimated for the considered iteration as the predefined quality overhead must also be part of the iteration.
Thus, the planned assignments in the iteration take 12 workdays, while the iteration including quality work takes 15 workdays. As quality distribution function is \( f(x) = 0.75x + 0.25 \) is chosen. As shown in Figure 9-5, compared to the standard function, this function is used to apply a larger amount of the quality effort at the beginning of the iteration.

Table 9-2 shows the calculations of the quality measure distribution, whereas Figure 9-6 shows the final distribution in the considered development iteration. In detail, the calculation is started upon completion of the standard assignments\(^ 3\). Column ‘Q Available’ shows the calculation of the available time for quality measures. In turn, ‘Q Used’ shows the amount of quality time already used and ‘Q early’ reflects additional time for quality measures from early assignment completion. Finally, ‘Q Applied’ shows how much time was used when applying a quality measure.

We explain this calculation using the first assignment of the first human: It takes 25% of the absolute amount of hours of the standard assignments (96 h). The second part (in brackets) is the result of the quality function for the current time point (25% of the assignments have been completed here). This results in the availability of 10.5 hours for quality measures after that assignment. The CPM framework finds a matching quality measure and thus uses eight hours of the available time for that assignment.

\(^3\) It could also be considered that the calculation is also applied after the quality measure assignments. That would result in an even more fine grained distribution.
measure. After the following time further hours are added (if ‘Q Early’ is greater zero) or subtracted (if ‘Q Used’ is greater zero). Note that, for example, after assignment 4 of human 1 there is 1 taken as value for the distribution calculation instead of the real value. This is done because 1 is the maximum representing the time point when the standard assignments would have been completed without quality measure assignments.

<table>
<thead>
<tr>
<th>User 1</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>User 3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>User 4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Figure 9.6: Schedule with additional *quality overhead factor*

### Problem Detection

Problem detection makes use of the environmental awareness of the CPM framework to identify potential problems, e.g., in the source code. In this context, external data from tools needs to be integrated. This information is utilized for calculating various metrics measuring the quality state of the SE project. Metrics directly indicating problems in the source code are obtained from static code analysis tools like PMD [Cope05] and FindBugs [AHM+08], whereas certain testing problems can be detected with test coverage tools [YaWe06] such as Cobertura [Cobe15] or EMMA [EMMA15].

However, not only code-related and product-level problems threaten software quality, but process-related factors should be also assessed to ensure quality. These assessments include functional testing, profiling, and load testing. Since the CPM framework is aware of the enactment of respective activities, it can ascertain their absence. Thus, process metrics can include these facts as well. Facts available to the CPM framework can be incorporated in metrics, which enables quality awareness through the presence of quantifiable information. To reduce the associated configuration effort, a set of pre-configured default metrics will be included with the CPM framework.

After detecting any problem, actions (i.e., quality measures) can be used to counter them. We apply the option to specify triggering rules. Such predefined rules are executed for triggering the automatic proposal of a measure when violating the defined threshold for a particular metric (cf. Example 9.4). Metric or violation reports are received and analyzed, and a list of the violated metrics, together with assigned quality measures, is created.

Example 9.4 (Source code monitoring):
Company policy includes a nightly build process on a build server that invokes static code analysis tools to enable continuous measurement. Thus, a deterioration of the source code quality can be detected by metrics such as cyclomatic complexity. If complexity exceeds a certain threshold, the code becomes difficult to maintain and test, and there is a higher probability of introducing defects.

### 9.2.3. Quality Measure Processing

In fact, automated quality measure processing involves many different factors and decisions that must be incorporated into the CPM quality management integration approach. Quality measures must be automatically attributed to metrics violating thresholds. As the number of such measures is mostly higher as the available time and resources for applying them, they must be carefully prioritized and contextually selected. Different quality goals can be conflicting and may develop differently during the course of a project. Therefore, we apply different collaborating components and integrated technologies. Figure 9.7 illustrates these.
The context management component coordinates all actions of the CPM framework. Therefore, it interacts with the other central components. For enabling automated quality measure processing, this involves the process and quality management components. The former utilizes the WfMS to enact the workflows for such quality measures and enables their seamless integration into the SE process. The quality management component, in turn, utilizes two other components integrating technologies into the CPM framework as already discussed in Chapter 6. The multi-agent system component has been integrated due to its capabilities for efficiently and robustly processing dynamic and potentially conflicting situations (cf. Chapter 6). Therefore, we apply it to quality measure prioritizing in alignment with potentially conflicting quality goals. This component receives an initial measure list from the rules processing component, which has been selected due to its capabilities for the simple processing of standardized recurring tasks (cf. Chapter 6). In this case, it is responsible for initially assigning a quality measure to each metric violating a certain threshold.

**Measure Proposal**

At this point in the procedure, problems as well as q-slots have been detected and an initial assignment of quality measures to metric violations has been made. The generation of a q-slot then triggers a measure selection and proposal process. The latter is coordinated by the context management component. First, this component triggers the quality management component to prioritize the measures strategically in alignment to the quality goals of the SE project. This is done using the multi-agent system component. Thereafter, the measures are tailored to the current situation.

The process of prioritizing measures is rather dynamic due to different goals, various metrics violating certain thresholds and different project situations. Therefore, the quality management component utilizes the multi-agent system component to prioritize measures in alignment with the project goals to be able to accommodate these various factors. The process is based on an extension of the GQM (Goal-Question-Metric) technique [vBCR02]. GQM consists of a hierarchical structure starting with the definition of certain project goals. During configuration, for each goal various questions are defined. These answers should provide indicators for the level of goal fulfillment. Each question can be associated with certain metrics, establishing a connection from the abstract project goals to concrete facts in the project. Example 9-5 shows the application of the GQM technique.

**Example 9-5 (GQM plan):**

As part of a GQM plan, a goal ‘Maintainability’ could be defined relating to the maintainability of the produced source code. An example for a question in the context of this goal is ‘How understandable is the code?’ In turn, for this question, different metrics could apply, e.g., the metric ‘Comment Ratio’ denoting the amount of source code documentation related to the amount of source code.

**Automated Goal-Question-Metric Approach**

This subsection shows the basic concept the measure prioritizing relies on. Two main requirements must be satisfied to facilitate automatic GQM execution support. First, a GQM plan must exist that defines the relations between goals, questions and metrics. Second, the metrics must be integrated in
Automated Quality Management Integration in Software Engineering Processes

the CPM framework, enabling the automatic extraction of corresponding values and thus the automatic receipt of possible deviation information.

Some extensions to the GQM technique became necessary to support automation. Different abstractions of key performance indicators (KPIs) were introduced to enable automatic calculation of goal deviations. Furthermore, metrics are encapsulated in KPIs to consolidate and calculate deviations. Since multiple metrics may be utilized for a single question in GQM, a Question KPI (QKPI) was created for consolidating the metric values at the question level. Similarly, multiple questions may apply to a single goal, thus a Goal KPI (GKPI) is used for calculating goal deviations. For each KPI, formulas specify how metrics are combined. To support automated attainment of multiple goals, we attribute one agent in the multi-agent system component (cf. Chapter 6) to one goal. That way, each goal can be separately and automatically monitored and pursued.

The different KPIs are calculated by the context management component as part of the quality trend analysis. Figure 9-8 shows the relation between the conceptual elements.

![Figure 9-8: Extended GQM structure](image)

To prescribe appropriate countermeasures for (potential) quality deterioration, measures were categorized as follows. Respective measures are *reactive measures* directly associated with concrete metrics or violations, whereas *proactive measures* support certain quality goals at an abstract level and may not be readily associated with a concrete problem. Proactive measures are assigned to goals and can be triggered either when a GKPI deviation occurs or in the absence of reactive measures. This differentiation is pragmatic since reactive measures can be based on concrete existing problems and can thus be more fine-grained, whereas proactive measures support a goal in general.

Procedure 9-5 shows how the lists of quality measures are created:

**Procedure 9-5 (Goal question metric procedure):**

1. **Quality configuration:** At the beginning of a project or a phase or iteration, a quality manager assigns points to each goal (implying its importance) and chooses a strategy for the agent managing that goal. These points are used by agents for negotiating proposed measures.

2. **Report receipt:** Upon creation of a unified report containing violated metric thresholds and related quality measures, the prioritization is triggered. This is done by the agents that incorporate a proactive bidding procedure for proactive measures and a reactive voting procedure used for the reactive measures (this is detailed in the following). In this case, the reactive section is used since the measures received are reactive.

3. **Measure request:** The creation of a q-slot triggers the context management component to start measure selection. Therefore, a measure list from the quality management component is requested that, in turn, utilizes the agents to create the list. The created list may contain either proactive or reactive measures. To define an appropriate proportion between the two, a proactive-to-reactive ratio can be defined. This determines how often reactive vs. proactive measure sets are provided by the agents. For a proactive measure list, the agents start the bidding procedure to create it. In
case of a reactive list, the list created in step two is utilized. If no metrics and, thus, no reactive measures are available yet, no question or goal deviation is detectable since there is no basis for calculating them. In this case, all agents participate in proactive bidding so that any q-slots can be used for proactive measures.

**Measure Prioritization**

Quality goals can be conflicting. Furthermore, determining the appropriate balance between them is project-specific. Thus, we have applied a competitive bidding process among agents for enabling proactive measures, whereas a cooperative voting process is applied for enabling reactive measures. The competitive bidding allows agents with greater importance to definitively have opportunities to support their goal with measures, in contrast to voting where agent majorities might win. That way, a group of lower-priority goal agents does not hinder a higher priority goal from ever asserting influence. The bidding strategies enable agents to win opportunities earlier or later in an iteration cycle.

The reactive voting process is cooperative since a potentially large number of concrete reactive measures based on metric violations become possible for a limited number of quality opportunity slots (q-slots). The measures that will have the greatest overall quality impact across all goals are favored. The agents cooperatively vote on the measures list received from the *rules processing* component (see Example 9-6 for an illustrating scenario concerning this process). Utilizing the structure shown in Figure 9-8, for each measure each agent determines whether a measure belongs to a metric related to the agent’s goal. The points of an agent are then distributed (currently uniformly) across all measures associated to its goal. The output of this process is a prioritized list of reactive measures. The final selection of a measure will then be applied by context management based on the situation.

The section of the multi-agent system component responsible for proactive measures utilizes the received metric values for calculating the different KPIs, QKPIs, and GKPIs. If there are any deviations at the goal level of an agent (relating to its GKPI), the agent may participate in the bidding session (this favors those goals known to be at risk). Each agent bids and the highest bid wins, elevating its proactive measure set to a proposal. In this process, not just the points differentiate between the goals, but also the strategy chosen for the agents. The strategies influence how an agent increases or decreases its bids after winning or losing for the next bidding process. Choosing a defensive strategy for an agent will increase the likelihood that a proposal of its associated measures will occur in later phases of the iteration. This behavior occurs because in early sessions the agents with more aggressive strategies will place much higher bids. The defensive agent can then place winning bids later when the aggressive agents run out of points (see Example 9-6 for an illustrating scenario).

**Example 9-6 (Measure prioritizing):**

Figure 9-9 shows an exemplary scenario for automatic measure prioritizing incorporating three quality goals as well as proactive and reactive measures. First, ‘Maintainability’ with proactive measures like ‘Review Style Guidelines’ and reactive measures like ‘Refactor Code’ is present. Second, ‘Reliability’ with proactive measures like ‘Analyze Error Handling Implementation’ and reactive measures like ‘Refactor Code’ is applied. Third, ‘Performance with proactive measures like ‘Do Profiling’ and reactive measures like ‘Do Performance Testing’ is taken.

Figure 9-9 further illustrates the different steps taken for automatic measure prioritizing: (1) The quality management component receives a list of metrics and reactive measures from the *rules processing* component. (2) The three agents analyze the report for metrics and measures attributed to their goal via the GQM structure (cf. measures R1-8). (3) The agents prioritize the list by voting on their measures. (4) The agents create a prioritized list of proactive measures attributed to their respective goal (cf. measures P1-9). (5) The list gets unified utilizing a pre-defined proactive to reactive ratio to receive a final prioritized measure list.
Measure Tailoring

The multi-agent system component has created a list of prioritized measures according to project goals. However, the finally selected measure should also depend on environmental factors to enable the most effective and efficient measure application. These include properties of the measure itself, properties of the human applying the measure, and other contextual properties.

The properties of the measure are defined in the context management component by the measure template and include, for example, the type of the measure or the number of humans for which the measure is intended (e.g., a code review involves multiple humans). A property of the human being of interest is the skill level. The other contextual properties are modeled based on the concepts of the q-slot and the extension point (see Figure 9-10 for a selection of matching properties). The extension point (cf. Chapter 7) corresponds to a pre-specified point in the workflow where the integration of a new activity is possible. In this case, a special type of the extension point is used that marks a possible insertion point for a quality measure. The extension point involves an abstraction level (e.g., concrete activity, iteration, phase) and an applicable measure type since, for example, at the end of a project phase other measures might be applicable compared to a time point after directly implementing new functionality. The q-slot captures a time category indicating how much time is left for a quality measure. Via these properties, a measure that fits the current situation can be chosen. Figure 9-10 shows a simplified example in which the measure fits the situation.

The extension point determination process is started when an assignment is completed and either enough spare time for applying a quality measure exists or the quality effort has not exceeded the specified quality overhead yet. The CPM framework searches future assignments of the current human, which have already been modeled, for extension points and makes a selection based on different properties. For example, if a human works on a source code artifact for which the CPM framework has detected a code quality problem, an extension point at that activity can be selected. Thus, two changes to that artifact may be applied, whereas the process of checking out, testing, and checking in the artifact is performed only once for efficiency reasons. This scenario is illustrated by Example 9-7.
Example 9-7 (Extension points with problem relation): 
Figure 9-11 illustrates the measure tailoring and extension point selection procedure: it contains workflows on three different levels and shows the related concepts on the context management as well as the process management side.

For the sake of readability, the connecting links between the two sides are omitted in Figure 9-11. The depicted workflows consist of an abstract workflow for a project phase (‘Phase 1’) that contains another workflow for an iteration (using the connection of work unit ‘A2’ to work unit container...
The ‘Iteration 1’ workflow, in turn, contains two concrete workflows. Extension point ‘E1’ is attributed to a work unit of the ‘Phase 1’ work unit container. Thus, a quality activity can thus be inserted after completing work unit ‘A2’ (and the connected iteration work unit container). There are two other extension points in this example, ‘E2’ in the iteration and ‘E3’ being part of a concrete workflow. Various properties can be included in the extension points that act as a filter for the types of measures applicable at a certain point: In this example, the three extension points are differentiated according to their abstraction level. ‘E3’ is attributed to a concrete activity, and the type of that activity (e.g., testing or development) can be stored in a separate property.

In the given example, the human finishes ‘A2’ and the CPM framework detects a q-slot and checks future assignments, i.e. ‘A3’ in the given case. This assignment has an attributed sub-workflow that contains the extension point ‘E3’. That activity involves processing of an artifact for which a problem has been detected as part of the problem detection procedure. That way a quality measure can be chosen in alignment with the type of activity the human has just processed and matching to the concrete problem of the processed artifact.

To enable better distribution of SQA measures over time and to avoid the consideration of quality aspects solely at the end of the project, each extension point will receive a higher weight if it is close to the current time (*imminence*). The context-based measure and extension point selection both depend on weighting the measures. In turn, the latter depends on different factors. First, the imminence $i$ of the extension point is considered. This value is applied in favor of the extension points that are in the near future, since measure application should be done as soon as possible after detecting the opportunity for it ($0 < i \leq 1$; initial value $i=0.9$ for the extension point in the nearest future, for each following extension point 90% of predecessor). A concrete example is given by Figure 9-12C, which shows three exemplary consecutive assignments of a human with associated workflow including the imminence values of the extension points.

Second, the strategic alignment $sa$ of the measure is considered. This property indicates the suitability relating to the goals in the project that has been determined by the measure prioritization (i.e., the position in the ordered list of the multi-agent system component). For a concrete example with a list containing four measures we refer to Figure 9-12B. Third, the measure utility $mu$, which represents the utility or usefulness of applied measures, which is always updated in the measure assessment process.
Automated Quality Management Integration in Software Engineering Processes

(0.5 < \( \mu \) < 1.5; initialized with 1). For details on the utility calculations see Section 9.2.4. For a concrete example see Figure 9-12A. It shows four applied measures. One of them was not successfully applied resulting in a lower measure utility value.

The fourth factor depends on the measure type and permits weighting of certain measure types (\( \text{measure type factor mtf} : 0.5 < \text{mtf} < 1.5 \); predefined, standard value 1). For instance, measure type ‘code’ denotes measures related to source code problems, and, thus, can only be applied if future activities involve artifacts for which problems have been detected (such as activity ‘Ac7’ in Figure 9-11). These measures improve efficiency since they can be applied after planned changes to artifacts, avoiding additional overhead for checking out / in and testing these artifacts. For such measures, a high mtf can be chosen so that they, if they are applicable, have a high chance of being applied. After applying this weighting procedure, the extension point with the highest weighted measure is chosen for insertion. The calculation of measure weight \( mw \) is shown in Formula (1).

\[
mw = i \times sa \times \mu \times mtf
\]

\[
i = 0.9^* 
\]

\[
sa = \left(1 - \frac{\text{listPosition}}{\text{listItems}}\right)
\]

For calculating the imminence, the index for upcoming extension points is \( n \), starting with 0 for the first one. The strategic alignment is computed via the number of measures in the AGQM list (listItems) and the position of each measure in the list (listPosition).

Example 9-8 provides a small scenario illustrating the measure tailoring.

Example 9-8 (Measure tailoring):
For illustrating measure tailoring, we use the OpenUP process to apply extension points. In particular, the ‘Develop Solution Increment’ workflow and its superordinated iteration, the construction iteration are taken. For this iteration, four extension points have been defined. One is placed after the ‘Implement Solution’ assignment activity, having the most concrete abstraction level ‘0’ and measure type ‘code’ (referred to as ‘Code’). The second is placed after the ‘Implement Developer Test’ assignment activity with measure type ‘test’ (‘Test’). Both extension points are directly related to the processed source or test code the human currently works on. Thus they are only taken into account if measures related to the source or test code processed by those activities have been proposed by the quality management component. The third extension point is attributed to the ‘Develop Solution Increment’ workflow (i.e., to its related assignment), enabling the insertion of quality measures between estimated assignments (‘assignment’). The last extension point is assigned to the iteration and used for quality measures only fitting at the end of an iteration (‘Iteration’).

Table 9-3: Q-Slot detection

<table>
<thead>
<tr>
<th>Developer</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>dev1</td>
<td>0</td>
</tr>
<tr>
<td>dev2</td>
<td>1</td>
</tr>
<tr>
<td>dev3</td>
<td>1</td>
</tr>
<tr>
<td>dev4</td>
<td>0</td>
</tr>
<tr>
<td>dev5</td>
<td>1</td>
</tr>
<tr>
<td>dev6</td>
<td>-4</td>
</tr>
<tr>
<td>dev7</td>
<td>0</td>
</tr>
<tr>
<td>dev8</td>
<td>1</td>
</tr>
</tbody>
</table>

In the current scenario, eight developers are involved, each of them having eight estimated ‘Develop Solution Increment’ assignments. For the sake of simplicity, the scenario only relies on execution time
deviation and no specified quality overhead to enable q-slots. Table 9-3 shows the actual execution time deviations from what was estimated; negative values indicate that an activity took longer than estimated, positive values indicate shorter actual execution times. Grey boxes indicate assignments after which the measure proposal process is started for the respective developer. Since the q-slot detection only relies on the execution time deviations, it is independent of the initial duration of the assignments or the work hours per day. To keep the current scenario simple, it is assumed that all assignments take one workday. As the table shows, some assignments take longer, while others finish earlier. For this scenario, the quality measures consuming the least time take two hours, which makes quality measures possible for developers 2, 3, 5, and 7.

Table 9-4 illustrates future extension points and their values for each human at the point of time the measure proposal is started for the first time. In turn, the shaded cells show which extension point was selected based on the highest weight. For human ‘dev2’, this happens at the end of assignment 3 at which measure ‘m1’ is selected. The value of 0.7 is computed from the following values: i=1, mu=1, mtf=1 and sa=0.7 (the measure being at position 30 in a list of 100 proposed measures from the multi-agent system component). All other ‘Activity’ extension points have the same values except for Imminence, which decreases with each additional activity. For the ‘Iteration’ extension point, the same values apply except that it has a strategic alignment of 0.99 and a type factor of 1.2. For human ‘dev2’, it is assumed that it was detected that he will process a source code artifact for which a problem has been detected as part of activity 4. Therefore, measure ‘m2’ is chosen for the ‘Code’ extension point of activity 4. The measure has a strategic alignment of 0.75 and a higher type factor of 1.4. Therefore, it is selected for the human. For human ‘dev3’, the calculation starts after activity 5. Since measures ‘m1’ and ‘m3’ have not been proposed yet, they can still be proposed due to the same parameters. Thus, at the end of the iteration, ‘m3’ has the higher weighting and is applied for that human.

<table>
<thead>
<tr>
<th>Dev</th>
<th>Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>dev2</td>
<td>type</td>
</tr>
<tr>
<td></td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>m</td>
</tr>
<tr>
<td></td>
<td>w</td>
</tr>
<tr>
<td>dev3</td>
<td>type</td>
</tr>
<tr>
<td></td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>m</td>
</tr>
<tr>
<td></td>
<td>w</td>
</tr>
<tr>
<td>dev5</td>
<td>type</td>
</tr>
<tr>
<td></td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>m</td>
</tr>
<tr>
<td></td>
<td>w</td>
</tr>
<tr>
<td>dev7</td>
<td>type</td>
</tr>
<tr>
<td></td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>m</td>
</tr>
<tr>
<td></td>
<td>w</td>
</tr>
</tbody>
</table>

A=Activity, C=Code, I=Iteration, a=assignment, m=measure, w=weight, m1=analyze resource usage, m2=refactor code, m3=verify documentation, m4=code inspection preparation

When starting the calculation for human ‘dev5’ at the end of activity 6, there is the q-slot of human ‘dev3’ that has been planned, but has not been used yet. Therefore, two connected q-slots are available, causing measure ‘m4’ to be proposed. The latter is assumed to be a multi-user measure having a higher strategic alignment of 0.85. Due to the higher weight, measure ‘m4’ is integrated for humans ‘dev3’ and ‘dev5’. When starting the calculation for human ‘dev7’ at the end of activity 7, therefore, measure ‘m3’ for the ‘Iteration’ extension point is still available and is used for the human having the highest weight now.
Measure Application

To enable a high degree of automated guidance, the human is not only informed about the measure to be applied, but the measure is also directly integrated into the human’s workflow. Example 9-9 roughly sketches a scenario in which a quality measure is distributed automatically by the CPM framework.

Example 9-9 (Measure application):
To enable automated support for quality measures, the Company introduced the facilities for automated problem and quality opportunity detection. A GQM plan was created with maintainability and reliability of the code as well as the creation of new functionality as goals. If developers now finish early on assignments, the CPM framework can automatically assign them quality measures that fit the project goals and that are appropriate to the personal situation. As an example consider refactorings of complex code. This measure was triggered as problems in the source code were detected, for example, based on the cyclomatic complexity metric. The measure was prioritized as high since it was judged as important and applicable to both the maintainability and reliability goals of the project. Finally, the CPM framework can choose the matching human for the measure based on properties like the skills of the human or the amount of time they can spend on accomplishing a measure in relation to the expected time needed for the measure.

To enable the automated integration and application of a quality measure, new items must be inserted both on the context and the process management side. This is achieved by Procedure 9-6. This procedure is applied when a quality measure shall be concretely integrated into a running workflow instance.

Procedure 9-6 (Quality measure insertion):
1. Integration event: The procedure starts with the context management component receiving an event for measure integration. At that time, a q-slot has been detected, the measure to be integrated has been selected, and the concrete human, workflow instance and extension point have been determined.
2. Create concepts: The context management component creates the necessary individual concepts: A quality measure is inserted as a new assignment with a set of assignment activities and a separate work unit container with work units. These are created from template concepts that are connected to the measure template. To be able to insert the quality measure at the specified point, a new work unit is created and inserted there, which is then connected to the newly created work unit container belonging to the quality measure (cf. Example 9-10).
3. Prepare workflow: As the horizontal workflow governance (cf. Chapter 7) is managed by the process management component, the workflow instance must be adapted to integrate the quality measure at the desired point. Same as for the context management side, the measure is inserted as an activity that depends on a new workflow instance governing the measure execution. The connection of the activity to the workflow instance is managed in the context management component (cf. Chapter 7) and, thus, on the process management side only the insertion of a new activity in the running workflow instance is required. The insertion is not done immediately as it is possible that in the target workflow instance an activity is active at the moment and, in most WfMS, an adaptation is not feasible when an activity is currently processed. The reason for this is the fact that it is common that WfMS do not permit changes to running workflow instances if an activity is currently active (if they even permit such changes). If an activity is running, therefore, the workflow instance is just prepared for the insertion. If no activity is running, the procedure continues with Step 5.
4. Activity completion event: An event is received denoting the completion of the current activity in the target workflow instance.
5. Workflow adaptation: The new activity is inserted into the target workflow instance.
6. Measure execution: When the execution reaches the newly inserted activity, a new workflow instance is started for the measure, utilizing the connection between the new work unit and the measure work unit container managed in the context management component.
Example 9-10 illustrates the measure integration procedure in a concrete case.

Example 9-10 (Measure integration):
To illustrate measure integration, we utilize the ‘Develop Solution Increment’ workflow of the OpenUP process as the chosen target workflow. It is assumed that a measure ‘Increase Code Coverage’ was chosen, whose goal is to increase the share of the source code that is covered by tests. That measure was chosen to be integrated right after the ‘Implement Developer Tests’ activity of the target workflow instance as the properties of that activity match the properties of the selected measure (and indicated by the relating extension point). Figure 9-13 depicts the used concepts. It shows the ‘Develop Solution Increment’ workflow instance as well as its contextual extensions and the workflow instance for the quality measure. For the sake of readability, most of the connections between the two components and the additional contextual concepts are omitted here. As depicted, the process management component manages the concrete insertion point for the measure, while the context management component manages the connection between the new activity and the measure workflow instance as well as the human assignment. For the sake of readability, the contextual annotation links are only shown for the new activity and workflow instance that realize the quality measure. The same applies to the contextual extensions, as the assignments and assignment activities are only shown for the quality measure work unit container.

![Figure 9-13: Measure integration](image)

**Quality Trend Analysis**

The continuous monitoring of the source code quality is essential to be able to detect any impact of applied quality measures. Therefore, the list of quality measures created with the rule processing component is utilized by the context management component. The list not only contains proposed measures, but also the metric belonging to each measure and its value. To enable automated evaluation of quality trends in conjunction with the GQM technique, different levels of key performance indicators (KPIs) were introduced as depicted in Figure 9-8.
KPIs are composite metrics unifying the values of other metrics or KPIs to enhance their expressiveness and significance. KPIs are not only used for quality trend analysis on different levels of abstraction, but also for automated goal deviation monitoring with respect to the GQM technique. Therefore, three levels of KPIs were introduced. On the most concrete level, the KPI unifies one or more metrics since different metrics may be utilized by the CPM framework. The QKPI represents a Question of the GQM technique as a value to facilitate automated deviation calculation, which is automatically computed from attributed KPIs and base metrics. The same applies for the GKPI, which unifies the values of the questions belonging to one project goal. Procedure 9-7 elucidates the calculation of the KPIs.

Procedure 9-7 (Quality trend analysis):
1. Model GQM structure: One or multiple GQM structures must be defined.
2. Initiate project: When a project is initiated, certain quality goals are assigned to it. At that point, the concrete GQM structure is initiated using the template structure.
3. Receive Report: Each time a unified metric violation report is received by the context management component, the concrete KPI calculation is triggered.
4. Normalize values: For the KPI calculation, the received metric values must be normalized to enable uniform KPI calculation.
5. Calculate KPIs: The complete KPI structure is calculated for all KPIs with metric values in the current report.
6. Display quality trends: The CPM framework can provide these values to the humans to illustrate quality trends.

Figure 9-14 illustrates the quality trend analysis. It shows a schedule with different consecutive assignments. Between these assignments, quality measures are inserted. At defined time points (e.g., every night as part of a nightly build process), measurements are conducted. At each of these time points, a new value is recorded for each metric. Utilizing the GQM structure, each time, a new value for KPIs and quality goals is also calculated. Quality trends can be observed by the deltas of the KPIs and goals.

The calculations are accomplished in a uniform way for all KPIs, applying a weighted average of all values a KPI aggregates as depicted in Formula (2). There, $M_i$ represents the concrete values to be aggregated and $W_i$ is the attributed weight of each of the $n$ metrics or KPIs being aggregated. All received metric values are normalized to a range from 0 to 1.
The quality trend analysis is conducted in the context management component. As for most CPM concepts, all quality management concepts comprise a template for the definition and an individual concept with concrete values. To enable a uniform KPI calculation, all received metric values are normalized to values between 0 and 1, where 1 corresponds to the best possible value and 0 to the worst possible one. Therefore, as part of the template, there is a maximum defined for each metric. The actual value is divided through this maximum to derive a value between 0 and 1. It further defines a limit for the value, e.g., if 15 has been defined as the maximum for cyclomatic complexity, this will be the worst possible value. If the actual values had exceeded this limit, then 15 would be taken instead.

If a metric value is not available, the calculation will be done without that value. For some metrics, the absence of a value also constitutes a negative indicator and, thus, a standard value can be defined in the template. If, for example, a metric had indicated the degree of functional testing compliance (a measure for the outcome of functional testing), its absence would indicate that no functional testing has been done yet. Since that fact should not be overlooked, a standard value can be defined.

Further, it is possible to integrate values from external tools as KPIs. This case can be indicated using a property 'external'. The KPI calculation is a weighted average and therefore a weight used for that KPI is defined as part of a template as well (cf. Section 9.2.5).

### 9.2.4. Quality Post-Processing

When automatically distributing software quality measures another challenge concerns their usefulness and applicability. The same measure might not be equally effective in different companies with different humans, tools, and processes. Therefore, there needs to be some kind of assessment of the applied measures. This section shows what actions are taken by the CPM framework after applying the quality measures.

**Measure Assessment**

Regarding various companies with different humans, tools and processes, applied measures may show different degrees of effectiveness. To reflect this as well as to improve future measure proposals, a *measure utility* is introduced that allows indicating the usefulness of the applied measure (cf. Example 9-11). That usefulness is neutrally initialized and updated after each application of the measure (cf. Figure 9-14). Therefore, the delta of the KPI related to this measure is used. Since some measures may not have an immediate effect, multiple future deltas of the KPI can be taken into account.

**Example 9-11 (Measure assessment):**
The special refactoring proposed by Example 9-9 can now be automatically assessed for the Company since it has applied continuous quality measurement. If the refactoring is successful and the complexity of the code is reduced, this will be indicated by a subsequent measurement showing a lower value for the cyclomatic complexity metric. This value will then affect the value of a KPI related to the maintainability goal defined by the Company. The KPI value, in turn, will affect the utility value of the proposed refactoring measure. That way, a successfully applied measure will have a higher probability of selection by the context management component in the future.

The concrete calculation of the *measure utility* works as follows: The time of the measure execution is associated with the impact on KPI trends for utility analysis. However, the application of a quality
measure might not have an immediate effect. Therefore, from the point of time a measure is applied, multiple KPI deltas are selected for trend analysis (cf. Figure 9-14). As a starting point we take 10 KPI deltas but that number can be configured. Since a multitude of factors can influence evolution of the KPIs, and since the measures can be of diverse nature, all 10 values of each KPI are used with decreasing influence. The skill level of the human involved is also considered, reflecting the higher probability of ineffectively applying a measure by less skilled humans (Skill Factor $sf$: $0 < sf < 1$; predefined, 1 for highest skill level). The equation for calculating $mu$ is shown in Formula (3) where $n$ is the index for the KPI deltas starting with 0.

$$mu = mu \ast \left(1 + \sum_{n}^{9} kpi_{\Delta n} \ast \left[1 - \frac{n}{10}\right] \ast sf\left(person(m_{n})\right)\right)$$

Example 9-12 illustrates concrete calculations along an example scenario.

Example 9-12 (Measure assessment calculation):
To illustrate the measure assessment calculation we use the values of Example 9-8 where tailored quality measures had been applied. At the end of the iteration, after all measures have been applied successfully, the assessment process is carried out by the CPM framework. This process relies on the development of the KPIs that, in turn, rely on received reports from analysis tools. The reports are generated as part of a nightly build process that builds the code, executes all tests, and applies all metrics to the code in the current scenario. Thus, in this scenario incorporating ten work days, there are ten points where the KPIs are calculated, each taking place after completing one of the estimated assignments. Table 9-5 illustrates the development of the related KPIs, which are computed by the context management component to have a uniform value between 0 (extremely bad) and 1 (perfect).

<table>
<thead>
<tr>
<th>Measure</th>
<th>KPI</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>m2</td>
<td>kpi2</td>
<td>0.5</td>
<td>0.64</td>
<td>0.45</td>
<td>0.6</td>
<td>0.6</td>
<td>0.62</td>
<td>0.62</td>
<td></td>
</tr>
<tr>
<td>m4</td>
<td>kpi4</td>
<td>0.6</td>
<td>0.62</td>
<td>0.6</td>
<td>0.58</td>
<td>0.59</td>
<td>0.6</td>
<td>0.67</td>
<td>0.74</td>
</tr>
<tr>
<td>m3</td>
<td>kpi3</td>
<td>0.4</td>
<td>0.43</td>
<td>0.42</td>
<td>0.45</td>
<td>0.46</td>
<td>0.46</td>
<td>0.46</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Since this scenario only covers one iteration assuming no activities have been executed prior to it, not all time points were available for the measure assessment procedure. Measure ‘m2’ was applied at the end of time point 4 but before KPI calculation. Therefore, the first reports of the analysis tools that reflect the impact of this measure are generated at time point 4. Thus, the first delta of the related KPI being of interest is from time point 3 to time point 4, which is a positive change of 0.15. For measure ‘m4’, the first delta used is from time point 5 to 6 and for measure ‘m3’ from time point 7 to 8. Since all measures were initialized with a value of 1 for measure utility, applying the calculation depicted in Formula (3), the new values for the measures are as follows: 1.1662 for measure ‘m2’, 1.154 for ‘m4’, and 1.14 for ‘m3’.

**9.2.5. Conceptual Framework**

This section elaborates on the conceptual framework for integrating quality management into CPM. Therefore, we show the different concepts in Figure 9-15 and briefly describe their interplay afterwards.

As aforementioned, all concepts are separated into template concepts on one hand and individual concepts on the other. There are concepts for the GQM goals and questions unified under the GQM template / individual. The concrete goals are connected to a project concept to define the concrete goals for each project (this can be predefined with the template concepts). Both metric and KPI are united under the concept of the quality indicator. In turn, the latter is connected to the GQM individual
to be used as concrete calculation values for questions and goals (again, these structures can be predefined using the templates). In addition, there is an abstract list concept having two concrete sub-concepts: The violation list containing metrics and KPIs on one hand, and the measure list containing measures on the other. When a violation list containing multiple metrics is received, it will be determined which KPI can be calculated via the KPI template and the metric template. For the computed values, new KPIs are then created. The measure list contains prioritized measures obtained from the multi-agent system component. For measure distribution the q-slot is used. Therefore, a selected q-slot is connected with a measure. The measure, in turn, is one of the concrete sub-concepts of the extension and can therefore be matched to an extension point (cf. Chapter 7) to realize integration of the assignment relating to the measure into the humans workflow.

Example 9-13 (Quality trend analysis):
Figure 9-16 shows an excerpt of a hierarchy of concepts used to model the quality goal ‘Reliability’ and related metrics and measures. For the sake of readability not all connections are shown here. It shows the abstractly defined quality goal (goal template) that has two defined questions dealing with complexity and defect density of the source code. These concepts have associated KPIs and metrics. In
the example, the defect density QKPI only has one associated metric, which is the defect density metric that can be measured using the lines of code compared to the number of defects in the code. The QKPI for complexity has two metrics: cyclomatic complexity (CYC) [McCa76] and Npath complexity (NPA) [Nejm98]. Furthermore, there are predefined measures for the metrics, one of them shown in Figure 9-16: the refactoring (measure template). In turn, these abstract values are used to define the structure. When a project is initiated, concepts are created to hold the concrete values for the project. These are based on the abstract ones and can be seen on the right side of the figure. The metric concepts, for example, hold the concrete normalized measurement values and the concrete measure concept holds information on a concrete application of a measure.

Figure 9-16: Quality management concepts example

9.3. Discussion

This section provides a discussion of work related to SE quality management. There exists other work concerning quality management and processes. An example is business process quality management [LoRe13] and process improvement with process patterns [LoRe15]. However, the approach presented in this chapter focuses on software quality management and its automated support. There have been several different approaches covering different aspects like measurement or analysis. An overview of respective approaches is given in [CSS09]. The following sub sections present a selection of approaches directly related to the concepts developed in this section. To the best of our knowledge, there is no approach directly comparable to one presented. Therefore, the discussion of related work is subdivided into different sections dealing with metric application in SE projects, tools or systems for software measurement, and support for the GQM technique we have applied.
9 Automated Quality Management Integration in Software Engineering Processes

9.3.1. Metric Application

We first consider work related to the application of SE metrics in software projects. [Dask92] provides a report on the implementation of a software measurement program at Motorola. A set of different views on metrics to support their successful application is described, as well as success in several areas by using software metrics. In [OfJe97], a formal meta-model for measurement in SE is established. Strong focus is put on storing, interpreting and analyzing gathered data. Furthermore, a practical framework is offered that supports creating models for software measurement, connecting these to measurement tools, and storing and browsing the results. [GKM02] presents a comprehensive industry-wide survey on the success of metrics programs. That success is assessed by two main factors: use of metrics information in decision-making and improved organizational performance, with various success factors categorized as technical or organizational variables.

These approaches describe how to successfully implement metrics programs, how to organize metrics, and how to interpret them. However, they neither describe actions to be taken on account of these metrics, nor do they provide a concept or infrastructure to automatically triggering counter-measures against detected problems.

9.3.2. Measurement Tools

This section covers practical measurement tools that enable the detection of various problems and properties of SE. The tool PR-Miner can be used to automatically analyze source code and to efficiently and automatically extract implicit, undocumented programming rules from it [LiZh05]. It automatically detects violations to these rules as well. The tool has been tested on various large code bases and proven to be efficient in extracting violations. In [OYS04], an empirical project monitor (EPM) is proposed that aims at supporting effective SE process management by providing quantitative data. To this end, it collects and measures data from different repositories within SE support systems, as for example, bug trackers or configuration management systems. This information is then presented graphically to assist humans in project progress awareness. EPM is a partial implementation of a framework called Empirical Software Engineering Environment (ESEE) that shall unite a variety of tools to support automated software process improvement. ElectroCodeoGram (ECG) is a modular framework to collect and summarize data about programming behavior [ScJe06]. It consists of different sensors to acquire data and other modules used to interpret that data. The framework aims at providing micro-process data to aid researchers in understanding how programming is carried out on a fine grained level. A framework having similar goals is SUMS (Standard User Monitoring Suite) [NU05]. It implements data acquisition for programming in various languages. That data is stored and aggregated in a database and automatically analyzed using different components like neural networks and Bayesian analysis to achieve automated learning features.

The presented tools provide a means of automatically acquiring data from SE projects and to interpret this data. Yet, they are limited to primarily visualizing or analyzing data and do not aim at potential problem resolution.

9.3.3. GQM support

The combination of GQM with agents has been used for providing automated support for GQM plan creation [CHW03, HuFa05, FaWu09] and for computing values for questions and goals [SST05, STS05]. [FaWu09] utilizes a goal-driven use case method to elicit requirements. A set of agents assists the human in identifying goals and questions, which are then used by another agent to obtain metrics. The collection of the measurement data and the creation of the measurement plan are then executed by two other agents. The ISMS (Intelligent Software Measurement System) [CHW03, HuFa05] follows a similar approach, using different groups of agents for human assistance and determination of different areas of the GQM plan. In [SST05, STS05], agents are used in the requirements process of the SW-CMM (Software Capability Maturity Model) model. The focus is the measurement and analysis of SE processes using agents and fuzzy logic.
The approach presented in [Lava00] aims at automated human assistance in GQM plan creation and execution, but does not utilize agent technology. A tool was developed which allows creating GQM plans that follow predefined forms, verifying the structural consistency of the plan, and supporting the reuse of plan components. Furthermore, the tool supports data interpretation and analysis through aggregation of collected data. This approach is extended in [LavBa05], which integrates GQM more tightly with a SE process to support GQM plan creation by an explicit process model.

For tighter integration of the GQM technique into the project flow via automation, different approaches were considered. [LoRo93] aims at integrating measurement programs as well as data collection into explicit process models, while [Mori99] provides an object-oriented process model whose target is measurement. [Broe96] proposes the usage of process models for creating GQM plans. Finally, the tool Prometheus [Visa94] links executive plans with process models.

[dKM97] extends the GQM technique by adding concepts such as entities, attributes, and units. cGQM [Lofi05] proposes the use of the Hackystat framework for GQM, applying continuous measurement with short feedback loops.

As opposed to the aforementioned approaches, this work’s AGQM process integrates its techniques into live software engineering environments, actively injecting software quality measure proposals as guidance for developers. Agent technology is used differently, as the aim is neither human assistance in GQM plan creation nor assistance in interpreting measurement results. It is rather the fully automatic monitoring of goal fulfillment and the automatic assignment of quality measures for different types of quality deviations.

### 9.4. Summary

SQA should be aligned to the actual SE process and be applicable at the operational level. The manual combination of SQA with SE process management requires constant vigilance and associated labor in order to avoid missing quality opportunities and to continuously monitor quality goal states. Automated quality guidance support could assist developers by providing SQA triggering based on actual situational data, continuously monitoring quality goal states and trends, and selecting and tailoring software quality measures to being most appropriate in the current situation.

This chapter presented an automated solution to support the integration of process management and quality assurance in SE. This was enabled by combining contextual awareness, automated and adequate quality measure selection, and seamless integration of quality measures into the SE process.

The first thing that must be in place to enable automated proposal of quality measures in SE projects is contextual awareness. This concerns awareness of existing problems, as without knowledge of concrete problems concerning the produced product, countermeasures cannot be taken. Our solution creates such an awareness mainly utilizing sensors applied into static code analysis tools and sensors in tools or manual information by humans. Our solution further enables the distribution of quality measures not interfering with the standard development schedule. It can dynamically utilize free time slots as they may occur in SE projects due to various reasons. Furthermore, time for quality measures can be allocated prior to execution. That way, a balanced distribution of quality measures within a project becomes possible.

The second crucial factor for successful automated quality measure distribution is the quality of the measure selection. In our solution, measures are strategically prioritized according to the quality goals of a project. That way, it becomes possible to define certain quality goals for a project and weight them accordingly. These goals can be supported by reactive and proactive quality measures. The former are applied to counteract concrete problems with the source code while the latter may even prevent such problems. Another important factor is the current situation in which the measure shall be applied: It must match the properties of the situation, like the applying human, the amount of time available, or the point in the process. Our solution chooses a measure whose properties fit the situation...
for effective application. However, different companies, different cultures, or different organization structures could greatly influence the effectiveness of certain types of measures. Therefore, this solution assesses the effectiveness of measures after their application and records that for future use.

The third component having great influence on successful automatic measure distribution is the type of application of these measures. In many companies, quality management and the standard SE process may interfere with each other. This is avoided in our solution, as quality measures are supported with the same process management technology as standard process enactment and seamlessly integrated into humans’ workflows. This also enhances and unifies traceability concerning the execution of bug fixes. Furthermore, it seamlessly integrates the measures into the humans’ workflows without requiring them to use different tools or disturbing them in any other way.

By incorporating the various aforementioned factors, the approach we developed shows promise in aiding quality management in SE and in promoting the automation of the latter.
10. Workflow Coordination in Software Engineering Processes

Software development is a collaborative process. Hence, the communication and coordination of various humans is crucial for project success [JYW07, CoCh06]. In multi-project and multi-team environments, this can be challenging, as there are many information channels and dependencies between humans. In particular, the connections between humans, teams, tools, and artifacts are very important for SE [JYW07, SQTR07]. Nowadays, in many projects, much of the required coordination effort is handled manually, as collaborative work in SE is still not adequately supported by tools [LeBo07]. Efficient communication and process-aware collaboration constitute a great challenge [Dust04] and team work remains to a large extend unplannable and unpredictable [BSV07]. In particular, this often results in high additional effort for each human. Furthermore, it can involve missing coordination activities and thus even endangering project success.

For these reasons, automated coordination and collaboration support in SE processes is required. To enhance the automated coordination capabilities in software engineering environments (SEEs), various challenges must be tackled. First and foremost, SE is project-oriented and lacks the typical production stage with repeatable activities or interactions. Besides this, Process-Centered Software Engineering Environments (PCSEEs) [Gruh02] support SE projects with both tooling and processes, yet these need to be tailored to the unique and diverse project and product needs (e.g., quality levels or team size). Although common SE process models have proven to be beneficial [BWHW06, RiJa00, Mall09], they are manually implemented, especially in small- and medium-sized enterprises. Furthermore, the models are too coarse-grained and documented too generally and rely on humans to follow and map actual low-level concrete actions and events to the appropriate higher-level process. In the following, we denote such connection of an abstract process to concrete activities process navigability. For other perspectives on process navigability and navigation, we refer to [HMR12, HMMR14].

To comprehensively support the SE process, various other aspects need to be considered as well: Collaboration activities need to be triggered and orchestrated automatically. To enable configurable collaboration support, different kinds of dependencies between activities should be considered. For example, some activities might necessitate other activities to directly follow them. We denote such activities follow-up activities. In other cases, notifying other team members about a change might suffice. Such additional activities are executed in addition to the planned SE process activities and, therefore, belong to the formerly introduced category of extrinsic activities. However, the latter should be traceable as well. Therefore, they need to be connected with the intrinsic activities that initiated them, so that it can be reconstructed why they were executed. To support context-awareness of humans, automated guidance should not only be provided for the activities in a specific workflow (horizontal connections between the activities as introduced in Chapter 7), but also vertically, making the hierarchical connections between processes and workflows explicit. This chapter illustrates coordination support for SE. This incorporates the following features:

- Process navigability information: Humans working in multi-project environments are supported by the automatic provisioning of extended information about the currently processed activity as well as the superordinate levels (e.g., the current project phase).

---

1 This chapter is partially based on the publications [GOR11c] and [GOR12a].
10 Workflow Coordination in Software Engineering Processes

- Passive coordination support: Automatic information distribution is enabled to inform humans about events in a project. For example, this includes information about artifact changes as well as the activities of other humans.
- Active coordination support: Automatic initiation and governance of follow-up activities required by certain actions is provided.

10.1. Requirements

This section elicits requirements for coordination and collaboration support in SE projects using a practical example. The addressed issues will be illustrated along two problem areas: The horizontal connection of intrinsic and extrinsic activities as well as the vertical connection of abstract processes and concrete workflows. Usually, extrinsic workflows involve activities triggered by intrinsic activities and, thus, being dependent on them. Example 10-1 illustrates the various dependencies between activities.

Example 10-1 (Coordination shortcomings):
The Company is a growing SME with increasing demands for coordinating their SE workflows (cf. Example 4-3). The team sizes and number of parallel projects grow. Employees often must switch between projects and concurrently work on the artifact bases of the latter.

A particular problem is related to frequent project switching. For example, software developers often take part in different projects in parallel. Such projects may have different properties. For example, a particular developer may participate in a project, in which a new software is developed while being also a member of a project extending another software of The Company. In addition, he might participate in a project where he must provide support for a legacy application not being developed any further. A human that often switches between projects must manually and repeatedly gather context information on each project to work effectively: Which assignment must be processed for which project? What are potential milestones and deadlines? What is the state of the currently processed assignment? What are upcoming activities to complete it?

Another problem relates to cooperative work on the same artifact base. It might be an issue that activities and their changes on artifacts will often remain unnoticed by others. For example, if two teams (e.g., a development team and a test team) are working on the same source code artifacts, they need to be informed about changes of the SE artifacts. Usually, respective information is transferred manually and is therefore prone to the forgetfulness of the involved humans.

A third problem is directly related to the artifacts and their relations: Changes of one artifact might have an effect on other artifacts. For example, the change of specific source code artifacts might have an impact on specification artifacts or test code artifacts. The latter might have to be adapted in separate activities by humans. In turn, these tasks might belong to other project areas or departments, have different responsible humans, or be not communicated, and thus never happen. As a consequence, inconsistencies between related artifacts may increase over time.

To enable the CPM framework to provide assistance in solving the issues illustrated by Example 10-1, basic requirements must be satisfied (cf. Chapter 4). The latter include basic automation (cf. \textit{R:AutoProc}) as well as the integration of the SE process and contextual data (cf. \textit{R:ContInt}), and have been already covered in Chapter 7. We now elicit additional requirements related to coordination as refinement of the requirement \textit{R:Coord} (cf. Chapter 4).

- \textit{Requirement R:Coord:InfDist (Autonomic information distribution):} The framework shall allow for the automated distribution of information about activities and their effects on artifacts like source code or specification documents. These notifications shall be automatically distributed to the concerned users and be seamlessly presented to them in order
to not distract them from their present activity. It shall be flexibly configurable what causes such notifications for whom.

- **Requirement R:Coord:Navi (Navigability information provision):** The framework shall provide the users with enhanced situational information according to the current process state. This should incorporate information about the activity the humans currently processes as well as the more abstract areas the activity relates to (e.g., the current project and the state of its process).

- **Requirement R:Coord:FolAct (Automatic follow-up activities):** The framework shall be able to automatically initiate follow-up activities to cover changes that become necessary on account of the changes other activities have made. These follow-up activities shall be automatically distributed to the responsible humans. In particular, this involves capabilities to automatically detect what effects certain activities may have on artifacts. Furthermore, the framework shall be able to automatically determine the right human for executing the follow-up activities.

- **Requirement R:Coord:ActConfig (Configuration for follow-up activities):** The framework shall enable configuration of the way follow-up activities are provided. Users shall not be overwhelmed with numerous rather small activities (as, e.g., small changes to one document). Activities of the same type shall be groupable. Thresholds shall be in place for the number of follow-up activities per user to enable the collection and consolidated provisioning of the activities. That way, a human could get one activity comprising a number of small changes to different documents instead of receiving an activity for each of these changes.

The requirements are summarized in Table 10-1.

<table>
<thead>
<tr>
<th>Req ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R:Coord</td>
<td>Coordination Support</td>
<td>The CPM framework shall provide automated support for coordination and collaboration.</td>
</tr>
<tr>
<td>R:Coord:InfDist</td>
<td>Autonomic Information Distribution</td>
<td>The CPM framework shall enable automatic notifications about activities and activity effects.</td>
</tr>
<tr>
<td>R:Coord:Navi</td>
<td>Navigability Information Provision</td>
<td>The CPM framework shall be able to provide the user with enhanced situational information according to the current state of the process.</td>
</tr>
<tr>
<td>R:Coord:FolAct</td>
<td>Automatic follow-up activities</td>
<td>The CPM framework shall be able to automatically initiate follow-up activities.</td>
</tr>
<tr>
<td>R:Coord:ActConfig</td>
<td>Configuration for follow-up activities</td>
<td>The CPM framework shall enable configuration of the way follow-up activities are provided.</td>
</tr>
</tbody>
</table>

### 10.2. Automatic Activity Coordination

This section presents an approach that addresses the aforementioned requirements. Automatic coordination is achieved utilizing the context and process management components. An approach is presented, separating coordination into passive and active coordination support. Passive support comprises the actions providing humans with additional information, whereas active support relates to activities the CPM framework initiates for its users.

#### 10.2.1. Passive Coordination Support

In many situations, SE project participants can profit from automatically delivered information concerning their colleagues and environment. Passive coordination support therefore provides process navigability information and automatic change notifications (cf. R:Coord:InfDist and R:Coord:Navi). First, the navigability information approach is discussed. The contextual extension to workflows provided by the context management component already provides humans with additional information. This comprises the current user assignment; i.e., the assignment activity, activity steps, current task, and activity group the current activity belongs to (cf. Chapter 7). Furthermore, by
mapping workflow variables to user decisions (cf. Chapter 7), the human has a simple facility to influence his workflow being aware of forthcoming activities. Another type of information relates to the more abstract process areas like phases of a project. As the latter are connected to the concrete workflows, the human can also receive information about them. This includes information about the current project or its milestones, which are modeled in the context management component as well (cf. Chapter 7). Example 10-2 illustrates this using the OpenUP process.

Example 10-2 (Process navigability):
A human switching between two projects can benefit from navigability information. In one project, he deals with requirements elicitation and processes the OpenUP ‘Identify and Refine Requirements’ workflow. In the other project, he develops software processing the ‘Develop Solution Increment’ workflow. Figure 10-1 shows both workflows as well as their mapping in the context management component.

As shown in Figure 10-1, there exists additional information about the activity specified in the OpenUp process. In turn, this information can be modeled in the CPM framework based on the
There are supportive activity steps (as e.g., “Gather Information”), a discipline for the activity (e.g., “Requirements”), the currently processed task (e.g., “Coding”) and the specific user assignment (as ‘Develop Feature X’). Additionally, the specific project (e.g., ‘Project A’) and its milestones from the OpenUP process (e.g., ‘Initial Operational Capability’) are included. Finally, when mapping workflow variables to user decisions (cf. Chapter 7), the human can directly select, which activity he wants to process next.

The second passive coordination ability provided by the framework deals with configurable user notifications. To support collaborating humans as well as to reduce omissive errors, automatic notifications may be beneficial mainly for two situations in SE projects: First, situations, in which events happen that relate to activities or artifacts and, second, when their status changes. Therefore, several concepts in the context management component are involved as shown in Figure 10-2.

To enable notifications for the presented scenarios, concepts for event and status are utilized. The event concept primarily relates to events occurring in the context of a SE project and being automatically detected by the event management component. The status concept (cf. Chapter 7) has been introduced to explicitly model the status of various other concepts like assignments or artifacts. In an SE project, various artifacts exist with different relations belonging to different project areas. Examples include requirements specifications or source code artifacts. To be able to explicitly describe such artifacts in the context management component, the project component has been introduced (cf. Chapter 7, see Figure 10-2 for an illustration). The latter serves as an abstract building block for structuring a project. Specializations of this concept are artifacts and sections. The latter are used for structuring and the former for directly modeling existing artifacts in the project. As example consider a source code structure where the sections depict the source code packages. User management includes concepts for roles, persons and teams (cf. Figure 10-2). A role can be used as placeholder for a person when it is not yet known who shall execute the activity. Roles can also be used in relation to project components to express, e.g., that a person is responsible for a certain source code package. Persons and teams are unified under the abstract resource concept to enable the assignment of activities to teams as well as single persons.

Utilizing the aforementioned concepts, the notification template can be used to configure user notifications relating to various events and status changes of various entities in a SE project. A notification can be created when a particular event occurs relating to such an entity or when it enters a particular state. It further allows defining who shall receive the notification. It can be configured for a role template, for pre-configured general notifications, for a concrete role in a project, or for a concrete resource (i.e., a person or a team). If a team is configured, the team leader will receive the notification. In turn, the latter is generated when the trigger defined in the notification template applies. Practical notifications are illustrated in Example 10-3.

Example 10-3 (Automatic user notification): Consider two exemplary situations for user notifications: A general pre-configured notification and a user-configured personal notification.

The ‘Package Responsible Notification’ is a pre-configured notification template to inform users or teams responsible for source code packages of changes applied to the latter (cf. Figure 10-2). As the notification is pre-defined, it does not relate to a concrete person or team, but to a role template defined for a section template (cf. Chapter 7). Via a connection to an event template, it is configured for what type of event relating to the section template a notification shall be created. The role template is later taken by a resource. When the CPM framework detects that an event relating to the section (artifacts contained in it) has occurred, the person filling the concrete role (or the leader of the team filling the role) is automatically notified.
The second example is presented as a simple assignment notification (cf. Figure 10-3): A user is interested in a certain assignment of another human since his work depends on it. Therefore, he creates a notification template relating to the state of this assignment. When the assignment reaches that state (e.g., ‘completed’), a notification is created and the human is automatically notified.

Procedure 10-1 describes notification handling.

Procedure 10-1 (Notification handling):
1. Notification registration: The CPM framework has pre-configured notification templates to which the human can add new personalized ones.
2. Event detection: As project execution progresses, various events occur and entities change their state. Some of the events and changes can be relevant for notifications. Some are explicitly conducted by users interacting with the framework. Finally, others are detectable by the event management components sensors.
3. Notification generation: The context management component uses the information from the notification template concepts to create concrete notifications for humans.
4. Notification distribution: The information is displayed to the users via the integrated GUI component.

10.2.2. Active Coordination Support

To enable active coordination support (cf. R:Coord:FolAct), the CPM framework must be able to automatically identify areas of interest in a project, e.g., ‘Implementation’ or ‘Architecture’ (cf. Example 10-1). For this purpose, the area concept (cf. Chapter 7) is used. In turn, the latter can be further segregated into the aforementioned sections (cf. Figure 10-4). These definitions can be tailored for projects and automatically supported. For example, the sections can be used to model the package structure of the source code. To enhance such a structure with further information, sections and artifacts can have a type like a source code artifact or an interface section. The latter would, for example, mark a source code package containing interfaces of a specific software component.
With this structure in place, we add three concepts to enable active coordination support: First, the potential impact that connects two areas indicating that a change in the source area might necessitate a change in the target area. Second, we add two concepts for the follow-up activity: the follow-up activity template being a sub-concepts of the extension template (cf. Chapter 7) and the follow-up activity being a sub-concept of the extension (cf. Chapter 7). As sub-concept of the extension (template), the follow-up activity can be defined with its own assignment and work unit container (cf. Chapter 7). Thus, follow-up activates can be guided by the CPM framework just like any other user activity. Furthermore, they can be integrated seamlessly into other work unit containers just like the quality measures of Chapter 9. To illustrate these concepts and their interplay we provide Example 10-4.

Example 10-4 (Active coordination support):
As illustrated in Example 10-1, the modification of a source code artifact that belongs to the interface of a component is the target of this scenario. The respective change may require adapting integration tests or architecture documents. Usually, dependent adaptations do not appear in the workflows belonging to SE processes; thus they are extrinsic workflows. The given example illustrates the case for the follow-up activities regarding the tests shown in Figure 10-4.

Figure 10-4 shows two defined project areas, i.e., ‘Implementation’ and ‘Test’. There is a potential impact configured for related technical issues from ‘Implementation’ to ‘Test’. This captures the fact that changes at source code artifacts might also necessitate changes in the integration tests relating to these artifacts. For the implementation area, there are the Modules x and y having different packages and appearing in the test area. To enable the CPM framework to automatically manage these relations, additional connections between the modules can be established (see the curved lines in Figure 10-4).

Developer 2 is responsible for testing Modules x and y. Assume now that Developer 1 changes a class belonging to Package b, indicated by the change activity. In the first step of the procedure, the CPM framework has configured a rule that uses the technical potential impacts according to changes to source code artifacts belonging to interfaces. The rule determines that the change activity affects the test area. With this information, the second step can be performed, which looks for an impact target relating to the processed component while taking all overlaying components into account as well. In this example, the following impact target is detected: in the ‘Implementation’ area, the source code file belongs to Package b that belongs to Module x, which has a relation to the Module x Section in the area ‘Test’. After determining the concrete target, the recipient of the follow-up activity is determined. In the given case, it is Developer 2. With this information in place, the best matching follow-up
activity is determined and, via the next upcoming extension point of the determined recipient, integrated into his workflow instance. This is illustrated by Figure 10-5 where a follow-up activity for adapting tests is integrated into the Open UP ‘Test Solution’ workflow. This activity includes a separate workflow instance and supportive human activity concepts.

The approach illustrated in Example 10-4 comprises four steps we now explain in detail:

1. **Determine project areas affected by an activity**: The first step is configurable and considers various facts to determine which areas are affected by an activity. For the scenario from Example 10-1, such a configuration can be as follows
   - ‘Search for affected areas in case of technical issues if an activity implies a change to an artifact and the artifact is a source code artifact and belongs to an interface component’.

2. **Determine the concrete target affected within the area**: The second step takes the selected areas and the target of the applied activity as input. This target can be an artifact as in the given scenario or a more abstract section of the project as, e.g., a module. The target is then determined by relations of the different sections. An example can be implementation and testing: the testing of a module relates to its implementation. In the given example, the relation does not need to be in place for the concretely processed artifact, but can be also found if one exists elsewhere in the hierarchy. If there is no direct relation from the processed source code artifact, the framework searches other components the file belongs to, e.g., the module.

3. **Determine the person responsible for the target chosen**: Once the target for the follow-up activity is determined, the responsible persons or teams must be discovered. For example, if the target of the follow-up activity is a source code file with no direct responsible human defined (cf. Class x in area test in Figure 10-4), the superordinate sections are taken into account as well, e.g., the encapsulating packages or modules. If a team is responsible, the information is referred to the designated contact of that team for further distribution.

4. **Determine the concrete activity to be issued**: After determining the target and the responsible party, a concrete activity must be chosen. This is accomplished using properties of the involved artifacts, areas, sections and the activity that was the trigger.

To enable automated detection of follow-up activities, different concepts must be in place for the context management component to be aware of them:

1. The project must be hierarchically split up into components like areas or modules.
2. Connections between related components must be established as, for example, the fact that testing a module relies on implementing it.
3. Information that can be used to clarify under which circumstances a particular area affects another one must exist.
4. Different components must be classified relating to their special properties, e.g., a package in the source code realizing the interface of a component.
Automatic initiation of follow-up activities could result in a large number of new activities for a human. For example, frequent changes in a source code package could result in a high number of activities for adapting the information of a single source code artifact in the specification. This, in turn, may impede the progress of his normal project activities. Therefore, the CPM framework incorporates facilities to configure the treatment of follow-up activities (cf. R:Coord:ActConfig). Therefore, the follow-up activity has a template which enables various options for the configuration of follow-up activities:

- Retention (with the property ‘retain’): If retention is enabled, no automatic activity initiation takes place. The activities are collected in a list and can be manually analyzed and distributed.
- Thresholds: A threshold is used to configure how many follow-up activities must be in place for a human before they all are distributed. For example, if a follow-up activity template has a threshold of 3, activities based on this template will be distributed in groups of three or more when that number of activities for one user has been reached. That way, the user will not be disturbed multiple times by these activities but he will receive them all at once.
- Grouping: This option can be set when thresholds are active. If set, all follow-up activities for one user are grouped together in one larger activity. That way, a user can save time on multiple small similar activities as, e.g., multiple adaptations to a specification.

### Conceptual Framework

This section extends the CPM conceptual framework such that it enables active automatic activity coordination support as well. The concepts used for modeling the facts needed are shown in Figure 10-6.

Separating the project into logical components is accomplished based on the area, the project component and its sub-concepts, the section and the artifact. An assignment activity, which is executed by a human, processes a particular project component. In turn, a project component has a responsible role taken by a resource being a team or a person. An activity that changes an artifact can impact various other artifacts processed in other areas of a project. As discussed, the CPM framework provides concepts to model such impacts: The potential impact concept captures potential impacts between areas (cf. Figure 10-6 1) like ‘When a technical change happens to a component in area a, this has an impact on area b’. In particular this comprises the source and target areas as well as a type for the impact (i.e., technical). Project components of different areas may be inter-related (cf. Figure 10-6 2) like ‘Testing of module x relates to the implementation of module x’. Concepts may have sub-
concepts further classifying them (cf. Figure 10-6) and depend on certain conditions. This applies to the project component and its sub-concepts. Figure 10-6 shows two sections with special properties.

As final step of the procedure, the assignment of follow-up activities must be integrated in the respective human’s operational workflow. This step is much simpler than the quality measure distribution (cf. Chapter 9) because the follow-up activities are assumed to be smaller and less time consuming activities: Therefore, the next upcoming extension point for that user that has the matching type is taken.

10.3. Discussion

This section discusses various approaches targeting at automated collaboration support. CASDE [JYW07] and CooLDev [LeBo07] utilize activity theory for building an environment supporting collaborative work. CASDE features a role-based module managing mutual awareness of different roles. CooLDev is a plug-in for the Eclipse IDE that manages activities performed with other plug-ins in the context of global cooperative activities. Ariadne [SQTR07], which is another Eclipse plug-in, is designed to visualize social dependencies between developers arising from technical dependencies of their code within the configuration management repositories. These approaches strongly focus on activities, humans, and roles. They do not have capabilities for integrating the same concepts for artifacts. The CPM framework not only integrates artifacts, but also features sensor concepts to remain updated on the states of artifacts via modification tools used in the projects.

The approach presented in [GHM98] provides a tool for semi-synchronous and asynchronous editing of artifacts as well as a tool for defining and enacting simple collaborative processes. As opposed to the CPM framework, this concept incorporates rather simple cooperative activities and does not provide holistic support for entire projects. By contrast, CAISE [CCJ04] is a collaborative SE framework with the ability to integrate SE tools. CAISE supports the development of SE tools based on collaboration patterns. Thus, it is capable of holistic project support. However, it does not have the active capabilities of the CPM framework for supporting collaboration or software quality management.

CognitiveDust [BSV07] builds on CSCW concepts and aims to support small teams automatically executing support actions that foster creativity and collaboration. The focus lies on the analysis of the human perception. In this area, it goes beyond the capabilities of the CPM framework. However, it is not capable of holistic support for projects like the CPM framework. Caramba [Dust04] targets at the coordination of virtual teams with support for ad hoc processes. This is achieved by enabling links between different resources, artifacts and processes. Processes can be pre-modeled in an UML Activity Diagram notation. If a process does not fit a defined template, an empty process is instantiated, leaving the team members the opportunity to coordinate their work using so-called organizational objects. Thus, Caramba is relatively similar to the CPM framework. However, it is not capable of automatically enacting, supporting and changing the processes as they are executed. If the process does not fit the situation, Caramba requires manual intervention and can only provide limited support.

An industry approach for collaborative development is provided by IBM Jazz and Rational Team Concert [IBM08]. Jazz offers an infrastructure for distributed development including the technical basis for integrating various clients as well as data and services. It enables comprehensive project, bug and configuration management as well as event notifications, traceability and other software development related tasks. Team Concert is a collaborative software development environment, which was built on Jazz technology utilizing its capabilities to provide an integrated solution for software configuration management, work item management and build management with additional features like customizable dashboards, milestone tracking, or process templates for common processes. The greatest difference to our approach is the absence of active support capabilities in Jazz and Team Concert. The CPM framework supports collaboration by covering obscure relations between different
artifacts and areas. Finally, it not only enables notifications about changes, but can even schedule automated follow-up activities.

10.4. Summary

The high degree of dynamic collaboration in SE raises challenges for automated support of process awareness and guidance in software engineering environments (SEE). Currently, SEE lacks contextual information and integration, especially with regard to adaptive collaboration and workflows. The CPM framework enables the explicit modeling and management of both intrinsic and extrinsic activities. It further provides active and passive collaboration support for SE. This incorporates three different types of support:

The framework supports humans with extended information related to the activities they are processing. Not only the current activity is shown, but also the different steps of the activity and the current task. Furthermore, the user has high level information on what has to be done next. Additionally, the framework monitors process progress on different levels and provides navigability information concerning more abstract process areas and related items (e.g., milestones) that might concern the user. Communication and collaboration of different humans is further supported by passive information distribution automatically conducted by the CPM framework. Configurable mechanisms are in place to enable various notifications concerning various events or state changes of various entities or activities.

Finally, the CPM framework enables the automatic initiation and governance of related follow-up activities caused by other activities. Based on the concepts introduced in this chapter, related components can be associated even if no direct relations between the source component and the target component exist. Thus, the framework can automatically determine possible effects of activities, even when they concern different teams or different areas of a project. The human responsible for a component can also be determined if no direct responsibility is defined. However, automatic activity initiation has the potential to overwhelm users with numerous micro tasks and thus delay more important scheduled activities. Therefore, options are integrated for flexible configuration of activity governance that even enable the complete retention of the activities and manual distribution if desired.
11. Exception Handling in Software Engineering Processes

The development of software is a very dynamic and knowledge-intensive process that depends on a variety of environmental factors as well as humans and their effective collaboration. As opposed to production processes that are highly repetitive and predictable [LeRo00], SE processes have hitherto hardly been considered for automation. Existing SE process models, like VM-XT [RBTK05] or the Open Unified Process [Ecfo15], are rather abstract and thus do not really reach the executing humans at the operational level [Wall07]. In sparsely governed processes, deviations from the planned process, exceptions, or even errors often remain undetected [RDB03]. Even if detected, an automated and effective exception handling is hard to find.

To increase the level of standardization of process enactment (e.g., in respect to repeatability or conformance), automated SE process support is desirable. To enable such process support in a holistic way, a framework should be able to handle exceptions during SE process enactment. In particular, the occurrence of exceptions should not deteriorate process performance. Furthermore, automated process exception handling will only be acceptable to humans if it is not too complex or more cumbersome than manually handling the exception [EKR95]. However, automated handling also implies automated detection of exceptions, which depends on the capabilities of the WfMS enacting the processes [LSKM00, ReWe12]. However, existing systems are still rather limited regarding the automated detection and handling of exceptions [RAH06].

Generally, exceptions can arise for various reasons such as constraint violations [LRKD11], deadline expiration [RAH06], activity failures [BRK+06], or discrepancies between the real world and the modeled process [RHEA04b, RHD98]. Especially, in the highly dynamic SE process domain, exceptions arise from various sources, and it can be difficult to distinguish between anticipated and unanticipated exceptions [RDB03]. Even if they are detected, it can be difficult to directly correlate them to a simple exception handler. Exceptions may be related to various items such as activities, artifacts, or the process itself. Many of them are difficult to detect, especially for a system having no direct knowledge of the environment. It might be further unclear when exactly to handle the exception and who should be responsible. Generally, the knowledge about the exception varies greatly, making unified handling difficult and the application of standardized exception handlers unsuitable.

This chapter\(^1\) contributes a flexible approach for exception handling incorporating the following features:

- Detecting exceptions from various sources.
- The flexible handling of exceptions by separately determining the problem, handling and responsible human.
- Combining high automation where possible and human involvement where necessary.

\(^1\) This chapter is partially based on the publication [GOR11e].
11 Exception Handling in Software Engineering Processes

11.1. Requirements

This section summarizes basic requirements and elicits specialized requirements relating to SE process exception handling (i.e., extensions of the requirement R:Exc). For illustration, consider Example 11-1 of ‘The Company’:

Example 11-1 (Exception handling shortcomings):
As The Company does not have a properly implemented, governed and supported SE process, it is not able to properly control exceptions relating to the SE process. As aforementioned, project planning (including the development schedule and the related process) is only executed abstractly and the concrete processing of the user assignments is completely left to the executing humans. The same applies to progress reporting regarding the process as executed. In such an environment, various exceptions might occur, but remain undetected. A factor aggravating the handling of these exceptions are the various possible causes: For example, exceptions may occur directly related to executed activities, to SE artifacts, or to problems with the process. In the following, a concrete scenario is presented, illustrating such exceptions in practice: In applying a bug fix to a source code file, the removal of a known defect might unintentionally introduce other problems to that file. For example, source code complexity might increase if multiple humans applied “quick and dirty” fixes. Thus, the understandability and maintainability of that file might drop dramatically and raise the probability of further defects.

A CPM framework being able to handle exceptional situations (cf. Example 11-1) must address several requirements related to contextual integration (cf. R:ContInt) and automation (cf. R:AutoProc) of the SE process (cf. Chapters 4 and 7). In addition to these basic requirements, a framework must consider more specialized requirements related to SE process exception handling. These requirements are elicited as extensions of the basic exception handling requirement R:Exc (cf. Chapter 4):

- **Requirement R:Exc:ExcOcc (Detecting exception occurrence):** The CPM framework shall enable the automatic detection of various exceptions; i.e., to automatically infer the occurrence of an exception based on various events acquired from the framework environment.
- **Requirement R:Exc:ExcHand (Determining exception handling):** The framework shall determine situationally matching exception handleings. On one hand, this depends on the correct classification of the exception. On the other, it depends on contextual factors the framework must incorporate (e.g., properties of the current project or situation).
- **Requirement R:Exc:RespDet (Determining responsible person):** The framework shall automatically determine the human responsible for handling a particular exception. Note that this is no trivial task. Depending on the exceptional situation, a human could be responsible for an involved activity or artifact or maybe the principal of a human.
- **Requirement R:Exc:GovExcHand (Governing exception handling):** The framework shall automatically initiate and govern exception handling. When all parameters of the exception and the planned handling are determined, the framework must have access to the process only then it will be able to automatically initiate a handling, distribute it to the responsible human, and govern its execution.
- **Requirement R:Exc:IncExcKnow (Dealing with incomplete knowledge):** The framework shall deal with incomplete knowledge about exceptions. In many situations, not all needed data about an exception might be in place. The framework shall also take action in respective situations, utilizing all available knowledge.

Table 11-1 sums up the requirements for automated process exception handling.
Table 11-1: Exception handling requirements extension

<table>
<thead>
<tr>
<th>Req ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R:Exc</td>
<td>Dynamic Exception Handling</td>
<td>The CPM framework shall provide dynamic exception handling for complex exceptions.</td>
</tr>
<tr>
<td>R:Exc:ExcOcc</td>
<td>Detect Exception Occurrence</td>
<td>The CPM framework shall enable automatic detection of the occurrence of exceptions.</td>
</tr>
<tr>
<td>R:Exc:ExcHand</td>
<td>Determine Exception Handling</td>
<td>The CPM framework shall determine situationally matching exception handlings.</td>
</tr>
<tr>
<td>R:Exc:RespDet</td>
<td>Determine Responsible Person</td>
<td>The CPM framework shall automatically determine the responsible for an exception handling.</td>
</tr>
<tr>
<td>R:Exc:GovExcHand</td>
<td>Govern Exception Handling</td>
<td>The CPM framework shall automatically initiate and govern exception handlings.</td>
</tr>
<tr>
<td>R:Exc:IncExcKnow</td>
<td>Deal with incomplete knowledge</td>
<td>The CPM framework shall deal with incomplete knowledge about exceptions.</td>
</tr>
</tbody>
</table>

11.2. Flexible Software Engineering Exception Handling

To respond to the special properties of dynamic SE process enactment, this chapter proposes an advanced process exception handling approach. Basically, the latter is grounded on two properties: (1) The ability to automatically gather contextual information utilizing special sensors and complex event processing; and (2) An enhanced flexibility in the handling of exceptions by separating different concerns. The latter include determining the responsible human as well as the addition of concrete countermeasures to the process. This section introduces the building blocks of this exception handling approach.

11.2.1. Abstract Approach

The presented approach to exception handling in SE processes can be roughly understood as extension of ECA (Event-Condition-Action) approaches [Pato99]. In the following, developed concepts for exception handling are elaborated. Further, these concepts are mapped to concrete concepts previously introduced in this thesis:

**Event:** *Event* is used to capture a multitude of events that may occur in the course of an SE project. Examples include the saving of a source code artifact in an integrated development environment (IDE) or the execution of a static source code analysis tool that provides certain metrics. These metrics can be indicative of an emerging problem and thus lead to an exception.

**Exception:** The notion of *exception* is utilized to classify a deviation from the plan that was recognized to have a potential negative impact on the SE process and thus should be dealt with to avoid such an impact. In [ReWe12], a distinction between anticipated exceptions, whose occurrence can be easily foreseen, and unanticipated exceptions is made. For the former, standard exception handlers can be defined, whereas this is usually not possible for the latter. Since SE projects typically feature a dynamic process and it might be difficult to foresee a multitude of possible exceptions, we do not discriminate between anticipated and unanticipated exceptions. Further, we do not use standard exception handlers tied to specific exceptions. In particular, flexibility is improved through the explicit separation of events, exceptions, exception handlings, responsible humans, and the point in the process where the handling shall be invoked. This enables the CPM framework to react specifically to various exceptions. This can relieve humans, as they do not receive exceptions abstract exceptions but rather specific ones including guidance in resolving them.

In the CPM framework, occurring events can be classified, and it can be separately determined whether exceptions shall be raised, what to do with them, when to do it, and who shall do it. Utilizing this separation, together with the contextual awareness features of the CPM framework, various exceptions can be detected that other approaches might miss. Additionally, the approach manages
different levels of knowledge about occurring events (cf. R:Exc:IncExcKnow). Depending on the amount of available event knowledge, it can be decided whether a generic exception shall be raised or a more specialized one. The more specific an exception is, the more data it stores in the CPM framework about what happened and how it may be resolved. A generic exception might only be connected to an event that triggered it, whereas a specific artifact-related exception might be triggered when a problem arises with an artifact. Respective information will then be stored in the exception and can be used to specifically determine a matching handling and human to handle it. Figure 11-1 shows a basic exception hierarchy including three different kinds of exceptions. In general, an arbitrary number of different exceptions types is possible in the CPM framework. For the sake of simplicity, we show a selection of relevant ones.

![Figure 11-1: Exception hierarchy excerpt](image)

As stated in [RHEA04b], anticipated exceptions occurring during the enactment of pre-specified workflows include the following categories: activity failures, deadline expiration, resource unavailability, discrepancies (between a real-world process and its computerized counterpart), and constraint violations. Therefore, we show three types of exceptions here capable of covering this selection: activity-, artifact-, and process-related exceptions (cf. Figure 11-1). The presented exception hierarchy is not intended to cover every possible exception in every project. It rather presents an extensible basis for frequent exceptions:

- **Activity-related exception**: This type of exception covers exceptions occurring in relation to an executed activity. Three sub-types exist: The *activity failure exception* is used when an activity is executed by a human, but the goal of the activity cannot be reached. The *activity dependency exception*, in turn, becomes relevant when the activity cannot come to execution as something it relies on (e.g., the executing human) is not in place. Finally, the *activity deadline expiration exception* is applied when the activity has not been started and the planned end date is reached.

- **Artifact-related exception**: This exception type covers the *artifact complexity exception*: Reconsider the scenario from Chapter 4. The complexity of a source code artifact is high and the threshold exceeded by a particular activity. The problem may be detected later and then relate more to the artifact than to the activity. Finally, the appropriate human to deal with the problem could be the one responsible for the entire artifact rather than the last human who worked on it.

- **Process-related exception**: This type of exception relates to process enactment and deviations from it: The *unintended activity execution exception* covers the case when an activity is processed that is not present in the current workflow instance. The *activity omitted exception*, in turn, covers the case when one of the intended activities is omitted in reality.

**Handling**: The notion of handling is used to describe activities or workflows executed as countermeasures for a triggered exception. Since SE exceptions are usually complex, no simple rollback of the activities that caused the exception can be done. As an example, consider the activity of bug fixing (cf. Example 11-1): While fixing a bug, this activity might introduce additional problems to the code such as increased code complexity. This might happen when the human applying the bug fix is not the one responsible for the processed artifact. As a countermeasure, an explicit refactoring might become necessary. handling neither comprises the human to execute these activities nor the point in time or point in the process where they are to be executed.

**Responsible**: Responsible captures the responsible human for handling exceptions. As in Example 11-1, this can be the human that executed the activity introducing the exception or the human responsible for an artifact related to an exception.
**Target:** Target is the point in the process the handling shall be executed. For certain exceptions, it can be suitable to integrate handling directly into the workflow instance where the exception occurred, whereas in other cases a separate exception handling workflow instance shall be enacted.

### 11.2.2. Conceptual Framework

This section deals with the mapping of the abstract concepts for exception handling to CPM concepts. Figure 11-2 illustrates it by means of two concrete workflows.

![Figure 11-2: Mapping of exception handling concept to CPM](image-url)

Figure 11-2 shows a sample workflow (i.e., ‘Develop Solution Increment’ from the OpenUP process [EcFo15]) and its mappings in the context management component. The latter includes an assignment, assignment activities, activity steps, and atomic tasks (as introduced in Chapter 7) that are executed by persons being parts of teams. The work unit container comprises a work unit that has an associated extension point allowing for the insertion of an exception handling. The ‘Event’ concept of the sketched exception handling approach, in turn, can be directly mapped to the CPM event. The event has a relation to exceptions that are, in turn, directly mapped by an exception concept. The handling for the exception is mapped by an exception handling concept that includes a complete work unit...
container, comprising all user related support concepts CPM provides to be able to support the exception handling executed by a human. Finally, the ‘Responsible’ of an exception handling is mapped by a person or team in CPM. That way, it becomes possible to automatically distribute exception handlings to users.

11.2.3. Concrete Procedure

The procedure we apply to enable automated exception handling is separated into three phases: recognition, processing, and action. Each of them comprises several activities performed by the CPM framework. The procedure is illustrated in Figure 11-3 and described thereafter.

Figure 11-3: Abstract exception handling concept

**Recognition phase:** In this phase, low and high level events are gathered from the environment (cf. R:Exc:Occ). For this purpose the steps of Procedure 11-1 are performed:

**Procedure 11-1 (Exception handling - recognition phase):**
1. Event Detection: To enable automated assistance for exception handling, the detection of events related to exceptions must be automated. In a SE project, these events relate to processed activities and artifacts, and thus to supporting tools (like source control systems or IDEs) as well. Therefore, the event management component gathers a multitude of events from various tools like IDEs or source control management tools.
2. Event Aggregation: Automatically recognized events relating to the tools in an SE project provide information about currently executed activities. Nevertheless, these events are often of rather atomic nature (like saving a file) and provide no information about the complex activity a human is processing. Therefore, atomic events need to be processed and aggregated to derive higher-level events of more semantic value (like the application of a bug fix).

Example 11-2 illustrates the steps of Procedure 11-1 by means of a concrete situation.

**Example 11-2 (Exception handling - recognition phase):**
We refer to the scenario from Example 11-1. A developer working on a source code artifact to fix a bug introduces another problem. On one hand, the event management component detects that the developer is changing source code and the source code artifact the changes are applied to. On the other, because a static code analysis tool is executed, an event is generated informing the CPM framework that a new static analysis report is available. This report is parsed and, utilizing the rules processing component, checked for any problem in the source code violating defined thresholds. For metrics violating thresholds, an event is generated, informing the framework that new problems exist and to what artifacts they relate.

**Processing phase:** In this phase, necessary exception handling parameters are determined (cf. R:Exc:ExcHand and R:Exc:RespDet). Procedure 11-2 explains the different steps.

**Procedure 11-2 (Exception handling - processing phase):**
1. Event Classification: Event classification can be used to gain knowledge about events to be able to find a specific handling later. The gained knowledge can be related to the current project and its
properties, (e.g., its quality goals as introduced in Chapter 9) or the current situation (as introduced in Chapter 7).

2. Handling Determination: When an exception occurs, it must be decided when and how to take measures against it. In general, this choice depends on the current project situation. The latter can be classified based on parameters like risk or urgency (cf. Chapter 8). If urgency is high, meaning there is a high schedule pressure on the project, one may decide not to address the exception immediately, but to retain it for deferred handling. Since this approach, using event classification, can cope with different levels of knowledge about events, it may be decided to retain an exception if the available knowledge does not suffice for immediate automatically supported handling. Furthermore, different types of exceptions with different handleings can be connected to different events related to different levels of knowledge. Thus, generalized exception handleings are applicable in situations in which only a small amount of knowledge about the exception is present.

3. Responsible Determination: If immediate action shall be taken in case of an exception, the human responsible for performing that action must be determined. For this, various options exist: For example, if an activity-related exception occurred, the human who processed that activity can be responsible or, if an artifact-related exception occurred, the responsible human for this artifact or source code package may be responsible for handling the exception as well. There may be no direct responsible party for each processed artifact. However, note that responsibilities can be hierarchically structured to simplify determination of the responsible party (cf. Chapter 10).

4. Target Determination: When the responsible party for handling the exception is determined, the concrete point in the process the handling shall be applied to must be determined. In certain situations, it may be appropriate to directly integrate a handling in a running workflow instance. In other cases, a new workflow instance needs to be initiated. As exception handling is a sub-concept of the extension, as is the measure concept described in Chapter 9, the integration procedure of handleings can be executed in a similar fashion to the measures described in Chapter 9.

5. Exception Retainment: If no immediate handling is favored due to the parameters of the situation, the exception is retained in an exception list. That list can be analyzed, e.g., at the end of an iteration by the project manager.

The steps of Procedure 11-2 are illustrated by Example 11-3:

Example 11-3 (Exception handling - processing phase):
The information received from static analysis tools (cf. Example 11-2) is received by the context management component to become aware of problems in the source code. To be able to effectively utilize this information, it is further classified. For example, problems related to source code can be further classified utilizing information about the concrete metrics that violated threshold values. For example, assume that the artifact the developer works on to apply a bug fix has violated a threshold related to the ‘Cyclomatic Complexity’ metric. Thus, the problem relates to source code complexity. This information can now be related to the properties of the project. Assume that the project has ‘Maintainability’ and ‘Reliability’ as quality goals for the source code. Thus, the violation in the artifact will be of concern for that project as high source code complexity contradicts good maintainability and high reliability of the code. Therefore, an exception will be raised according to the problem. Assuming further that the situational properties of the project also support this (e.g., ‘Risk’ with a value not too low and ‘Urgency’ with a value not too high), a concrete handling will be determined and be chosen for immediate execution. Having these decisions in place, a responsible party for the handling must be found. In this concrete situation, two options are possible: If the problem is detected while the developer is still working on the artifact, he would be chosen as responsible, as it would be very efficient to distribute the handling of an exception related to that artifact to him while he is working on it. If, at the time the problem is detected, the occurrence of the problem lies in the past, the human responsible for the related artifact would be taken. If no direct responsible is defined, the latter will be searched for in the project hierarchy. For example, there could be a responsible for the source code package and the framework would be aware of this utilizing the project component concept (cf. Chapter 7).
Action phase: In this phase, the concrete execution of the selected exception handling is applied (cf. R:Exc:GovExcHand) via the steps of Procedure 11-3:

Procedure 11-3 (Exception handling - action phase):
1. Handling Preparation: After determining all required parameters, the concrete handling of the exception must be prepared, i.e., a new workflow instance must be created or the handling must be integrated seamlessly into a running workflow instance.
2. Handling Execution: The prescribed handling is executed by the chosen human.
3. Deferred Handling: When exceptions are retained, a human may decide for which exceptions a deferred handling is preferred. Therefore, an additional GUI will be developed presenting a list of retained exceptions and enabling manual determination of a handling or discarding of the exception.

Example 11-4 illustrates the action phase and the steps from Procedure 11-3:

Example 11-4 (Exception handling - action phase):
Consider the scenario described in Example 11-2 and Example 11-3: Based on contextual information, an exception related to the complexity of a source code artifact has been detected. Based on the goals of the project and the properties of the situation, the handling procedure was initiated. In this case, two options were presented: The human introducing the problem is still working on the respective artifact, or the detection of the problem happens sometime after its introduction. In the first case, the framework would directly integrate the handling as one or multiple activities in the running workflow instance of the user. In the other case, a new workflow instance comprising these activities would be initiated automatically for the responsible human. In both cases, as the handling is executed within the CPM framework, the latter would govern and support the handling for the human and enable easy tractability of the handling.

In another situation, an exception may have been raised, but due to situational properties (e.g., a project that is more urgent) it may have been decided not to handle the exception immediately. In that case, the exception would have been retained within the framework. At a defined time point (in this case the end of an iteration), a list of retained exceptions would be presented automatically to the quality manager to enable him to decide on their handling.

11.3. Discussion

This section covers two areas related to the presented exception handling approach in: Exception handling in WfMS and rule-based approaches as we applied a rule-based approach to achieve automated exception handling.

For automatically detecting exceptional situations and determining the actions required to handle them, ECA-based (Event-Condition-Action) models have been suggested. Usually, these approaches limit adaptations to currently enabled and running activities (e.g., to abort, redo, or skip activity execution) [CCPP99]. An approach to enable automated adaptations of the unexecuted regions of a running workflow instance (e.g., to add or delete activities) is AgentWork [MGR04, Müll02], which was implemented based on the ADEPT technology [RRD03, RRK05]. AgentWork allows process adaptations to be specified at an abstract level and independent from a particular process model based on a temporal ECA rule model. Temporal estimates are made when an ECA rule fires during run-time in order to determine which parts of a running process instance are affected by the identified exception. For these parts, predictive and reactive changes are possible. Predictive changes are applied immediately, whereas reactive changes are applied at the time the concerned process fragments are entered. Another approach to workflow adaptation is presented in [DZG10]. It consists of a rule-based, data-driven approach to workflow adaptation. Therein, hierarchical context rules are utilized to tailor workflows to changing data contexts. Additionally, for environments involving event paradigms, an
event-driven adaptation pattern catalogue is presented. An example is the context-dependent cancelation of a workflow segment and the triggering of a special handler task.

The discussed approaches are event- and rule-based as our approach. However, they cannot utilize the variety of contextual events since they lack environmental sensors. Furthermore, the approaches are rather rigid in the way exceptions are handled, since events, conditions and related actions are connected statically. Our approach not only splits exception treatment into additional refinement steps, including contextual classification, but also allows for the flexible assignment of handlings based on various factors. That way, an appropriate handling can be found for various situations taking different levels of knowledge about a situation into account. Greater flexibility for the exception handling itself is achieved by adaptively combining what is to be done, who shall do it, and when it is to be applied.

Exception handling could also be applied utilizing only the WfMS. For example, most BPEL (Business Process Execution Language) workflow engines support fault handlers to enable some kind of exception handling, for instance WebSphere [KKL+04]. However, usually, respective engines do not provide advanced adaptation abilities. As opposed to our approach, the automatic exception handling abilities of such systems are rather limited since they lack both access to context information and contextual classification capabilities.

Another area of related work are rule-based approaches and their application to SE environments. Classical examples include MARVEL [Barg92b], OIKOS [MoAm94] and Merlin [JPSW94], which have been discussed in Chapter 7. In MARVEL, rules are defined in a proprietary language to enable forward and backward chaining. The system may request additional activities from a human executing an activity to satisfy the preconditions of the desired action. OIKOS rules, in turn, are defined in Prolog and are processed by agents. The cooperating agents operate in different workspaces and enable user cooperation. Furthermore, Merlin processes different contexts being assigned to roles. Between these contexts, artifacts are distributed to foster collaboration. As opposed to these approaches, ours features the combination of an extended flexible rule-based approach with an advanced adaptive WfMS, contextual classification abilities, and sensors providing contextual information. Therefore, process enactment is more robust and the discrepancies between real world and modeled process are minimized compared to the approaches discussed here.

11.4. Summary

SE is a very dynamic domain, posing significant challenges for process management. SE process models are often abstract and document-centric, and they are not directly utilized in process enactment. Moreover, SE processes depend on a variety of environmental and contextual factors. Appropriate process automation could enhance quality and repeatability in SE to better connect the abstract processes with the operational level. However, such a process automation framework must be able to accommodate these various aspects and to deal with a variety of unforeseen situations regarding process enactment. A major issue for automatic process implementation are unforeseen situations and thus exceptions to the planned process. This chapter presented an approach to enable flexible exception handling incorporating diverse features to support the dynamic SE process.

Exception detection is supported by a set of sensors gathering environment knowledge and by complex event processing that combines those events to derive higher-level events with more semantic value. Classification of events based on various factors like the current situation or the goals of a project is enabled as well. The proposed approach can deal with different levels of knowledge concerning events and exceptions and thus does not require the separation between anticipated and unanticipated exceptions.

The combination of environmental awareness with contextual capabilities enables the discovery of links between activities and exceptions having no direct connection. These features allow determining a situationally matching handling for an exception. The flexibility of the handling is enhanced by separately determining the handling, the responsible party, and the target, where the handling shall be
placed within the process. Based on the event gathering and contextual aggregation of knowledge, it is possible to flexibly determine the right human for the handling. By the contextual extension to process management concepts, exception handling can be automatically and seamlessly integrated into users’ running workflow instances and be guided and supported by the system. If, due to various reasons, a contemporaneous handling is not favorable, deferred handling and analysis of exceptions are enabled as well.
12. Knowledge Management Support in Software Engineering Processes

The process of creating software is a highly dynamic one whose support and governance is far from being trivial. Typically, the SE process involves the development of new products, concepts or components and, therefore, cannot be standardized and automated as in, for example, industrial production. Product development is further considered as a knowledge-intensive task [RaTi99] and SE processes are mostly knowledge processes [KeHa02]. Still, SE is a relatively immature discipline. Although work has been spent on closely integrating knowledge and process management to support SE, a comprehensive and viable solution is elusive. Currently, SE process management is often accomplished in a documentation-centric (e.g., Rational Unified Process) or agile way (e.g., Scrum) lacking any automated process support.

Knowledge management is crucial to enable the distribution of knowledge among different humans as well as to keep and exploit experience gained in various projects. Supporting this with an automated system can be beneficial [TFB00]. In turn, important capabilities of such a system are to capture, maintain, reuse, and transfer knowledge [TFB00]. Wikis are often used for SE knowledge management because of the easy creation of and access to knowledge [SBBK08]. However, retrieval of contextually relevant knowledge from Wikis still remains a challenging task [SBBK08]. Although relevant knowledge is captured and stored, its reuse is still problematic. However, this could be facilitated if knowledge use was connected with process enactment.

By combining these management disciplines, human-relevant SE knowledge could be used at specific points in the SE process (e.g., source code implementation). Usage of such knowledge in SE everyday work can be easily forgotten without automated support. Even if knowledge is used, its relevance and usability depend on contextual factors like the type of component being developed or the skill level of the SE engineer executing the task. To accommodate this, this chapter provides an approach based on the introduced CPM concepts (cf. Chapter 7) for supporting the SE process with guidance. The approach\(^1\) provides the following features:

- An automatic connection between knowledge and process management is established.
- Knowledge is seamlessly integrated into everyday SE workflow enactment. Humans are not distracted.
- Contextual information is utilized to dynamically choose knowledge matching the humans’ current situation.

12.1. Requirements

This section details the already elicited requirement concerning knowledge management in SE processes (R:Know). Consider Example 12-1 for knowledge management problems as applied in ‘The Company’:

Example 12-1 (Knowledge management shortcomings):
Assume that in The Company, SE process enactment lacks guidance, traceability and automated support. The process is only recorded and documented passively in tools not directly involved in

\(^1\) This chapter is partially based on the publications [GOR11f], [GOR12d], and [GOR16].
process. Thus, the execution of human assignments being part of the process is not guided, but largely depends on the executing humans. The same applies to the acknowledgement of assignment completion: humans must manually enter information about that to make it visible in the process.

The same problems apply to the knowledge management sector in The Company. Although it is known that software development is a knowledge-intensive task, no efforts have been made in The Company to create a facility enabling effective and efficient knowledge transfer between different humans or teams. In many cases this impedes effective software development because specific knowledge about the project and its process, the used technology, or customer requirements concerning the implementation is disseminated slowly. Hence, situations like the following often might occur: An SE engineer solves a specific issue concerning a framework utilized (e.g., he discovers a best practice for using a complicated source control system), but that solution remains only in his mind. He distributes it orally to some of his colleagues, but the entire team cannot benefit from the solution. Others also have to find that solution on their own consuming additional time. In some teams in The Company, the use of special Wikis has been tried out to store and share such knowledge. These attempts have not been successful as, for example, the stored knowledge often was not well-structured impeding its retrieval.

Based on this example, the detailed knowledge management requirements are now elicited as extensions of the requirement R:Know (cf. Chapter 4). As a basis for a solution providing automated support in this area, the CPM framework must comprise facilities for automation support regarding the process (cf. R:AutoProc) as well as the integration of contextual information (cf. R:ContInt). These requirements have been elicited and covered in Chapters 4 and 7.

- **Requirement R:Know:KnStor (Knowledge store):** To be able to use and disseminate knowledge in an SE project, some facility to store knowledge is required. The storage and management of knowledge shall not be cumbersome for humans.
- **Requirement R:Know:ExtKnow (External knowledge):** Since not all the knowledge required by humans is stored locally in the framework, a facility to integrate external knowledge sources is required. Examples of such knowledge include process documentation or external web pages.
- **Requirement R:Know:KnSel (Knowledge selection):** The knowledge provided by humans should be available to the CPM framework to be automatically distributed. This includes automatic access to the knowledge store and machine readable semantics of the knowledge to enable automatic selection of fitting knowledge. For example, the knowledge on the use of a source control system from Example 12-1 must have some kind of annotation indicating that it has to do with tool usage and source control management.
- **Requirement R:Know:ContKnow (Context-based knowledge matching):** Automatically provided knowledge must match the humans current situation and context. Hence, there should be a means to utilize contextual information for knowledge selection. An example of such context information is an SE engineer just modified a specific source control system.
- **Requirement R:Know:KnowProv (Knowledge provision):** There shall be a facility to automatically inject the selected knowledge to standard SE process enactment. Humans should not be distracted or burdened with additional effort for using the knowledge. Seamless integration with everyday work is crucial.
- **Requirement R:Know:CfgKnowDist (Configurable knowledge provision):** Often, specific knowledge applies only to specific points in the process or specific roles in the project (e.g., knowledge about the effective use of a source control system). Furthermore, humans might be overwhelmed by high amounts of knowledge relating to various topics. Therefore, the amount of knowledge provisioned should be configurable, e.g., based on the needs or preferences of projects, processes, or humans.

Table 12-1 provides an overview of the new requirements.
Table 12-1: Knowledge management requirements extension

<table>
<thead>
<tr>
<th>Req ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R:Know</td>
<td>Knowledge management support</td>
<td>The CPM framework shall provide automated knowledge management support.</td>
</tr>
<tr>
<td>R:Know:KnStor</td>
<td>Knowledge store</td>
<td>The CPM framework shall enable easy storage and structuring of knowledge.</td>
</tr>
<tr>
<td>R:Know:ExtKnow</td>
<td>External knowledge</td>
<td>The CPM framework shall be able to integrate external knowledge sources.</td>
</tr>
<tr>
<td>R:Know:KnSel</td>
<td>Knowledge selection</td>
<td>The CPM framework shall be able to access stored knowledge on a semantic level.</td>
</tr>
<tr>
<td>R:Know:ContKnow</td>
<td>Context-based knowledge  matching</td>
<td>The CPM framework shall be able to utilize context information for knowledge selection.</td>
</tr>
<tr>
<td>R:Know:KnowProv</td>
<td>Knowledge provision</td>
<td>The CPM framework shall be able to seamlessly integrate provided knowledge with process enactment.</td>
</tr>
<tr>
<td>R:Know:CfgKnowDist</td>
<td>Configurable knowledge  distribution</td>
<td>The CPM framework shall enable facilities to configure the knowledge distribution</td>
</tr>
</tbody>
</table>

12.2. Software Engineering Knowledge Management Approach

SE project knowledge provisioning presents unique challenges since it involves knowledge with specific properties: this knowledge is directly process-relevant and dynamic (e.g., items can be adjusted due to defect causal analysis). In addition, it is contextually dependent (i.e., some items are irrelevant in certain contexts, are activity-specific, specific to programming language, or platform-specific), and to some degree specific to a user profile (e.g., junior vs. senior engineers, database developer vs. GUI developer). To match various situations, respective knowledge is provided by the CPM framework considering various contextual facts. Furthermore, to provide matching knowledge for the different situations and configurations in an SE project, the knowledge has a set of properties enabling different knowledge types. The following sub-sections give insights into the properties and types of knowledge as well as the procedure being applied for its distribution. First of all, we introduce the components that realize the knowledge support as well as their collaboration. Then, the organization of the knowledge and its application is discussed followed by a description of the concrete knowledge-provisioning procedure and an example.

12.2.1. Basics for Enabling Software Engineering Knowledge Management

This section provides the basics for satisfying the aforementioned requirements and thus for building a knowledge provision system (KPS) for SE on top of the CPM framework. Figure 12-1 illustrates this.

As basis of any KPS for SE a facility to store the knowledge is required (cf. R:Know:KnStor). Therefore, we add a knowledge store as illustrated by Figure 12-1 (1). This component shall enable the structured storage of knowledge and, via a GUI, the creation and management of this knowledge (2). To enable automated dissemination of the knowledge (cf. R:Know:KnSel), the CPM framework must
access and select the knowledge automatically. Therefore, we integrate machine readable-semantics to the knowledge in the store (see Example 12-2 for an illustration).

However, to be usable in SE projects, the KPS must incorporate management facilities as well. The latter shall enable the configuration of the knowledge provision (cf. \( R: \text{Know:CfgKnowDist} \)) as well as the integration of external knowledge (cf. \( R: \text{Know:ExtKnow} \)). For this purpose, we add a management component called knowledge provider (3). To further enable context based knowledge selection (cf. \( R: \text{Know:ContKnow} \)), this component interacts with the CPM context management component providing the contextual information (4). Finally, to enable the direct integration of knowledge provision with the SE process (cf. \( R: \text{Know:KnowProv} \)), the knowledge provider interacts with the CPM process management component (5). Example 12-2 illustrates this with a concrete situation.

**Example 12-2 (CPM knowledge management approach for SE):**

Recall the situation from Example 12-1: An SE engineer discovers a best practice for using a complex source control system. In particular, he describes different things to be kept in mind when creating a source control branch within this tool. To make this knowledge available to other team members, he uses the GUI of the KPS and enters it to the knowledge store. Further, the engineer annotates the knowledge with tags like ‘source control system’ or ‘source control branches’ to foster the automatic distribution by the KPS. In addition to this, he found external knowledge regarding this situation in the documentation of the SE process model The Company applies (e.g. the OpenUP process [EcFo15]). He further adds this knowledge via the KPS GUI and enhances it with information on where this knowledge might be applicable in the process. With this information and the contextual information gathered from the context management component, the KPS can automatically inject this knowledge at the right places in the SE process and thus automatically support the other SE engineers.

We now provide details on how the different components interact to enable automated knowledge provision. Therefore, we utilize the CPM components for context, process and event management. Further, we add the aforementioned knowledge store and knowledge provider as well as two different GUIs. This is illustrated in Figure 12-2 and explained in the following.

![Figure 12-2: Knowledge provisioning components](image)

The central component of the KPS is the context management component, which stores, aggregates and processes all high-level project information relevant to the KPS. It incorporates context information about humans, artifacts and various events as well as information from SE process enactment. Furthermore, it receives context information from the event management component (1), whose responsibility is the acquisition and aggregation of events from the SE environment (2) (cf. 202
12 Knowledge Management Support in Software Engineering Processes

*R:Know:ContKnow*. This is accomplished by a set of sensors integrated in external tools, such as IDEs (Integrated Development Environments) or source control systems used within a SE project. The process management component enables a SE process model implementation as well as operational SE process support. This is done by means of automated workflows actively governed by that component. The context and process management components interact with each other to enable the usage of context information in the process as well as to better align process enactment with the actual project situation.

Knowledge management is realized by the *knowledge store* and *knowledge provider*. The former is utilized to store human-relevant SE knowledge and to make it available to the KPS via machine-readable semantics (cf. *R:Know:KnSel*). The knowledge provider, in turn, coordinates that knowledge (4) and provides it to the context management component (5) to be injected into the humans’ workflows (3) in the appropriate context (cf. *R:Know:KnowProv*). The knowledge provider is further responsible for the abstract definition of human-relevant knowledge within the KPS (called *guidance items*), which is referenced by the context management component, as well as for the integration of external knowledge resources (6) (cf. *R:Know:ExtKnow*).

The KPS interacts with humans as follows: The human can enter relevant project or SE knowledge (e.g., best practices) using the *knowledge collection GUI* (7) that works similar to a Wiki. That knowledge is stored in the Knowledge Store (8) (cf. *R:Know:KnStor*). The *knowledge management GUI* (9) allows humans to integrate external knowledge (e.g., process documentation at an external web location) or to configure the way the knowledge is provided (e.g., ‘This guidance is applicable to this role at that point in the process’) (cf. *R:CfgKnowDist*). Such configurations are directly stored in the knowledge provider (10). All support and guidance is then distributed to the human by the *process support GUI* (11), which receives its information from the context management component (12). The latter unites information on the activity and workflow from the process management component (3) with additional human-relevant SE knowledge from the knowledge provider (5).

12.2.2. Software Engineering Knowledge Management Specifics

The storage and management of human-related knowledge is realized using the knowledge provider and knowledge store as well as the above mentioned knowledge management GUIs. As aforementioned, knowledge can be collected and stored within the KPS (internal knowledge) or be integrated from external sources (external knowledge). Internal knowledge is collected via the knowledge collection GUI that enables humans to annotate that knowledge with tags. Examples of tags are ‘junior engineer’, ‘front end development’, or ‘high risk’ that may be used to automatically and appropriately select the knowledge required to support the humans in their context. This is accomplished by the knowledge provider, who also manages the integration of external knowledge sources. For organizing knowledge in the KPS, the concept of a *guidance item* (GI) is utilized. All guidance the KPS can distribute to humans is defined by the GIs created with the knowledge management GUI and stored within the knowledge provider.

To enable contextually relevant and meaningful knowledge, the GI has several parameters. Each GI has a set of tags used to describe the knowledge for the KPS as well as the humans entering and managing it. Tags may be any type of identifying property indicating to what the GI applies, like ‘Activity’, ‘Junior Engineer’, or ‘High Risk Artifact’. A GI constitutes an abstract unit of guidance knowledge that may contain an arbitrary number of positions or sub-items. The knowledge defined by it may be static (i.e., pre-defined) or dynamically compiled by the knowledge provider. The latter is only possible for internal knowledge stored in the knowledge store. If a GI is internal and dynamic, the knowledge provider will use its tags to query the knowledge store for items tagged in the same way, creating the GI out of these. Furthermore, a GI has different types, like ‘checklist’, ‘best practice’, or ‘tutorial’. Based on this type, a GI will be treated differently by the KPS. For example, a checklist will have a check mark for each sub-item, whereas plain knowledge will be just shown to the human. Example 12-3 provides an illustration.
Example 12-3 (Guidance item):
Recall Example 12-2: An SE engineer has discovered a best practice regarding the creation of a new branch in a specific source control system. He enters that knowledge in the knowledge store and annotates it with tags like ‘source control system’ or ‘source control branches’. Furthermore, he creates a GI of type ‘best practice’ and adds the tags to it. He configures the GI to be dynamic and internal. That way, the KPS will search for knowledge in the knowledge store with the defined tags and dynamically create guidance from it. Thus, other related knowledge the SE engineers might discover in the future can be automatically added to the guidance by the KPS when dynamically creating it.

12.2.3. Process-centered Knowledge Support

Automatic knowledge support must be aligned to the human’s context, otherwise it is likely to be irrelevant and rejected. Therefore, the activities performed by the humans and governed by the context management component are the initiators for GI provisioning. For these activities, we have defined four properties that decide how the context management component presents GIs to the human: These properties are GI alignment, target obligation, GI usage, and item compilation.

GI alignment governs when a GI (e.g., a checklist) shall be shown to the human in relation to the activity being the target of the GI. There are two options to support humans in preparing an activity or while processing it and to support them when finishing an activity to counteract forgetfulness: ‘Pre’ GI are shown at the beginning of the activity. ‘Post’ GI are shown at the end of an activity.

Target obligation associates the connection to the target activity. Some GIs (e.g., checklists) may be directly tied to a target activity. These are called ‘Synchronous’ and their lifecycle depends directly on the target activity. Other GIs, in turn, may be shown based on certain events (including, e.g., activity termination). These are called ‘Asynchronous’ and their lifecycle is also not tied to activities.

GI usage provides additional optional knowledge that may have to be incorporated. Therefore, this property distinguishes between ‘Required’ and ‘Optional’ GIs. Using ‘Required’ GIs, the target activity will not be marked as complete without also acknowledging the GI.

Item compilation defines how the items of a GI are created: ‘Static’ GIs are pre-defined with a static set of items. ‘Dynamic’ GIs, in turn, are dynamically built by the CPM framework at the time they shall be shown. For these GIs, context properties are incorporated as well: The GIs can have various tags like, for example, ‘Development’. The same applies to knowledge stored in the knowledge store. As example consider tags like ‘Development’, ‘Junior Engineer’ and ‘Database’. At runtime the CPM framework has access to that context information, e.g., on who is executing an activity (and, e.g., his skill level, like junior engineer) or in what area the activity is executed (e.g., relating to database development). Hence, the CPM framework can compile, for example, a dynamic database development checklist for junior engineers. As mentioned, GIs may only be dynamic if they are internal, as the CPM framework has no influence on GIs stored in external knowledge sources like web pages.

One of the four properties, ‘item compilation’, is governed by the guidance item concept. The other three depend on the process. To explicitly define them, two additional concepts need to be introduced, guidance template and guidance. The former defines points in the SE process where the application of guidance for a specific activity is feasible. The latter captures the relation to the concrete guidance item during enactment as well as its status. This is illustrated by Figure 12-3 (for better readability, a combined visualization for workflows and their counterparts in the context management component is chosen here).
The guidance template allows defining its target activity (with a bidirectional connection, cf. Example 12-4). However, it may be also applied to a project component as various SE process models (e.g., the OpenUP) include guidance for certain artifacts. In that case, the guidance will be applied when executing an activity related to the artifact. Besides this, the guidance template allows defining the GI alignment (pre, post), the target obligation (synchronous), and the GI usage (required).

The concrete application is governed with the guidance. The guidance features a connection to the GI attributed to it. Furthermore, for accessing the properties of the guidance template it is connected to it as well. The same applies to the concrete entity the guidance shall be applied to. It has a status describing the state of the application of the guidance including two different states: At the beginning, it has state ‘not applied’. When being shown to the human, it enters state ‘applied’.

Example 12-4 (Guidance template):
Recall Example 12-3: An SE engineer has discovered a best practice regarding the creation of a branch in a specific source control system. He has entered the knowledge in the knowledge store and created a GI for it. To further specify when and how this GI shall be shown to others, he creates a guidance template. This allows him to specify that it is applicable for an assignment activity called ‘create branch’. He further specifies that it shall be shown at the beginning of the activity (pre). Finally, to not bother experienced SE engineers with this knowledge every time they create a branch, he defines it as optional and synchronous. That way, it does not have to be reviewed when executing the activity, and will disappear automatically when the activity is finished.

GI s can be beneficial while performing different activities. Therefore, the CPM framework incorporates GIs for all featured activity concepts including assignments, assignment activities, and atomic tasks. However, as the different activity types have different properties, not all combinations are possible for all activity types. For example, an atomic task executed by a human can be detected by the CPM framework, but it is not governed by a workflow. For such a task, GIs cannot be required, as there is no means to prevent the human from simply switching to another task without consuming the GI. Furthermore, there is no event indicating completion of an atomic task to the CPM framework. Nevertheless, post-GIs are possible, as they can be shown to the human even when he switches to another task. Besides that, asynchronous GIs cannot be required, as they are not tied to another entity after creation. They are just created upon a certain event and remain visible until the human completes them. Table 12-2 shows the different types of GIs and their allowed properties.

Furthermore, the context management component is in charge of tailoring the amount of knowledge shown. It can decide how many GIs will be shown to humans at a certain point in the process and how many sub-items are allowed. This information is stored as part of the process information in the context management component and can exploit other context information: For example, a
development activity conducted in a situation with low risk and high schedule pressure could have a checklist with fewer items than it would have if more time were available or greater risk.

Combining the various properties, different types of GIs are possible. We briefly discuss three of them in the following examples (Example 12-5, Example 12-6, and Example 12-7):

Example 12-5 (Asynchronous static optional assignment pre checklist):
This type of checklist is intended for a high-level human assignment like, for example, ‘Develop a new GUI feature’. Being static, the checklist has a pre-configured set of checklist items defined in the knowledge base. The checklist is configured to be a pre-checklist, meaning that it is shown when the assignment is started to provide knowledge to the human to be considered before he starts working on the assignment. As this may be relatively general knowledge (e.g., to check the requirements specification due to the assignment), the checklist is configured to be asynchronous and optional; i.e., the knowledge is shown at the beginning, but it is not required to complete the checklist for completing the assignment and the knowledge does not disappear when the assignment is completed. The checklist persists for a defined time interval providing the human the option to review the checklist at a later time point, e.g., when he has some time left waiting for a build to complete.

Example 12-6 (Synchronous dynamic required activity pre-checklist):
This type of checklist relates to a more concrete assignment activity executed to complete an assignment, e.g., the creation and execution of developer tests. The checklist is dynamic, meaning the CPM framework will compile its items dynamically at run time depending on context information, e.g., the skill level of the human. The lifecycle of this synchronous checklist is tied to the target assignment activity. In this case, it is required and the human must not complete the activity without having processed the checklist before. As a required pre-checklist, it accompanies the activity during its entire processing time. When the activity gets started, the checklist items are shown guiding the human to prepare the activity. When the human wants to complete the activity, he must complete the checklist as well.

Example 12-7 (Asynchronous dynamic optional task post-checklist):
This type of checklist is intended for atomic tasks performed to complete assignment activities. Examples are ‘Coding’, ‘Debugging’ or ‘Checking in’. In this concrete case, the checklist is

<table>
<thead>
<tr>
<th>GI Target</th>
<th>Assignment Activity</th>
<th>Atomic Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>GI Target</td>
<td>Target Properties</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GI Alignment</td>
<td>Target Obligation</td>
</tr>
<tr>
<td></td>
<td>pre</td>
<td>post</td>
</tr>
<tr>
<td>Assignment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pre-GI</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Post-GI</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Assignment Activity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pre-GI</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Post-GI</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Atomic Task</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pre-GI</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Post-GI</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Table 12-2: Guidance item properties
asynchronous and optional, providing the human the option to consider additional knowledge for the tasks he performs. Being an asynchronous post-checklist, it is shown to the human when he switches tasks, and thus the considered task provides a reminder for things that should be considered during this task (e.g., during the task ‘Unit test creation’, if all functionality of the class to be tested has been covered). As atomic task switches often occur and one task might be executed multiple times in the context of an assignment activity, it can be configured to only show the checklist the first time the task ends to not distract the human too often from his work.

### 12.2.4. Software Engineering Knowledge Provisioning Procedure

To enable the CPM framework to directly utilize context information for process enactment as well as to unite the latter with knowledge management, GIs are added to process enactment utilizing the contextual annotations of the process management concepts (cf. Chapter 7). This is illustrated by Figure 12-4.

![Figure 12-4: Solution components](image)

As introduced in Chapter 7, the process management concepts are mirrored in the context management component by the work units and work unit containers. In turn, these are extended with the activity concepts (assignments, assignment activities, atomic tasks), which may comprise GIs attributed to them utilizing the guidance concept. Figure 12-4 illustrates this for the three GIs ‘G1’, ‘G2’, and ‘G3’ and the relating three guidances ‘G1’, ‘G2’, and ‘G3’. ‘G1’ is attributed to assignment ‘As1’, ‘G2’ to assignment activity ‘Aa2’, and ‘G3’ to atomic task ‘At2’. The GIs are presented to the human through an integrated GUI when the target activity concept is started or finished. The GI knowledge is taken from the aforementioned knowledge store. The latter integrates a semantically enhanced wiki (Semantic mediawiki [KVV06]) for knowledge collection, storage and management. This has several advantages:

- Wikis are a widespread mature technology for knowledge storage and sharing.
- Most SE engineers are familiar with the use of wikis.
- A semantically enhanced wiki adds more structure to the knowledge. Humans are not only able to structure it by creating a page structure, but also add a semantic structure to it. Knowledge can be tagged easily enabling a dynamic flexible knowledge structure.
- A semantically enhanced wiki enables automatic access to the knowledge using the machine readable semantics added to it.
- In this project, a semantically enhanced wiki fits well into the CPM framework as the former uses semantic web technology for adding semantics to the knowledge. Therefore, there is no technology gap to be bridged and the wiki can be easily and seamlessly integrated into the CPM framework.
At the time the GIs shall be provided, they are queried from the knowledge store. This implies either getting a complete GI that is pre-defined in the knowledge store or the dynamic querying of items that have tags matching the context of the current situation. This information, in turn, is continuously gathered by the event management component. The following three procedures (Procedure 12-1, Procedure 12-2, and Procedure 12-3) and figures (Figure 12-5, Figure 12-6, and Figure 12-7) illustrate the process of checklist provision for the three sample checklist types from the preceding section.

**Procedure 12-1 (Asynchronous static optional assignment pre-checklist provision):**

This procedure details the course of action for Example 12-5.

1. Assignment start event: The context management component receives an event indicating the start of a new assignment with an assigned asynchronous pre GI.
2. GI acquisition event: The context management component gets the GI name from the assignment and sends an event to the knowledge provider.
3. GI return: The knowledge store sends the GI back via an event.
4. GI provision: The context management component processes the event and sends it to the GUI for displaying it. The GI is displayed at a special area in the GUI and the human is notified about the new GI being available.
5. GI removal: The GI is removed. This is done in two cases: Either the human indicates the consumption of the GI or the defined life time of the GI ends.

**Procedure 12-2 (Synchronous dynamic required activity pre-checklist provision):**

This procedure details the course of action for Example 12-6.

1. Assignment activity start event: The context management component receives an event indicating the start of a new assignment activity with an assigned synchronous dynamic pre checklist.
2. Additional information request: The knowledge provider sends an event to the context management component to acquire additional knowledge on the situation as needed.
3. Context information provision: The context management component sends the requested context information to the knowledge provider.
4. GI acquisition: The knowledge provider acquires the GI. This includes a query to the knowledge store to get items for the GI.
5. GI return: The knowledge store sends the GI back via an event.
6. GI provision: The context management component processes the event and sends it to the GUI for display. The GI is displayed at a special area in the GUI and gets notified that there is a new GI available.
7. Assignment activity end event: The context management component receives an event from the GUI indicating that the human wants to finish the assignment activity. It checks whether the activity can finish and, because of the required checklist termination, completion is not permitted. The human is informed about the required processing of the GI.
8. GI completion: The human reviews and completes the GI. The related assignment activity terminates.
Knowledge Management Support in Software Engineering Processes

**Figure 12-6: Synchronous dynamic required activity pre-checklist provision**

Procedure 12-3 (Asynchronous dynamic optional task post-checklist provision):
This procedure details the course of action for Example 12-7.
1. Atomic task detection: The context management component receives an event indicating the execution of a new atomic task with an assigned asynchronous dynamic post-checklist.
2. Atomic task detection: The context management component receives an event indicating the execution of another atomic task. Therefore, the first detected task is finished for the first time.
3. Additional information request: The knowledge provider sends an event to the context management component to acquire additional knowledge on the situation as needed.
4. Context information provision: The context management component sends the requested context information to the knowledge provider.
5. GI acquisition: The knowledge provider acquires the GI. This includes a query to the knowledge store to get items for the GI.
6. GI return: The knowledge store sends the GI back via an event.
7. GI provision: The context management component processes the event and sends it to the GUI for display. The GI is displayed at a special area in the GUI and is notified about the new GI being available.
8. GI removal: The GI is removed. This is done in two cases: Either the human indicates the consumption of the GI or the defined life time of the GI ends.

**Figure 12-7: Asynchronous dynamic optional task post-checklist provision**

Example 12-8 illustrates the automated knowledge management support by the CPM framework utilizing checklists.

Example 12-8 (Automated knowledge management support):
Figure 12-8 illustrates what the various actors in the scenario do and how they cooperate to achieve automatic knowledge management support via checklists.
(A.) During project execution, humans may add knowledge to the knowledge store, e.g., when encountering problems and finding solutions for them. An example of such knowledge may be a hint to use the Model-View-Controller (MVC) pattern for GUI development. Humans can tag this knowledge to support its later discovery by humans or any automated system. Examples of tags on that knowledge include ‘junior’ to indicate applicability for junior engineers or ‘backend’ or ‘frontend’ to relate them to a specific implementation area.

(B.) Project execution is managed and governed automatically by the CPM framework. This is applied by enacting different workflows belonging to the projects’ SE process. Examples of activities governed that way include ‘Implement Solution’, where new source code is developed, or ‘Run Developer Test’ where source code is tested by the SE engineer.

(C.) These workflows can be annotated, e.g., by a process engineer at specific points to use GIs (e.g., requirements or testing checklists). The GIs can be easily pre-defined in the knowledge provider: On one hand, they can be explicitly and statically pre-defined; on the other, all knowledge entered can be tagged to be dynamically useable for GIs.

(D.) The CPM framework continuously detects new facts about the current situation and stores them in the context base. This is enabled by a set of sensors integrated in various SE tools that automatically provide information about tool and artifact usage. An example of such a detected event is the modification of a source code artifact in an IDE. Utilizing this situational information, dynamic GIs are supported - workflows can be annotated to include GIs at certain points, but these do not have to be predefined. Such a dynamic GI is automatically generated by the CPM framework based on information of the current situation as, e.g., the skill level of the human or the time and quality constraints of the project using tags on knowledge in the knowledge provider. For example, a junior engineer working at the frontend of an application could be provided a pre GI containing the aforementioned item concerning the MVC pattern when starting his ‘Implement Solution’ activity.

12.3. Discussion

There are many approaches targeting at knowledge management support for SE processes. [BjDi08] presents a literature study about knowledge management in SE. Therein, the authors describe two kinds of approaches to knowledge management: technocratic schools rely largely on information or management technology, whereas behavioral schools focus more on organizational or strategic aspects of the implementing company. In the following, approaches belonging to the technocratic schools are discussed, as they are similar to our approach.

[KuJe04] presents a study of the usage of a process-oriented knowledge management tool in a small-to-medium-sized software development company. That tool allows for web-based documentation and support for the SE process model. The study showed that the tool was accepted by humans and really supported them.
[BWT04] describes an approach to develop, retrieve and reuse management knowledge and experience concerned with SE risks. The approach incorporates the modeling of risk archetypes and scenarios to model risk impact and resolution strategies and to provide reusable project management knowledge.

The knowledge dust and pearls approach [BCL+01] aims at facilitating the application of a so-called experience base. The latter shall contain knowledge about experiences that have been analyzed and organized in packages. The approach combines both short- and long-term features in knowledge creation and sharing. It shall provide low-barrier access to knowledge and support the initial creation of it.

Outside the SE domain, there have been various efforts seeking to support knowledge management. [Liao03] presents a study reviewing a multitude of approaches to knowledge management systems. These are classified in different areas: knowledge-based systems, data mining systems, information and communication technology, database technology, modeling, and expert systems providing decision support. An example stemming from the data mining area is presented by [NSIH02]: By extending a data warehouse, a knowledge warehouse is built that shall facilitate capturing as well as retrieving and sharing knowledge.

As opposed to the above approaches, the presented approach not solely focuses on the acquisition, storage, and organization of knowledge. In particular, it provides a holistic solution that automates the provisioning aspect in the knowledge lifecycle, strongly focusing on the context-sensitive and process-oriented provisioning of knowledge to the humans.

12.4. Summary

Providing knowledge-based support for the operational process in the dynamic SE domain is challenging. In this area, this chapter contributes an approach for connecting and automating knowledge and process management. Semantic technology is used as a link between automatically gathered context information, knowledge resources, and process enactment. Thus, it becomes possible to dynamically assemble knowledge relevant to the executing human and to automatically and seamlessly integrate this knowledge with the humans’ current workflow.

On one hand, the CPM framework can automatically guide the SE process and obtain contextual information regarding the SE project (cf. Chapter 7). On the other, this approach encompasses a set of features to explicitly support knowledge management: A knowledge store is integrated as well as GUIs for collecting and managing knowledge. This supports the human while entering knowledge and enables him to configure the way in which that knowledge shall later be automatically provisioned. Automatic access to the knowledge store has been enabled by the CPM framework. Having knowledge being enhanced with machine readable semantics, the CPM framework can automatically gather and disseminate the knowledge. The management of the knowledge is done via a separate active component, the knowledge provider. The latter enabled integration and management of external knowledge sources. Furthermore, the CPM framework can utilize contextual information being automatically gathered and processed, querying knowledge from the knowledge store that matches the situation of the human with properties like the human skill, the project goals, or the implementation area. Finally, the knowledge gathered for the human is seamlessly integrated with the human’s workflow. This supports the use of the knowledge, since the knowledge not only fits the human’s current needs but also does not require and cumbersome extra work to be acquired.
Part III

Evaluation
13. Technical Feasibility

In order to demonstrate the technical feasibility of the CPM concepts presented in Part II of this thesis, we implemented a proof-of-concept prototype. This chapter elaborates this prototypical implementation. First, it elicits requirements for this prototype, which emerged during the Q-ADVICE project on one hand and partly stem from case studies in two industrial settings on the other. Second, the fundamentals of the technical realization of the CPM prototype are shown. Third, the implementation specifics of the different components are discussed. Finally, different aspects like quality or knowledge management are elaborated.

13.1. Requirements

This section elicits requirements for the CPM prototype implementation. These include functional as well as technical requirements we elaborated for three use cases: Two of them stem from industry, whereas the third case was created within the project team in the Q-ADVICE project itself. As an important implementation issue, the CPM framework aims at holistic support for SE projects and SE processes. Therefore, it touches many areas and topics. Accordingly, several proof-of-concept prototypes were created to demonstrate the applicability for the concepts developed. Selected functions have been further elaborated to be able to use them in industrial settings (cf. Chapter 14).

13.1.1. Functional Requirements

Within the industrial application cases, specific requirements concerning different base functionalities of the CPM prototype emerged. Taking these into account, functional requirements for the CPM prototype were elicited. The most basic one was the ability to provide automated workflow guidance to humans. Facilitating this, a simple GUI seamlessly integrated into everyday work became necessary. Each of the application cases involved a different process model. More precisely, three process models were required to be integrated: Scrum, the V-Model XT, and the Open Unified Process (OpenUP). Our practical experiences have further shown that software quality support and guidance is important for many companies. Based on this information, software quality measures were desired to be automatically proposed to the developers to improve the state of the source code. Another area of interest was knowledge management: They desired facilities enabling them to easily collect and manage knowledge. Finally, they wanted the CPM prototype to automatically generate checklists out of that knowledge and propose them to the developers. Table 13-1 summarizes these requirements.

<table>
<thead>
<tr>
<th>Req ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R:ProtoFunc:WFguid</td>
<td>Basic workflow guidance</td>
<td>Basic workflow guidance shall be in place.</td>
</tr>
<tr>
<td>R:ProtoFunc:GUI</td>
<td>Simple GUI</td>
<td>A simple GUI for enactment shall be in place.</td>
</tr>
<tr>
<td>R:ProtoFunc:Proc:Scrum</td>
<td>Scrum implementation</td>
<td>The Scrum process shall be implemented within the CPM prototype.</td>
</tr>
<tr>
<td>R:ProtoFunc:Proc:VMXT</td>
<td>VM-XT implementation</td>
<td>The VM-XT process shall be implemented within the CPM prototype.</td>
</tr>
<tr>
<td>R:ProtoFunc:Proc:OpenUP</td>
<td>OpenUP implementation</td>
<td>The OpenUP process shall be implemented within the CPM prototype.</td>
</tr>
<tr>
<td>R:ProtoFunc:Qmaware</td>
<td>Quality awareness</td>
<td>Awareness about the source codes state shall be gained with static analysis tools.</td>
</tr>
<tr>
<td>R:ProtoFunc:QMintegrate</td>
<td>Quality measures</td>
<td>Quality measures based on the measurements shall be integrated into the humans processes</td>
</tr>
</tbody>
</table>

1 This work originated from the Q-ADVICE project (cf. Chapter 1.2).
13.1.2. Technical Requirements

In addition to the functional requirements, technical requirements emerged that are grounded by the nature of a research project: As the CPM framework is a research prototype, many of its functions were implemented for a first test. However, to foster adaptability and flexibility, components should be easily exchangeable. As the CPM prototype should be run directly at companies’ sites, a certain level of robustness was required to limit maintenance efforts. Finally, as the CPM framework is a prototype intended for industry use, an important requirement was to use applicable technologies that bear a reasonable level of maturity. These requirements are summarized in Table 13-2.

<table>
<thead>
<tr>
<th>Req ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R:ProtoTech:Extend</td>
<td>Extendable prototype</td>
<td>The prototype shall be extensible and easily changeable.</td>
</tr>
<tr>
<td>R:ProtoTech:Exchange</td>
<td>Exchangeable components</td>
<td>Components shall be easily exchangeable.</td>
</tr>
<tr>
<td>R:ProtoTech:Robust</td>
<td>Robust execution</td>
<td>Technology shall be robust running at company site.</td>
</tr>
<tr>
<td>R:ProtoTech:AppTech</td>
<td>Applicable technologies</td>
<td>Applicable technologies shall be used that facilitate ‘real execution of the CPM prototype’ in a particular team’s development environment</td>
</tr>
</tbody>
</table>

13.2. Extending an Existing Architecture

The CPM framework incorporates a large set of modules and functions. An implementation of all of them from scratch would not have been possible within the scope of a research project. Therefore, the basic idea was to prototypically realize the CPM framework based on an existing architecture that incorporates suitable technologies. This architecture was provided by the Context-aware Software Engineering Environment Event-Driven framework (CoSEEEK) [ObSc07, Ober10] that was extended within the scope of this thesis’ project. In the following, design and architecture decisions for the prototype implementation based on CoSEEEK are elaborated, followed by a description of the implementation.

13.2.1. Design and Architecture Decisions

CoSEEEK had already incorporated a set of suitable technologies. However, the architecture had to be comprehensively revised and a set of technologies had to be exchanged to be able to realize a proof-of-concept prototype for the CPM framework. This sub-section reviews these technologies in place and discusses design and architecture decisions for the CPM prototype:

- For process enactment, the AristaFlow BPM suite [DaRe09, LRD10b, DRR+10] was used.
- Context management was integrated using semantic web technologies. This included the storage of information based on an OWL ontology as well as the use of the Bossam reasoner [JaSo04].
- A component-based architecture was implemented in terms of different modules that were started and stopped by a set of agents. Communication was realized by an implementation of the tuple space paradigm [Gele85] on top of the XML database eXist [Meie09].
- Basic quality management was implemented by transforming reports from the measurement tool PMD [Cope05] and assigning checklists to humans shown on a web page.
- With the framework Hackystat [John07], basic information was detected from the Eclipse IDE. This information was aggregated by an event pattern in the CEP (Complex Event
Processing) [Luck01] tool Esper [Espe15]. The information was used to govern an XOR decision in a workflow.

- Human activity management was implemented by forwarding workflow activities to a task display plugin of Eclipse (i.e., Mylyn [KeMu06]).

Based on this information, we decided which technologies were to be exchanged, omitted, or kept. In the following, the design and architecture decisions are discussed:

- WfMS: As process management technology, we selected the AristaFlow BPM suite [DRR+10, RDRL09, LKRD10]. AristaFlow features contemporary technology [RDR+09, Kreh14], enables dynamic adaptations to running workflow instances [ReDa98, DaRe09], provides correctness guarantees for enactment and adaptation [RRD03], is mature enough for industrial use [DaRe09], and provides full API access to all its functions [DRR+10]. Therefore, it was decided to use AristaFlow and to expand its usage to exploit more of its capabilities.

- Context management: Semantic web technology and, in particular, the web ontology language (OWL) offers ways of effectively organizing knowledge, incorporates well elaborated logical capabilities, is standardized, and is mature enough for being used in industry. Furthermore, it is standardized by the W3C and offers massive tool support [Fell13]. Therefore, semantic web technology was used for the CPM implementation. Especially, the logics based on the OWL-DL (description logics) profile were selected because it combines computational efficiency with high expressive power [Fell13]. No other types of logic are integrated since their implementations are mostly not mature enough and solely semantic web technology offers a proper combination of this logic, data organization capabilities and practical applicability. Furthermore, semantic web technology offers powerful languages for executing queries and rules. We applied both of them, SPARQL (SPARQL Protocol And RDF Query Language) [PrSe06] and SWRL (Semantic Web Rule Language) [WWW04c], in various occasions to achieve simplifications, foster flexibility, and enable workarounds for implementation issues.

For a comprehensive discussion of semantic web technology, see [KHKRS08, HKR11]. As both stability and sustainable execution were crucial, we decided to stick to mature implementations, i.e., Protégé [NCF+03] was used as the most mature and widely applied ontology editor. Furthermore, Bossam was replaced by Pellet [SPG+07] as the latter is the most prevalent reasoner for Java. Furthermore, it was decided to stick to OWL 1 and to not integrate beta versions of tools implementing OWL 2.

- Context gathering: Due to lack of alternative extensible sensor frameworks, Hackystat was used as sensor framework. Additional sensors (e.g., for Microsoft tools) were implemented. Further, Esper was kept being easy to use for applying simple event patterns to gather higher-level events from multiple low-level ones.

- Human activity management: Human activity management including various types of tasks, checklists, and other information, was integrated in the semantic web implementation of the context management component.

- Configurable automatisms: To enable the specification and enactment of configurable automatisms (cf. Chapter 6) rules processing technology was integrated. As rule engine, JBoss Drools [Brow09] was chosen as a simple and flexible way to execute recurring automatisms. It is applicable for different purposes where such processing should not burden the higher level knowledge implementation with semantic web technology.

- Loosely-coupled architecture: As the implementation was subjected to many changes, it was decided that the best way to react on these changes was to keep a loosely-coupled modularized architecture. The ability to easily exchange, add or remove components was well supported by the event-based communication over a tuple space [Gele85]. To suit industrial requirements, we exchanged the agent-based module management with the mature and robust OSGI framework [OSGI15].

### 13.2.2. CPM Implementation

This section reviews the architecture of the prototype CPM implementation. The different technologies are illustrated in Figure 13-1 and explained in the following:
The programming language chosen for the implementation is Java. The component architecture is implemented using the OSGI framework [OSGI15], which provides the environment for implementing the different components and for managing their lifecycle.

As aforementioned, the technologies for event management have been kept. However, the set of sensors has been extended: We added sensors for IDEs (Microsoft Visual Studio and Team Foundation Server) and bug trackers (Mantis) as well as a generic sensor applicable to multiple tools. Within industrial application cases a sensor calibration was conducted where developers documented their tasks while sensors where initially running to validate that the latter provided correct data.

For the context management component, mature and sustainable technology was required. However, in this area, there exist many research prototypes and technologies not suitable for production use. Therefore, we chose the reasoner Pellet [SPG+07] due to its stability, performance, and maturity. In addition, it offers capabilities for executing SPARQL [PrSe06] queries and SWRL [WWW04c] rules that were both needed for querying and rule execution in the ontology. Furthermore, Pellet offers a ‘DL safe rule execution’ feature [SPG+07] that prevents rule execution from interfering with the description logic of OWL DL [McVa04, WWW04a]. For programmatic access to the concepts in the ontology, the Jena framework [McBr02] was applied. To be prepared for big ontologies with great numbers of individuals, the ontology management was switched from an OWL XML file to a database based solution offered by the Protégé [NCF+03] ontology editor.

The process management implementation relied on the AristaFlow BPM suite [DRR+10], which offers comprehensive correctness guarantees for workflow enactment. Furthermore, it is the only
reliable and mature existing framework enabling dynamic adaptations to running workflow instances, while still guaranteeing correctness of the modified ones [DaRe09, ReDa09, DRR+10]. The In-Process component (cf. Chapter 6) could be implemented in AristaFlow with the so called ‘Environment’ that can be used as an activity implementation in an AristaFlow workflow in a convenient and sustainable fashion. That way, a running workflow instance transfers control to the environment when an activity containing the latter is started. The customly implemented environment then starts event-based communication with the context management component and thus transfers execution control to the latter.

For implementing the rules processing component, JBoss Drools [Brow09] was chosen as it is a mature framework that allows for convenient and sustainable definition and execution of rules. For the agent-system component the Jade framework [BPR99] was chosen. The latter is mature and complies with the general agent definition FIPA [BrNi98]. For knowledge management, the Semantic MediaWiki [KVV06] was integrated, due to its maturity, prevalence, and good applicability. To enable knowledge queries to the wiki, SPARQL queries were used enabled via the ‘SparqlExtension’ plugin and the SPARQL processor Joseki [McBr02]. That way, a seamless integration of knowledge management and context management with semantic web technology was made possible.

The different GUIs were realized with HTML and PHP so that GUIs could be used with a web browser. However, to enable better integration in the human’s everyday work, we wanted to integrate the execution GUI directly into the developers IDE. Therefore, a plugin for eclipse and a browser-based integration for Visual Studio was created. The GUIs were all realized in a uniform fashion with the jQuery PHP framework [BiKa08].

13.3. Software Engineering Process Enactment with CPM

This section discusses implementation aspects relating to the basic process enactment concept of the CPM framework. It illustrates one of the major implementation issues: the coordination of the different involved components to enable context-aware process management.

13.3.1. Technical Aspects

We first describe how the process and context management components interact utilizing a basic workflow enactment example as illustrated in Figure 13-2.

Figure 13-2 illustrates a basic workflow enactment use case: a workflow gets instantiated, an activity of the latter is activated, and human task information from the activity is delivered to the human. The latter then finishes the activity that gets terminated afterwards. For this example, we assume the workflow instance to be executed as part of the process executed in an SE project. Thus, it is the sub-
workflow instance of another workflow instance and the workflow start event happens within the context management component when the work unit connected to the sub-workflow instance gets activated. On account of this event, the individual concepts for the sub-workflow instance get created within the context management component. Then, it distributes an event over the tuple space to the process management component to start a new workflow instance (cf. Figure 13-2 (1), (2), (3)).

When the workflow is instantiated and its first activity becomes active, the environment associated to that activity receives control. It delivers information about the activity to the context management component so that the workflow extension concepts (e.g., the work unit container) can be connected with the workflow instance (cf. Figure 13-2 (4), (5)). Second, it delivers information about activation of that specific activity to this component (cf. Figure 13-2 (6), (7)). The latter then delivers human task information related to the activated activity to the GUI component to inform the human (cf. Figure 13-2 (8), (9)). After processing the activity, the human finishes it and information about this event is delivered from the GUI component to the context management component (cf. Figure 13-2 (10), (11)). The states of the individual concepts are then adapted and an event is delivered to the environment of the related activity in the relating workflow instance (cf. Figure 13-2 (12), (13)). Finally, the activity terminates and the workflow instance continues its course.

We now provide details about the implementation of the conceptual framework in the OWL-DL ontology. We have modeled both template and individual concepts in the ontology, as ontology concepts. The latter (as e.g., the work unit container template or the work unit container), in turn, can be instantiated with ontology individuals (e.g., for a specific work unit container template or a specific work unit container). As discussed, one of the advantages of an OWL-DL ontology is its logical and reasoning capabilities. Thus, we had the option to directly implement many of the consistency checks defined in Chapter 7. However, due to the open world assumption of this technology (cf. [KHKR08, HKR11]), it was not possible to detect or restrict the absence of a specific concept within a specific property. Therefore, we implemented corresponding consistency checks programmatical with the Jena framework. A second workaround to be able to utilize SWRL rules for simple consistency check implementation was the introduction of a property ‘problem’ indicating that such a rule was violated. Thus the SWRL rules could simply set this property to ‘yes’ if certain facts were in place. In the following, we show the implementation of the consistency check which could be realized with SWRL. For brevity, we only show excerpts of this implementation for each of the topics we have covered with CPM. For repeatable and omittable activities, we show one of the consistency checks that could be implemented in SWRL, as depicted in Listing 13-1.

<table>
<thead>
<tr>
<th>Listing 13-1 (Repeatable and omittable activities)</th>
</tr>
</thead>
</table>
This listing depicts the implementation of a consistency check for repeatable and omittable activities (cf. Definition 7.5).

\[
\text{basisWUT}(?x, ?y) \land \text{omittable}(?y, "FALSE") \land \text{finalized}(?x, "TRUE") \land \text{WUstate}(?x, "Created") \rightarrow \\
\text{problem}(?x, "yes")
\]

### 13.3.2. User Interfaces

This section briefly introduces the GUI component we developed to directly support SE project participants and, first and foremost, SE engineers. Therefore, we opted for developing a simple GUI that can be accessed via a web browser. However, in the course of the project, we realized that a more direct integration into the developers’ daily workflow is beneficial. Therefore, we adapted the GUI to be available either in a browser or directly in the IDEs of the developers as plugins for two prevalent IDEs, i.e., Microsoft Visual Studio and Eclipse. Figure 13-3 shows the integration for Eclipse.
Figure 13-3: Process enactment GUI integration

Figure 13-3 shows the IDE Eclipse with the CPM prototype GUI integrated in the upper right section. We have built the GUI to be small and uninvasive to be seamlessly integrated into the IDEs. Figure 13-4 introduces details about the different features of the GUI.

The CPM prototype GUI (cf. Figure 13-4), is separated into two sections: the upper section hosts various types of information organized in different tabs: general context information, knowledge provided to the humans, SE issue processing information, and settings for the GUI. These topics will be covered in the succeeding sections. In the lower section of the GUI, basic task information is shown to the human. This not only involves information about the processed activity, but also on the
superordinate assignment it belongs to. Furthermore, the current project, iteration, and activity group are provided to the human. In addition, the GUI shows activity steps used to support the current activity to the human as well as the currently detected atomic task. If the detection was not accurate, the human may simply select another task recorded by the CPM prototype. Finally, according to the planned activity the human is processing, the GUI shows the currently activated activity, which he can start and stop. It further shows the next possible upcoming activities, from which the human can simply choose one. This is facilitated by the concepts for abstraction of internal workflow logic presented in Chapter 7.

### 13.4. Software Engineering Workflow Adaptation Aspects

This section briefly describes how the adaptation of a running workflow instance is technically realized by the context and the process management components utilizing the adaptation capabilities of the AristaFlow BPM suite. In order to adapt a running workflow instance in AristaFlow, in most cases, it has to be suspended from enactment to apply the adaptations. This cannot be done if an activity in the instance is still active. However, in most cases, at the time the adaptation is triggered, an activity in the target workflow instance is active. Therefore, AristaFlow incorporates facilities to schedule the suspension of the workflow instance from enactment. The feature is called ‘soft suspend’ and gives a signal to the instance to suspend itself immediately after the running activity is finished. Thus it is even possible to insert a new activity right after that activity. The procedure to execute this is shown in Figure 13-5.

![Figure 13-5: AristaFlow workflow instance adaptation procedure](image)

The procedure depicted in Figure 13-5 starts with an event indicating that an adaptation to a workflow instance shall be conducted. An example is the integration of a software quality measure (cf. Chapter 9). On account of this, the context management component first creates necessary concepts in the ontology, like new work units. Then, the context management component sends a ‘soft suspend’ event to process management, causing AristaFlow to do a soft suspend on the respective workflow instance. Thus, the instance will be automatically suspended right after the currently running activity is finished.

The procedure continues when the human processing the current activity indicates its completion. On account of this event, the context management component will distribute an event to the AristaFlow environment being active in the current activity in the current workflow instance, causing the relating activity to terminate. Thus, the relating workflow instance will automatically suspend itself and the context management component can deliver the concrete adaptation information to the process.
management component that, in turn, will issue the concrete adaptation actions using the AristaFlow API.

13.5. Declarative Software Engineering Workflow Generation

This section elaborates on technical and human-related details relating to the declarative workflow approach for extrinsic workflows (cf. Chapter 8). It shows how the involved components interact to enable just-in-time contextual creation of a workflow matching an SE issue. Further, it presents extensions to the concept from Chapter 8 to enable its implementation in the context management component.

13.5.1. Technical Aspects

First of all, we briefly illustrate the communication of the modules when enacting declaratively specified workflows. Then, the realization of the concepts developed in Chapter 8 is illustrated and explained. This includes the implementation of consistency checks and further implementation details, as, e.g., algorithms for different purposes.

Declarative Workflow Enactment

Figure 13-6 goes into detail about the technical realization of extrinsic workflow enactment using declaratively specified workflows as developed in Chapter 8. It depicts the sequence of actions and events performed and delivered by the different components to enable this specific function.

![Figure 13-6: Concrete procedure realization for extrinsic workflow enactment](image)

The use case starts with an ‘ad-hoc workflow event’ that may come from a human or a sensor. That event contains information about the kind of issue and the human that shall execute the workflow instance. This information is utilized for selecting an appropriate case template (cf. Chapter 8). The event is then automatically distributed to the context management component (cf. Figure 13-6 (2)), which creates the required concepts in the ontology and distributes an event to the process management component to instantiate a workflow skeleton based on the template of the selected case (cf. Figure 13-6 (3, 4, 5)). The first activity of each case is ‘Analyze Issue’ to let the human gain knowledge about the case and provide information about process and product properties to the CPM prototype via the GUI. To enable this communication with the human, the environment in this activity in the workflow instance delivers information to the context management component (cf. Figure 13-6 (6, 7, 8, and 9)) as already explained for Figure 13-5. The latter then distributes information to the GUI to enable the human to provide information about the properties of the issue (cf. Figure 13-6 (10, 11, 12, and 13)). After receiving the information, it applies the concepts and algorithms presented in Chapter 8 to select the appropriate activities for this situation and creates the related concepts. Then, it
distributes an event to the process management component to initiate the required adaptation of the workflow instance (cf. Figure 13-6 (14, 15, and 16)).

Declarative Workflow Specification

Figure 13-7 depicts the structure of ontology concepts used for implementing the declarative workflow modeling and generation concept. In the following, implementation details will be explained. This, in turn, will be followed by a more detailed description of the implementation of the main concepts and the modeling conditions defined in Chapter 8.

Figure 13-7 shows the basic concepts of the CPM framework for SE issue processing. As introduced in Chapter 8, we apply a case concept for SE issue processing. This concept wraps all necessary data of the use case as contextual information necessary and the work unit container and workflow used for enacting the use case. Figure 13-7 further includes concepts added for the implementation and marked in grey. We briefly explain the interplay of the concepts. The work unit container has a relation to work units, and the work unit container template has a relation to work unit templates. As the declarative container (template) is a specialization of such a container, it has been realized as a subconcept of the work unit container (template). A declarative container has work units being the equivalents of the activities in the workflow instance that will be generated. It further features building blocks (with their sub-concepts parallel, conditional, loop, sequence, and activity) representing the subset of building block templates of the corresponding declarative container template that are selected for the specific case in which the declarative container is used.

Figure 13-7 further shows the case and case template concepts used to represent a case for the enactment of a workflow. During the modeling of a case template, the latter is connected to one or more property templates to distinguish, for which situations the building block templates contained in the case template apply. To model how these properties of a concrete situation will be determined, the property influence, a new concept, has been added. The latter is used as a super-concept for all factors used to determine the properties like skill level or product category. When a case comes to enactment, a case concept is created, with a set of properties defined by the generated case template. These properties’ values will then be computed using concepts attached to the case that have relations to sub-concepts of the property influence concept. For example, the case will have an assigned human that, in
turn, will have one or multiple skills based on skill templates. The relations of the latter to the property template will be used to determine the actual value of the property.

For defining the structure of interconnected activities of a case, the building block template, declarative container template, and related concepts are utilized, as explained in the following. The building block template has various sub-concepts: For defining the structure of activities, the conditional template, parallel template, loop template, sequence template, and activity template are utilized. All of these concepts, except the activity template, have asserted sub-concepts enabling the reasoner to automatically classify them as inconsistent if they violate any of the conditions defined in Chapter 8. How these sub-concepts are defined will be explained as part of the description of the respective concepts later in this section. All conditions have been implemented in the ‘inconsistent’ concepts, allowing for the proper classification of inconsistencies.

A number of additional template concepts have been added as well: The building block with decision as sub-concept of the building block template, which has an additional property (‘decisionAlternative’) connecting it with a user decision alternative (cf. Chapter 7). This concept is used for an easy implementation of the conditional template. That way, building block templates can already have the required information needed for building conditionals including their different decision alternatives.

To aid the implementation within the ontology, we not only introduced a set of concepts but also properties and property hierarchies (cf. Figure 13-8). This includes a new super-concept of the declarative container template, the declarative modeling element. The latter incorporates a super-property of the specific building block properties (e.g. the ‘parallelExecBBset’). The new declarative modeling element enables new properties (e.g. ‘containedIn’) for abstractly indicating the containment relations between all elements as well. These properties, in turn, enable transitive closures for easily determining which elements are incorporated in one container.

Figure 13-8: Structuring property relations

Figure 13-7 depicts another concept, the building block type. It favors the creation of a building block library, which groups all building block templates of one type. Further, it incorporates additional information about the building block template to aid the human in retrieving and re-using it.
Building Block Concepts Implementation

We now go into detail about selected concepts with important implementation specifics, starting with the building block template. The latter is the central concept for declarative workflow modeling. It is the super-concept for the other modeling elements from the simple activity to the other structuring elements like the loop. It features properties for creating a structure of building block templates and thus implements the basic constraints introduced in Chapter 8. These properties are implemented using the modeling capabilities the ontology provides, like inverse or transitive properties. For example, the property ‘mutexBBset’, which implements the ‘mutualExclusion’ constraint, is defined to be symmetric and thus inverse to itself. That way, when a relation ‘a mutexBBset b’ is created, the inverse ‘b mutexBBset a’ is created as well.

As aforementioned, we have added a set of ‘inconsistent’ concepts for classifying certain building block templates that violate the checks defined in Chapter 8. We will exemplify how this classification is applied. To make the check implementation easier and the SWRL rules more compact, we add sub-concepts of the building block template that classify them as having certain properties (cf. Listing 13-2). An example of this is a building block template that has parallel connections to others (BuildingBlockWithParallel). In this section, we will discuss the implementation of these specific building block templates in the ontology. Further, we will show how they are utilized for other classifications by showing the implementation of the classification of an inconsistent sequence template.

Listing 13-2 (Building block template)

```
BuildingBlockWithParallel:
BuildingBlockWithParallel = BuildingBlockTemplate ∧ parallelBBset ≠ Ø

BuildingBlockWithSuccessor:
BuildingBlockWithSuccessor = BuildingBlockTemplate ∧ successorBBset ≠ Ø

BuildingBlockWithPredecessor:
BuildingBlockWithPredecessor = BuildingBlockTemplate ∧ predecessorBBset ≠ Ø

BuildingBlock_Start:
BuildingBlock_Start = BuildingBlockTemplate ∧ predecessorBBset = Ø ∧ BuildingBlockWithPredecessor ∉ parallelBBset

BuildingBlock_End:
BuildingBlock_End = BuildingBlockTemplate ∧ successorBBset = Ø ∧ BuildingBlockWithSuccessor ∉ parallelBBset

BuildingBlockInconsistent:
BuildingBlockInconsistent = BuildingBlockTemplate ∧ (|predecessorBBset| ≥ 2 ∨ |successorBBset| ≥ 2 ∨ problem = "yes")

parallelBBset(?b1, ?b2) ∧ successorBBset(?b1, ?b2) → problem(?b1, "yes")

|BuildingBlockTemplate.containedIn| = 1
```

The implementation of the sub-concepts of the building block template is shown in Listing 13-2. Most conditions are implemented directly in OWL. However, to be able to implement them all, an additional property and SWRL rules have been applied. Only one sub-concept needed a programmatic workaround due to the open world assumption – the unconnected building block. Property ‘problem’ indicates that there is a problem with a concept and thus the concepts will be classified as inconsistent. This property is initialized with value ‘no’ and written by SWRL rules, as, for example, for the BuildingBlockInconsistent concept: a rule sets the property for a building block template having both a successor and a parallel building block template as this is not permitted (cf. Definition 8.3). The fact that a building block template shall not be contained in two other concepts is covered by the property
'containedIn' (cf. Figure 13-8), whose cardinality is restricted to exactly one for the building block template. Listing 13-2 depicts the implementation of different sub-concepts of the building block template.

The sequence template (cf. Chapter 8) features only one special property, the ‘sequentialBBset’ containing sequentially connected building block templates. It has a sub-concept utilized for implementing its consistency checks. For a discussion of unwanted cases these checks avoid we refer to Appendix B. The implementation of this concept is shown in Listing 13-3. Check b) of the inconsistent sequence is not shown here as we have implemented it programmatically (a cardinality constraint was not possible as ‘sequentialBBset’ already had a restriction from its super-property).

Listing 13-3 (Inconsistent sequence)

```
Check a):
InconsistentSequence \equiv SequenceTemplate \land BuildingBlockWithParallel \in sequentialBBset

Check c):
successorBBset(?a, ?b) \land organizedBBset(?s1, ?a) \land organizedBBset(?s2, ?b) \land differentFrom(?s1, ?s2) \rightarrow problem(?s1, "yes")

Check d):
BuildingBlock_Start(?b1) \land BuildingBlock_Start(?b2) \land differentFrom(?b1, ?b2) \land sequentialBBset(?s, ?b1) \land sequentialBBset(?s, ?b2) \rightarrow problem(?s, "yes")

Check e):
BuildingBlock_End(?b1) \land BuildingBlock_End(?b2) \land differentFrom(?b1, ?b2) \land sequentialBBset(?s, ?b1) \land sequentialBBset(?s, ?b2) \rightarrow problem(?s, "yes")
```

To make an easy implementation in the ontology possible, the checks for the inconsistent sequence have also been partly implemented by SWRL rules. Check a) has been directly integrated into the definition of the inconsistent sequence. Check c) has been implemented using the ‘organizedBBset’ property in a way that, if two sequentially connected building block templates are contained in different other building blocks, a problem will result. Checks d) and e) utilize sub-concepts of the building block template shown in Listing 13-2.

The loop template (cf. Chapter 8) features multiple special properties, as it shall not only be able to contain another building block template to be executed repeatedly, but also contain elements enabling the decision when the contained building block template is to be re-executed and the enactment shall proceed. This is enabled by properties ‘loopAlternative’, ‘proceedAlternative’, and ‘repeatableBB’. The latter contains the building block template, whereas the first two hold references to user decision alternatives (cf. Chapter 7). These decision alternatives govern the decisions in a workflow. They deal with discrete values for each alternative. For realizing the loop template, these alternatives are utilized similar to standard WfMS: One decision is attributed to one variable. In this case, this is the variable governing the loop enactment. However, WfMS (as AristaFlow) implement XOR decisions using value ranges and require a complete coverage of the whole possible value range. To be able to implement decisions modeled with user decision alternatives in such a WfMS, we automatically extend these values to value ranges.

As opposed to the loop template, which only contains one building block template and two user decision alternatives, the conditional template may contain an arbitrary number of building block templates, each with a relating user decision alternative, and an additional user decision alternative if the enactment of the contained building block templates shall be optional. For implementing the conditional template (cf. Chapter 8), a separate sub-class of the building block has been created to model the cohesion of a building block template and a user decision alternative in property ‘conditionalBBset’. That sub class is called BuildingBlockWithDecision and is extended by property...
Technical Feasibility

‘decisionAlternative’ containing a link to a user decision alternative. In addition to that, the conditional template also features a property ‘optionalAlternative’ that can contain a user decision alternative. If the latter is present, the building block templates contained in conditional template will be optional. For implementing that in a WfMS, we extend the values to value ranges similar to the loop template.

Auto Completion Feature Implementation

The implementation of the auto completion feature (cf. Chapter 8) is straightforward using SWRL rules (cf. Listing 13-4). Some properties of OWL could be exploited to support the implementation: As the successor and predecessor properties are defined to be inverse to each other, scheme b) from Definition 8.4 could be omitted. Furthermore, the rules for the transitive closures didn’t have to be explicitly implemented, as the three inferred properties are simply defined as transitive.

Listing 13-4 (Auto completion)
This listing depicts the implementation of the auto completion feature (cf. Definition 8.4).

| successorBBset(?a, ?b) → |
| inferredSuccessorBBset(?a, ?b) |
| parallelBBset(?a, ?b) → |
| inferredParallelBBset(?a, ?b) |
| inferredSuccessorBBset(?x, ?y) ∧ inferredParallelBBset(?y, ?z) → |
| inferredSuccessorBBset(?x, ?z) |
| inferredSuccessorBBset(?a, ?b) ∧ inferredParallelBBset(?a, ?c) → |
| inferredPredecessorBBset(?a, ?c) |
| inferredPredecessorBBset(?b, ?a) ∧ inferredParallelBBset(?b, ?d) ∧ inferredPredecessorBBset(?d, ?c) → |
| inferredParallelBBset(?a, ?c) |
| inferredSuccessorBBset(?a, ?b) ∧ inferredParallelBBset(?a, ?c) ∧ inferredSuccessorBBset(?c, ?d) → |
| inferredParallelBBset(?b, ?d) |

13.5.2. User Interfaces

To make the approach usable and testable a set of GUls was created for declarative workflow modeling. These were not intended to be production ready, but served as a proof-of-concept implementation. Figure 13-9 shows a selection of these GUls.

Different GUls for different parts of the concept exists. On the left side of Figure 13-9, there is a navigation pane allowing the humans to access the different areas. The GUls for declarative workflows enable the easy creation of context properties, activities, building blocks, and cases (cf. Chapter 8). For each of them, a screen in the GUI enables the creation, editing, and deletion. For all entities, a list of all existing ones can be displayed offering to delete, create or edit an item. As based on these template concepts concrete workflow instances can be executed, real editing of the templates is forbidden. In that case, editing means creating a copy of the selected item and editing the new one. The distinction between the template and individual concepts does not concern humans. Therefore, the concepts are not called templates in the GUls to not bother humans with the high number of different concepts. Figure 13-9 (b) shows the screen containing the list of building block templates. From that list, the screen for editing / creating building block templates can be accessed as shown in Figure 13-9 (a). That screen allows defining a name, description, and category for the building block. The type of building block template may be selected and, according to the type, the special properties of the block. Figure 13-9 (a) shows this for a ‘sequence’: on the left, the contained activity / building block templates can be specified; on the right, the context properties to which the specified building block should apply.

228
Activities may be defined similarly as shown in Figure 13-9 (c): A name, description, and category can be defined as well as context properties to which the activity shall apply. The description and category are stored in the building block type concept, which is a super-concept for the activity as well as the building block templates. The definition of context properties is depicted in Figure 13-9 (e). For them, a name, description, and influences can be defined. The example shows the ‘Skill Level’ of the human processing the activity as influence, which is defined to enhance the context property ‘Risk’ when it is low. The definition of cases can be easily accomplished as well (cf. Figure 13-9 (d)). Besides a name and description, the human can define how building blocks or activities shall be included utilizing the four basic constraints (cf. Chapter 8).

As the creation of the workflow instances based on the defined templates is context dependent, a facility to collect such information from the human is integrated in the enactment GUI as shown in Figure 13-10.
The cases described in Chapter 8 are primarily used for covering recurring SE issues like bug fixing. The upper part of the enactment GUI shows the facilities to let the human specify process and product properties. This is realized in a simple fashion with sliders for process properties and a selection dropdown list to choose a product category for the product properties. Furthermore, the human may initiate a new issue via the GUI.

13.6. Software Engineering Coordination Aspects

This section depicts important implementation aspects regarding the coordination concept proposed in Chapter 10. In particular, it shows how the implementation solves two main challenges, i.e., how to represent the concept in the CPM framework to enable a high level of automation and how to let humans easily configure the automatisms provided by it.

13.6.1. Technical Aspects

The coordination concept comprises three aspects (cf. Chapter 10): The navigability information provision, automatic notifications, and automatic issuing of coordination activities. The former two will be shown as part of the user view, as there are no unique technical implementation specifics. The automatic activity coordination is implemented by SWRL rules and SPARQL queries. The first step, the determination of impact areas, is configurable by the human. In the following, an example for a SWRL rule configured by the human is shown. The first step of the detection procedure (cf. Chapter 10) for follow-up actions is done by a SWRL rule that takes into account the PotentialImpact, the fact that the package containing the artifact is an interface component, and the type of artifact that was processed. To enable potential impacts for the whole hierarchy of project components, a new property 'subCompSetTransClosure' has been introduced that realizes the transitive closure for the 'subCompSet' (cf. Listing 13-5).

Figure 13-10: Enactment GUI for declarative workflows
As final step of the procedure, the responsible resource is determined. This step takes the person taken for the role of the target component matching the target area (e.g., ‘Testing’). If no match is found, the responsible person for the component is taken. If that is also not defined, the responsible is searched in one of the super sections of the component. This step is executed with the generation of SPARQL queries. One of these is shown in the following. Listing 13-6 shows the simplest case, a query to get the human responsible for a specific project component.

The final step of the procedure, the selection of the appropriate follow-up activity via a set of matching properties (cf. Chapter 10) is then determined by an algorithm that is, due to its simplicity, not shown here.

### 13.6.2. User Interfaces

Simple GUIs have been created for the coordination feature. Figure 13-11 shows the one for defining coordination rules:
13 Technical Feasibility

With this GUI, humans can configure the first step of the coordination procedure (cf. Chapter 10). They can choose the type of impact, activity, artifact, and the relating section. Similarly, for configuring automatic notifications, a separate GUI has been created as shown in Figure 13-12:

![Notification Templates GUI](image)

**Figure 13-12: Modeling GUI for notification templates**

This GUI lets the human create notification templates including a source entity and a trigger event. These notifications are then automatically distributed to the human as shown in the upper section of Figure 13-13.

![GUI Example for Automatic Notifications](image)

**Figure 13-13: GUI example for automatic notifications**

The lower section of Figure 13-13 shows the navigability information the CPM prototype provides to the human including various parts of the process, such as iterations or activity groups.
13.7. Software Engineering Exception Handling Aspects

This section discusses implementation aspects related to exception handling (cf. Chapter 11). The main issues in this area were the representation of the concept and its high level of automation in the CPM framework and the creation of an easily usable configuration mechanism for it.

13.7.1. Technical Aspects

In this section, the realization of the process illustrated in Chapter 11 is described. The process involves the receipt and classification of events from the environment, the decision whether exceptions shall be raised when the events occur, and the automatic determination of a respective exception handling. To be able to execute this automatically, the received events must be classified first. Therefore, SWRL rules can be customly defined. A simple example for such rules is shown in Listing 13-7, which classifies an event relating to a source code artifact as a source code event:

Listing 13-7 (Source code event)

\[
\text{artifactType} (?\text{artifact}, \text{"Source Code"}) \land \text{relatedEntitySet} (?\text{event}, ?\text{artifact}) \\
\rightarrow \text{eventType} (?\text{event}, \text{"Source Code"})
\]

Based on different event types, various rules are possible for raising exceptions relating to these events. These rules can be configured by humans (cf. Section 13.7.2) and take into account the type of project and a quality goal of such a project. These parameters are applied to the proof-of-concept prototype. For the sake of brevity, we refrain from discussing these SWRL rules as they are similar to the ones used for the coordination concept.

13.7.2. User Interfaces

This sub-section shows two of the developed prototype GUIs related to exception handling aspects (cf. Figure 13-14 and Figure 13-15), one for creating rules for exception raising and another one for handling determination.

![CoSEEXK Configuration Web-Interface](image)

Figure 13-14: Modeling GUI for exception raising rules
The exception raising GUI allows for the creation of exception raising rules, including the selection of an event, a project type, a quality goal, and the exception type to be raised. The determination of the handling allows configuring the exception and situation, in which a handling is to be applied.

13.8. Software Engineering Quality Management Aspects

This section shows implementation aspects for the quality management concept. No special aspects like the implementation of consistency checks were applied here (for details to the automatic adaptation aspect see Section 13.4). Therefore the section focuses on the user interfaces. Figure 13-16 shows the GUIs for five concepts. For the quality management aspect, the list GUIs are shown, where the AGQM GUI is specific. This will be explained in the following.

Figure 13-16 shows the GUIs for managing the concepts related to quality management: The quality metrics, measures, tools, rules, and the AGQM configuration. The GUIs allow for the partial adaptation of the concepts in the list views. For example, the Q-Metrics GUI (cf. Figure 13-16a) allows for the selection of the related tool in that view. For these tools, a location can be configured, where reports created by them can be obtained (cf. Figure 13-16e). The Q-Rules GUI (cf. Figure 13-16c) allows defining a relation between a metric and a measure with a trigger indicating when the measure shall be proposed. Additionally, the rules may be prioritized and deactivated. The AGQM GUI unites a list view with an option to create new concepts: The human can create new quality goals by pressing the ‘Create new Goal’ button. By selecting a goal from the list, the ‘Create new Question’ button is activated, allowing for the same for the questions that will be related to the selected goal then. The same applies to the relation of questions and metrics. For a goal, a name, description, and GKPI can be defined as well as a strategy and the initial points for the agent that will be instantiated for that goal. Note that these GUIs assume a single project environment where the goals will be attributed to that project. The strategies for the agents are managed via a configuration file and comprise, for the first version, three properties: The starting bid configures the percentage of points, an agent uses for his first bid. The two other options configure how an agent raises a bid when he has lost a round or lowers the bid when he has won a round.
13.9. Software Engineering Knowledge Management Support

This section illustrates the prototypical implementation of the knowledge management aspects of the CPM framework. The knowledge collection GUI (cf. Chapter 12) is not shown here as it is implemented directly by the Semantic MediaWiki [KVV06]. The knowledge management GUI is shown in Figure 13-15 through Figure 13-17.
The knowledge management GUI consists of multiple screens embedded in the structure of the other modeling GUIs. Figure 13-17 depicts the navigation pane on the left, like the other GUIs. On the left side of the embedded knowledge management GUI, however, there is a menu providing access to basic functions (creating, editing, and browsing) related to GIs (guidance items, cf. Chapter 12). To its right, a frame is used to create a new GI with all required options (shown in Figure 13-17). There is an option for creating a private GI, enabling humans to add personal GIs that are only shown to them (e.g., something they want to consistently use or be reminded of). Besides that, the human can provide an ID, description, set of tags, and type. Finally, he can choose between creating an external or internal as well as static or dynamic GI. External GIs can only be static using the ‘Link’ specified in the GUI to gather the information to be displayed to the human. Internal GIs can be dynamic, which lets the CPM prototype forward the specified tags to the Semantic MediaWiki to gather information to be automatically compiled for the human.

Figure 13-18 shows two GUIs: first, the GI browsing functionality, where humans can apply various options and tags for querying and modifying existing GIs. Second, the figure shows a GUI called ‘Global Guidances’ that enable the humans to apply various options to connect GIs with the process. They can be attributed to an activity of a process concept and specify a frequency and a rank.
Figure 13-18: Modeling GUI for browsing guidances and attributing guidance

Figure 13-19 shows the process support GUI, which is directly integrated into the IDE of the software engineers. Note that due to screen space limitations and requirements from practical use cases, a different GUI style evolved. The lower section of that GUI gives comprehensive information about the activity currently performed, ranging from the project in which it is executed, over the assignment activity, to the atomic task currently being performed. In the upper section, an active GI of the type checklist is shown. It has three items that have to be checked and completed using the given ‘Complete’ button. Additionally, the GI can be rated, influencing its rank for future prioritization and use.

Figure 13-19: Process support GUI with example for checklists
13.10. Summary

In this chapter we have presented proof-of-concept prototypes for the different parts of the CPM framework. Moreover, a subset of these functionalities have been applied in industrial settings. The involved concepts and GUIs show that it is possible to hide much of the complexity of the concept and thus make it usable. Furthermore, we managed to implement a large portion of the correctness and consistency checks directly within the ontology. For some parts, however, workarounds became necessary. Some of them were implemented via programmatic access to the ontology with the Jena framework.

The implementation showed that much of the modeling correctness still depends on the humans. For a concept like CPM that covers many areas of SE projects that rely on human interactions, we believe that it is not possible to implement 100% fail safe and automatically enforced correctness. However, we implemented a set of conditions and were able to hide as much complexity as possible to assist humans in correct modeling. Furthermore, the main benefit of our approach comes from the seamless integration of many of these areas, innovative content-related concepts, and from enabling a holistic way of supporting SE projects.
14. Practical Application

The approach this thesis takes to evaluate the feasibility and applicability of the CPM framework comprises different parts. First, the basic CPM concepts developed in Chapter 7 have been used for different aspects of SE projects to achieve the more specific solutions developed in Chapters 8 – 12. Second, Chapter 13 showed the technical feasibility by describing the CPM prototype we have developed. This chapter now adds further parts to the evaluation: It presents the application of the CPM framework to different areas in SE and different SE process models respectively. Throughout this thesis, we have used parts of the OpenUP process to illustrate the CPM framework. We now consider process models being different from OpenUP and presenting a large spectrum of SE processes with different properties. On one hand, we consider the Scrum process, which is rather minimalistic and agile. On the other, we apply the vast and heavyweight V-Model XT. In addition, we present an exemplary application of the CPM framework to another domain in SE: software modernization projects. In particular, we show the implementation of the specific XIRUP process with the CPM framework.

In addition to the application of the CPM framework to the various SE process models, we present extended scenarios for CPM functions as, e.g., quality and knowledge management. These scenarios reconsider problems, already mentioned earlier in this thesis. The specific functions are orthogonal to the ability to implement different process models. Thus, these functionalities can be applied to all models implemented with the CPM framework. As an example, we briefly discuss the application of knowledge management guidance for software modernization projects. Finally, we present results and lessons learned from a preliminary industrial evaluation of the CPM framework.

14.1. Modeling the OpenUP Process

This section discusses the modeling of the OpenUP process based on the CPM concepts. It is organized as follows: First, a mapping of the process model concepts to the CPM concepts is shown. Then, a figure illustrates the modeling of the entire process. Finally, we describe the steps to be taken to enact such a model with the CPM framework.

14.1.1. Mapping the Process Concepts

The more abstract levels of the process model (i.e., project lifecycle, phases, and iterations) are implemented using work unit containers (cf. Figure 14-1). The same applies to the more concrete levels (activities and tasks). However, they also feature extensions by assignments and assignment activities to model the human tasks. Supportive features of the OpenUP process are mapped by activity steps and the guidance (cf. Table 14-1).

---

1 These scenarios are not a real validation as they partly reconsider situations that were already discussed at the beginning of this thesis. We integrate them amongst the other parts of the evaluation for better illustration of the applicability of our basic approach to different areas of SE projects.
Table 14-1: Concept mapping OpenUP

<table>
<thead>
<tr>
<th>OpenUP Concept</th>
<th>CPM Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project</td>
<td>Project + Work Unit Container</td>
</tr>
<tr>
<td>Phase</td>
<td>Work Unit Container</td>
</tr>
<tr>
<td>Iteration</td>
<td>Work Unit Container</td>
</tr>
<tr>
<td>Process Module</td>
<td>Work Unit Container</td>
</tr>
<tr>
<td>Activity</td>
<td>Assignment / Work Unit Container</td>
</tr>
<tr>
<td>Discipline</td>
<td>Area</td>
</tr>
<tr>
<td>Task</td>
<td>Assignment Activity / Work Unit</td>
</tr>
<tr>
<td>Task Step</td>
<td>Activity Step</td>
</tr>
<tr>
<td>Artifact</td>
<td>Artifact</td>
</tr>
<tr>
<td>Milestone</td>
<td>Milestone</td>
</tr>
<tr>
<td>Guidance / Checklist</td>
<td>Guidance / Guidance Item</td>
</tr>
<tr>
<td>Role</td>
<td>Role</td>
</tr>
</tbody>
</table>

Figure 14-1 illustrates the implementation of the OpenUP model. For the sake of readability, we show the process management concepts and their context management counterparts united in one workflow. Only a selection of the extensions (e.g., milestones, artifacts, or guidance items) are shown.

14.1.2. Process Model Enactment

This section shows the actions (cf. Appendix C) to be performed for implementing and enacting the process model.

Project start:
- Create project: All artifacts are provided or created and roles are staffed. All work unit containers, work units, and associated concepts are created as defined by the template concepts attached to the project template. The main role of the attached work unit container is the human in charge of the project who is assigned to its work units as well.
- Start work unit container: The project’s main work unit container is started.

Inception Phase:
- Create work unit container / add work unit to container dependency: During the ‘Plan Project’ activity of the ‘Initiate Project’ workflow, iterations for succeeding phases are added.

Phase / Iteration Execution:
- Create work unit container / add work unit to container dependency / Move Work Unit Dependency: During the ‘Identify and refine Requirements’ activity, new requirements can be elicited and workflow instances be created for them (‘Develop Solution Increment’, ‘Test Solution’). Then, these instances may be attached to iterations or be moved from one to another.
- Create work unit container / add work unit to container dependency: During the ‘Plan and Manage Iteration’ activity, an iteration for the current phase may be added (e.g., for the Inception phase that initially has only one iteration).
Figure 14-1: OpenUP modeling
14.2. Modeling the V-Model XT Process

This section shows how to map the V-Model XT process to the CPM concepts.

14.2.1. Mapping the Process Concepts

The V-Model XT has a set of special capabilities like the tailorings and various dependencies between activities and products (cf. Chapter 3). To enable static project-specific tailoring, the project (or project template) has an associated work unit container. For each project type or variant, a separate project template and work unit container structure can be modeled. The two types of product dependencies are mapped by hierarchical and requiring relations between project components. All project activity related items (e.g., phases, iterations or process modules) are mapped by work unit containers that may require their presence mutually. The full list of mappings is shown in Table 14-2.

### Table 14-2: Concept mapping V-Model XT

<table>
<thead>
<tr>
<th>VM-XT Concept</th>
<th>CPM Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project</td>
<td>Project</td>
</tr>
<tr>
<td>Project Type / Variant</td>
<td>Project Properties</td>
</tr>
<tr>
<td>Project Properties</td>
<td>Project Properties</td>
</tr>
<tr>
<td>Static Tailoring</td>
<td>Project defines process structure</td>
</tr>
<tr>
<td>Dynamic Tailoring</td>
<td>Add or remove Work Unit Container dynamically</td>
</tr>
<tr>
<td>Generative Product Dependency</td>
<td>Project Component requires Project Component</td>
</tr>
<tr>
<td>Content-related Product Dependency</td>
<td>Project Component: Sub / Super relations / requiring / relatedPCset</td>
</tr>
<tr>
<td>Decision Gate</td>
<td>Milestone</td>
</tr>
<tr>
<td>Phase</td>
<td>Work Unit Container / Loop</td>
</tr>
<tr>
<td>Iteration</td>
<td>Work Unit Container</td>
</tr>
<tr>
<td>Process Module</td>
<td>Work Unit Container</td>
</tr>
<tr>
<td>Process Module Dependency</td>
<td>Work Unit Container requires Work Unit Container</td>
</tr>
<tr>
<td>Activity</td>
<td>Work Unit Container</td>
</tr>
<tr>
<td>Activity Flow</td>
<td>Assignment Activity / Work Unit</td>
</tr>
<tr>
<td>Role</td>
<td>Role</td>
</tr>
<tr>
<td>Discipline</td>
<td>Area</td>
</tr>
<tr>
<td>Product</td>
<td>Project Component</td>
</tr>
<tr>
<td>Tool</td>
<td>Tool</td>
</tr>
</tbody>
</table>

The V-Model-XT features static as well as dynamic tailorings to enable a project-specific selection of process modules. Based on the dynamic tailoring, these modules can be even added or removed while enacting the process. The CPM actions for adding and removing dependencies (cf. Chapter 7) enable this kind of tailoring for work units and containers not yet completed. It is even possible to remove process modules that have already been started.

Figure 14-2 illustrates the implementation of a system development project (acquirer / supplier) of the V-Model XT. For the sake of readability, only selected process modules are detailed. For this kind of project, all mandatory process modules are shown as well as an optional process module, i.e., ‘Software Development’. The latter has been enhanced by a workflow structuring the contained activities. Project components and tools (i.e., ‘Tool References’, ‘Method References’, and ‘Work Products’) are only shown for process module ‘Logistic Elements’. The latter shows a particularity of the implementation: Due to its size, the ‘System Elements’ process module has been split up into smaller modules to enable more flexible modeling. For process modules being more cross-cutting or having to be conducted during the entire project (see ‘Project Management’ in Figure 14-2), two dependencies are applied: The asynchronous dependency from the ‘Define Project’ activity initiates the process module, whereas the synchronous dependency to the ‘Complete Project’ activity prevents the project from finishing if the respective process modules have not finished yet. As the V-Model XT does not specify sequencing relationships for all process modules, the latter are mapped with work unit containers that having associated workflow instance.
14.2.2. Process Model Enactment

This section describes the concrete activities to be performed when enacting a system development (acquirer / supplier) project (cf. Section 14.2.1). It includes the activities for both acquirer and supplier.

Project start:
- Create project: All artifacts are supplied or created and roles are staffed. The main role of the attached work unit container is the human in charge of the project. He is also assigned to the work units that are contained in it.
- Start work unit container: Project is started.

Project definition:
- Create work unit container / add work unit to container dependency: The number of iterations is planned. Assignments and work unit containers are created for all iterations items. The first iteration is attached to the ‘Execute Iteration’ activity.

Phase initiation:
Requirements are specified, a request for proposal is released and the contract is awarded. With the ‘Schedule Iteration’ activity the requirements to be processed in the upcoming iteration are determined.

Iteration execution:
- Create work unit container / add container dependency: The activities of the iteration are executed. New containers are initiated and added as dependency for activities like ‘Specify System’ (with ‘System Development: System Specifications’ as target) and ‘Design System’ (with ‘System Development: Requirements and Analysis’ as target).
- Create work unit container / add work unit to container dependency: During the ‘Design System’ activity, work unit containers for the requirements are created. For internally addressed requirements, ‘Software Development’ is applied. In turn, for integrating external realizations ‘System Development: System Elements’ is used. The newly created containers are attached to the ‘Realize System’ activity.

Iteration / Project Completion
In the ‘Accept Project’ activity it is determined if another loop (i.e. iteration) has to be conducted.
Figure 14-2: V-Model XT modeling
14.3. Modeling of the Scrum Process

This section discusses the CPM mapping of the Scrum process concepts.

14.3.1. Mapping of the Process Concepts

The structure of the CPM implementation is as simple as the one of the Scrum process itself. Table 14-3 shows the concrete mappings.

<table>
<thead>
<tr>
<th>Scrum Concept</th>
<th>CPM Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project</td>
<td>Project / Work Unit Container / Assignment</td>
</tr>
<tr>
<td>Sprint</td>
<td>Work Unit Container</td>
</tr>
<tr>
<td>Work Product</td>
<td>Project Component</td>
</tr>
<tr>
<td>Backlog Item</td>
<td>Project Component + Assignment / Work Unit Container</td>
</tr>
<tr>
<td>Role</td>
<td>Role</td>
</tr>
</tbody>
</table>

A project is mapped by a work unit container and an associated assignment. By default, the latter is subjected to the human responsible for the project. For each work product, a project component exists, which is taken as input or output of the work units of the mentioned containers. A particularity of the mapping are the backlog items that represent single requirements / functions of the produced software. For each of them not only a project component exists, but also a work unit container as well as an assignment to govern the activities to be executed on the backlog item. The roles of the Scrum process are directly mapped by roles in CPM. Figure 14-3 illustrates the implementation; thereby, process management concepts and the additional concepts are united for the sake of simplicity and readability.

Figure 14-3: Scrum modeling
Figure 14-3 shows the three main workflows of the Scrum process as well as their mutual connections. The project workflow has been extended with an explicit task to create the backlog items for the project. Furthermore, it shows some other exemplary concepts: A backlog item and related product backlog is shown as well as three roles and one milestone.

14.3.2. Process Model Enactment

This section presents the concrete activities for enacting the Scrum process based on the CPM implementation. For Scrum, process enactment is rather different from the other process models discussed in this chapter. Therefore, the CPM process implementation differs as well: The central elements of the Scrum process – the backlog items – are realized by work unit containers, assignments, and related concepts. In the meetings during which these items are planned, distributed or moved, the relating CPM concepts are created, distributed or moved as well.

Project start:
- Create project: All artifacts are supplied or created and roles are staffed. The main role of the attached work unit container is the human in charge of the project (Scrum Master) who is also assigned to the work units that are contained in it.
- Start work unit container: Project is started.

Release planning:
- Create work unit container: Assignments / work unit containers are created for all backlog items.
- Add work unit to work unit dependency: The work units in the new work unit containers for the backlog items are connected to the work units in the project work unit container as defined by the template concepts (prioritizing and estimating).
- Create work unit container: Work unit containers for the sprints are created and linked via the pastExec / futureExec properties. The loop variable in ‘Project’ work unit container is set to number of sprints.
- Add work unit to container dependency: The first sprint is added as target of ‘Sprint Execution’ work unit.
- Add work unit to work unit dependency: All backlog items are added as target of the first sprint. Therefore, the work units in the new work unit containers for the backlog items are connected to the work units in the sprint work unit container as defined by the template concepts (scheduling, item processing, and approving).

Estimate / prioritize backlog items
Both activities are conducted by the team led by the Scrum Master to whom the activities are assigned. This is done for all backlog items leading to the termination of the two activities in the project workflow instance as well.

Sprint execution / sprint planning meeting:
- Distribute assignment: In the sprint planning meeting, for all backlog items to be processed in this sprint, their respective assignments are distributed to the executing humans.
- Move work unit dependency: The remaining backlog items are moved to the next sprint that will take place in the future.

Item processing:
The variable in the items’ workflows is set such that the ‘Estimate Item’ activity can be continuously performed. When activity ‘Process Item’ is completed, the variable is set such that activity ‘Estimate Item’ is no more executed.
Next sprint execution:

- Add work unit to container dependency: When the sprint workflow instance is completed, activity ‘Sprint Execution’ activity terminates and restarts in the next iteration. As it has no dependency (as configured in the template) at that point, there are two possibilities: If the sprint workflow instance, which is still connected to it but has already been terminated, has a connected additional sprint workflow instance (property: futureExec), that instance is taken; if not, a new sprint workflow instance is created.

### 14.4. Extended Process Coverage Scenario

This section presents a comprehensive scenario for the concepts related to extended process coverage and extrinsic workflows presented in Chapter 8. This example resolves the shortcomings presented in Example 4-2 in Chapter 4.

#### 14.4.1. Bug Fixing Use Case

The process coverage scenario from Chapter 4 presented a bug fixing workflow with over 30 activities being conditionally executed on account of various situational properties as, for example, risk or urgency. Thus, the workflow is rather complex. However, due to the contextual dependencies of the activities, only a relatively small subset of the latter will come to execution in a specific situation. In this section, we will show the CPM implementation of this scenario and discuss how it can ease modeling and execution. More precisely, for this scenario, a set of properties has been defined as well as activities and their dependencies on these properties. For a better illustration, we have defined two rather different situations in which the bug fixing workflow will come to execution. In Chapter 8, we have denoted such situations as cases. The first one deals with an urgent fix of a GUI component. The latter is assumed to be part of a simple screen not often used by customers. The second case deals with a database component. The fix is assumed to have an impact on multiple tables in the database. Table 14-4 depicts the product and process properties that were defined for the cases in this scenario as well as concrete values chosen for them by the human during execution.

<table>
<thead>
<tr>
<th>Component</th>
<th>GUI (Case 1)</th>
<th>DB (Case 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Properties</td>
<td>criticality</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>user impact</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>dependencies</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>complexity</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>risk</td>
<td>0</td>
</tr>
<tr>
<td>Process Properties</td>
<td>risk</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>urgency</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>complexity</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>dependencies</td>
<td>0</td>
</tr>
</tbody>
</table>

It is assumed that no other influences exist for the properties. The activities being part of this scenario are shown in Figure 14-4. The latter illustrates different levels of encapsulated building blocks that foster easy modeling, while hiding the inherent complexity of the approach. On the top level, where the ‘Case’ is modeled, there is only a simple sequence consisting of activities and building blocks realizing the workflow patterns.

The scenario demonstrates the flexibility of the approach. Activities can be flexibly integrated: activity ‘Validation to Requirements’ will not always be required. Therefore, it is integrated and connected to the complexity property with a very high value (++) (this connection is not shown in Figure 14-4 for the sake of readability). The testing activities were integrated, mutually excluding each other in the initial workflow. In the declarative specification, they are grouped in a parallel building block and connected to different situational properties. Thus, the situation determines the execution of more than one or none of them. The two types of conditional building blocks are included as well. The review activities are mutually exclusive and it is possible that none of them comes to execution. Opposed to
this, the ‘Integration’ building block requires one of the two mutually exclusive activities to be executed. For the sake of readability, Figure 14-4 shows only a selection of the mutual connections between building blocks and the connections of building blocks to situational properties.

Figure 14-4: Activities of the example bug fixing scenario.

The chosen values lead to the selection of activities for the different workflows as illustrated in Figure 14-5. Due to the low complexity of the GUI case, the bug fix needs no special preparation or design. As the GUI component directly concerns the human, a GUI test and the documentation in the change log has been chosen. The unit test activities have been modeled to be applicable only for cases not being urgent and thus they were omitted. Due to the risk and complexity of the database component and the task related to it, the creation of a separate branch as well as an explicit check for dependencies have been prescribed. In the given case, activity ‘Design Solution’ was nevertheless omitted since it was modeled to be only applicable if ‘Complexity’ is very high (++). Unit as well as regression test activities were included because of low urgency and high criticality, whereas the creation of a regression test was conditionally integrated depending on the presence of regression tests. A code review has been prescribed due to the complexity and criticality of the case. The higher dependencies of the database component also caused the inclusion of an activity to inform another team about the changes. The integration activities are more complex for working with multiple branches as well. A requirement constraint ensures the presence of the ‘Branch Integration’ activities if a separate branch is created.
The workflows shown in Figure 14-5 are simpler than the pre-modeled example mentioned in Chapter 4. The building block library supports quick and problem-oriented modeling as different activity clusters for different purposes and situations are stored within it that can be easily reused. The automated adaption supports workflow diversity, reducing complexity and maintenance compared to all-encompassing models. The scenario illustrates the usefulness of the guidance via the chosen activities by these two considerably different workflows containing tasks matching the situation as well as the processed artifact.

14.4.2. Further Use Cases

This section illustrates other use cases that typically occur in SE projects to show the broader applicability of the approach as well as its reuse and simplicity capabilities. These use cases deal with technology swapping, migration, customer support, and infrastructural issues and are illustrated in Figure 14-6.

‘Migration’ deals with the migration to a new software version of a supporting technology (e.g., a web services framework). ‘Technology Swap’, in turn, deals with the replacement of a technology. Both are similar with the main difference being that ‘Technology Swapping’ is more complex and riskier. Therefore, they can be consolidated into one case. That use case includes a ‘Prepare Transfer’ building block containing activities to, for example, analyze the new technology or technology version. Subsequently, activities ‘Development Cycle’ and ‘Documentation’ are attached. The latter is extended to also include internal documentation, since in case of migrations or technology swaps...
internal documents of the developers may have to be adjusted. Following this, the activities for testing and integration are included.

The case of ‘Customer / 3rd level Support’ deals with situations where developers provide direct support to customers and start with the receipt of a support request. At the top level, it has a simple workflow: the actuator of the support request is to be contacted and the support activity is to be executed. Building block ‘Contact Actuator’, therefore, contains multiple conditional activities for contacting the customer by mail, telephone or directly. The building block for the treatment, in turn, contains conditional activities for direct and deferred treatment. Direct treatment means the immediate fixing of a problem and contains the aforementioned activities for development, testing, and so forth. Deferred treatment, in turn, includes activities for creating a new entry in the bug tracking system. Both of the top level building blocks described here also contain the option not to execute any activity. That way various situations can be handled. For example, if the developer realizes that the problem was only caused by misunderstanding or customer misconduct, he can just contact the customer to sort out the problem and close the case.

The ‘Infrastructural Issue’ use case deals with problems relating to the infrastructure that are reported to the responsible human. For this case, the ‘Customer / 3rd level Support’ case can be almost completely reused since there may be the necessity to contact the actuator of the request to gain additional information or to provide support on it. The second activity, the resolution of the issue, if required, contains slightly modified activities compared to the other cases. Further, there exists the option for deferred treatment involving the creation of a new bug report. Immediate treatment is split into two activities: for simple cases, such as a version change or simple compatibility issue, the issue can be directly resolved. However, in more complex cases, such as instability or licensing changes, further clarification might be required, e.g., with the project manager.

14.5. Automated Quality Management Scenario

This section presents the application of the CPM quality management approach to a comprehensive scenario. Therefore, we utilize the problem scenario introduced in Chapter 4. To have a realistic scenario, we apply a real process and real source code in this case. As input for code analysis, the org.eclipse.osee.framework.database package of the open source Eclipse Open System Engineering Environment was used.

14.5.1. Process

As a SE process, the OpenUP [Ecfo15], a simplified free derivative of the Unified Process [Scot02], was chosen. This process constitutes an iterative process featuring four project phases. In the inception phase, the scope of the project is defined, the use cases are outlined, risks are identified, and candidate architectures are selected. The elaboration phase serves for capturing a majority of system requirements as well as for addressing known risk factors. In addition, the system architecture is established and validated. In the construction phase, the system features are built based on the selected architecture. In the transition phase, the system is deployed to the humans. Each phase contains a number of iterations to complete its goals. Figure 14-7 shows how the OpenUP process can be used in the given scenario.

Since the construction phase is the largest one in the project comprising most development activities, it was the focus of the evaluation. Figure 14-7 shows the other phases in a compressed way. The focus here is on the developers’ activities. Thus, the 'Develop Solution Increment' workflow is shown in detail in Figure 14-7. Overall, 54 extension points (XPs) have been defined for the workflows as depicted in Figure 14-7. The ones relevant for the developers’ activities in a construction iteration are XP8 at the end of the iteration, XP21 for measures to be applied between two assignments, and XP45 as well as XP46 for measures directly related to coding or testing regarding certain artifacts.
14 Practical Application

### 14.5.2. GQM Plan

For the test scenario, we created a GQM plan to enable the AGQM agent processes shown in Table 14-5. Four goals have been chosen: maintainability, reliability, performance, and functionality. This is just a simplified example of what is possible and can be incorporated and tailored by a quality manager.

#### Table 14-5: Example GQM plan

<table>
<thead>
<tr>
<th>GKPI</th>
<th>QKPI</th>
<th>KPI</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>GKPI:REL</td>
<td>QKPI:CK</td>
<td>KPI</td>
<td>MET:WMC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MET:DIT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MET:NOC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MET:CBO</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MET:RFC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MET:LCOM</td>
</tr>
<tr>
<td></td>
<td>QKPI:QMOOD</td>
<td></td>
<td>MET:ANA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MET:CAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MET:CIS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MET:DAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MET:DOC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MET:MOA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MET:MFA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MET:NOM</td>
</tr>
<tr>
<td>QKPI:COMP</td>
<td></td>
<td></td>
<td>MET:CYC</td>
</tr>
<tr>
<td>QKPI:DD</td>
<td></td>
<td></td>
<td>MET:NPA</td>
</tr>
<tr>
<td>QKPI:CC</td>
<td></td>
<td></td>
<td>MET:CC</td>
</tr>
<tr>
<td>QKPI:DLT</td>
<td></td>
<td></td>
<td>MET:DLT</td>
</tr>
<tr>
<td>GKPI:MAINT</td>
<td>QKPI:UND</td>
<td>KPI:CSV</td>
<td>MET:CR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MET:TMM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MET:UEM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MET:UEC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MET:ECB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MET:TMF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MET:CC</td>
</tr>
<tr>
<td>QKPI:CC</td>
<td></td>
<td></td>
<td>MET:DEPOR</td>
</tr>
<tr>
<td>QKPI:CSD</td>
<td></td>
<td></td>
<td>MET:DECOR</td>
</tr>
<tr>
<td>QKPI:MII</td>
<td></td>
<td></td>
<td>MET:JHAWK</td>
</tr>
<tr>
<td>GKPI:FUNC</td>
<td>QKPI:UCC</td>
<td></td>
<td>MET:UCC</td>
</tr>
<tr>
<td>QKPI:FTCF</td>
<td></td>
<td></td>
<td>MET:FTCF</td>
</tr>
<tr>
<td>QKPI:CTAF</td>
<td></td>
<td></td>
<td>MET:CTAF</td>
</tr>
<tr>
<td>QKPI:PRAF</td>
<td></td>
<td></td>
<td>MET:PRAF</td>
</tr>
<tr>
<td>QKPI:PTCF</td>
<td></td>
<td></td>
<td>MET:PTCF</td>
</tr>
</tbody>
</table>

The different metrics and KPIs being part of the plan are shown in Table 14-6. To measure the reliability of the code, different kinds of metrics have been chosen. On one hand, well-known source code metrics like McCabe's cyclomatic complexity [McCabe76] or Nejmeh's npath complexity [Nejmeh98] have been used. On the other, metric suites were integrated, namely Chidamber and Kemerer's metrics suite [ChKem94, Spin05] as well as the QMOOD metrics suite [Badao2, ChCh13]. According to a study conducted in [OEQQ07], these are good predictors for fault proneness and thus for reliability. Another factor that could affect the reliability of source code is whether it is covered by unit tests. This metric can be provided by tools like Cobertura [Cobe15] or EMMA [EMMA15] (see [YaWe06] for a
comparison). Since, via sensors, it is possible to detect the execution of various tools for various activities, other factors can be used as metrics as well. An example for this is the degree of load testing that can also be an indicator of (the lack of) code reliability confidence.

Table 14-6: Utilized metrics

<table>
<thead>
<tr>
<th>GKPI: REL: Reliability</th>
<th>MET: WMC: Weighted Methods per Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>GKPI: MAINT: Maintainability</td>
<td>MET: DIT: Depth of Inheritance Tree</td>
</tr>
<tr>
<td>GKPI: FUNC: Functionality</td>
<td>MET: NOC: Number of Children</td>
</tr>
<tr>
<td>GKPI: PERF: Performance</td>
<td>MET: CBO: Coupling between Objects</td>
</tr>
<tr>
<td>GKPI: CK: ChidamberAndKemerer</td>
<td>MET: RFC: Response for Class</td>
</tr>
<tr>
<td>GKPI: QMOOD: QmoodMetricsSuite</td>
<td>MET: LCOM: Lack of Cohesion in Methods</td>
</tr>
<tr>
<td>GKPI: COMP: Complexity</td>
<td>MET: ANA: AvgNumberOfAncestors</td>
</tr>
<tr>
<td>GKPI: DD: DefectDensity</td>
<td>MET: CAM: CohesionAmongMethods</td>
</tr>
<tr>
<td>GKPI: CC: CodeCoverage</td>
<td>MET: CIS: ClassInterfaceSize</td>
</tr>
<tr>
<td>GKPI: DLT: DegreeOfLoadTesting</td>
<td>MET: DAM: DataAccessMetric</td>
</tr>
<tr>
<td>GKPI: UND: Understandability</td>
<td>MET: MFA: MeasureOfFunctionalAbstraction</td>
</tr>
<tr>
<td>GKPI: MI: MaintainabilityIndex</td>
<td>MET: CYC: CyclomaticComplexity</td>
</tr>
<tr>
<td>GKPI: UCC: UseCaseCoverage</td>
<td>MET: NPC: NPathComplexity</td>
</tr>
<tr>
<td>GKPI: CTAF: CodeTuningActivityFactor</td>
<td>MET: CC: CodeCoverage</td>
</tr>
<tr>
<td>GKPI: PAF: ProfilingActivityFactor</td>
<td>MET: DLT: DegreeOfLoadTesting</td>
</tr>
<tr>
<td>GKPI: PTCF: PerformanceTestComplianceFactor</td>
<td>MET: CR: CommentRatio</td>
</tr>
<tr>
<td>KPI: CSV: CodingStyleViolations</td>
<td>MET: TooManyMethods</td>
</tr>
<tr>
<td></td>
<td>MET: UncommentedEmptyMethod</td>
</tr>
<tr>
<td></td>
<td>MET: UncommentedEmptyConstructor</td>
</tr>
<tr>
<td></td>
<td>MET: EmptyCatchBlock</td>
</tr>
<tr>
<td></td>
<td>MET: TooManyFields</td>
</tr>
<tr>
<td></td>
<td>MET: UCC: UseCaseCoverage</td>
</tr>
<tr>
<td></td>
<td>MET: FTCF: FunctionalTestingComplianceFactor</td>
</tr>
<tr>
<td></td>
<td>MET: CTAF: CodeTuningActivityFactor</td>
</tr>
<tr>
<td></td>
<td>MET: PAF: ProfilingActivityFactor</td>
</tr>
<tr>
<td></td>
<td>MET: PTCF: PerformanceTestComplianceFactor</td>
</tr>
</tbody>
</table>

For maintainability, a set of source code metrics have been selected and grouped to a GQM question concerning the understandability of the code. To enhance the prediction quality of the GQM goal, external approaches have been integrated as KPIs as well: the maintainability index (MI) [OHA91, CALO94, Cole92] is a formula that has proven to be a good predictor of maintainability and can be provided by the tool jhawk [Jhaw15]. Maintainability may be also affected by code smells, i.e., certain problems in the source code. These can be detected via the DECOR approach [MGDM10], which is taken into account as well.

The implementation of all desired functionality is covered by the functionality goal. Two metrics have been chosen to measure it. The use case coverage indicates how much of the desired functionality is implemented. The functional testing compliance factor, in turn, indicates how many of the functional tests were passed. If no functional testing has been performed yet, the value of the functional testing compliance factor will be 0 in the worst case.

The performance goal comprises a metric called the performance testing compliance factor. This metric is similar to the functional testing compliance factor, but deals with performance tests. The other two metrics are related to code tuning and profiling.

14.5.3. Concrete Situation

The scenario is targeted for a construction iteration of the OpenUP process that takes two weeks implying ten workdays. Ten developers are assumed to be part of the team. Each developer has ten 'Develop Solution Increment' assignments that are assumed to take one day each. Each night, reports from code analysis tools are received as part of the nightly build process. As a static analysis tool, PMD [Cope05] is used; Table 14-7 shows the results of a report concerning the selected OSEE module. These results include the threshold for each metric defined via the rules processing component.
Table 14-7: Static analysis results

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
<th>Violation Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>MET:AccClGen</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>MET:AvoidDeeplyNestedIfStmts</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>MET:AvoidInstanceOfChecksInCatchClause</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>MET:AvoidReassigningParameters</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>MET:AvoidSynchronizedAtMethodLevel</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>MET:ClassWithOnlyPrivateConstructorsShouldBeFinal</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>MET:CloseResource</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>MET:CollapsibleIfStatements</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>MET:CompareObjectsWithEquals</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>MET:ConfusingTernary</td>
<td>6</td>
<td>20</td>
</tr>
<tr>
<td>MET:CyclomaticComplexity</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>MET:EmptyCatchBlock</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>MET:EmptyMethodInAbstractClassShouldBeAbstract</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>MET:ExcessiveImports</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>MET:ExcessivePublicCount</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>MET:LooseCoupling</td>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>MET:NPathComplexity</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>MET:OverrideBothEqualsAndHashcode</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>MET:PositionLiteralsFirstInComparisons</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>MET:SimplifyBooleanExpressions</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>MET:SingularField</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>MET:StaticMethods</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>MET:SwitchStmtsShouldHaveDefault</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>MET:TooManyFields</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>MET:TooManyMethods</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>MET:UncommentedEmptyConstructor</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>MET:UncommentedEmptyMethod</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>MET:UnconditionalIfStatement</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>MET:UseCollectionIsEmpty</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>MET:UseLocaleWithCaseConversions</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

For the concrete iteration, the focus is improving the quality of the source code, especially maintainability, since the functionality metrics are not violated, meaning the desired functionality is largely implemented. Therefore, the goal agents have been defined as depicted in Table 14-8.

Table 14-8: Goal agent configuration

<table>
<thead>
<tr>
<th>Agent</th>
<th>Points</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAINT</td>
<td>100</td>
<td>Offensive</td>
</tr>
<tr>
<td>REL</td>
<td>80</td>
<td>Balanced</td>
</tr>
<tr>
<td>PERF</td>
<td>80</td>
<td>Balanced</td>
</tr>
<tr>
<td>FUNC</td>
<td>60</td>
<td>Defensive</td>
</tr>
</tbody>
</table>

For this scenario, the three strategies used for the agents have been defined as shown in Table 14-9. They comprise three values: a start bid indicating how many of the distributed points an agent uses for its first bid and raise / reduce values indicating how the agent raises (reduces) its bid in case of loss (win).

Table 14-9: Agent strategies

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Start bid</th>
<th>Raise</th>
<th>Reduce</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offensive</td>
<td>35%</td>
<td>20%</td>
<td>10%</td>
</tr>
<tr>
<td>Balanced</td>
<td>30%</td>
<td>15%</td>
<td>13%</td>
</tr>
<tr>
<td>Defensive</td>
<td>25%</td>
<td>10%</td>
<td>20%</td>
</tr>
</tbody>
</table>

Each time a q-slot occurs, the AGQM module is triggered to output an ordered list of proposed quality measures. For the current scenario, a 50:50 ratio between proactive and reactive measures was defined. Table 14-10 shows the first ten proposed quality measures generated for a q-slot. Proactive
measures are identified by the prefix “M:P:” and the assigned goal, reactive measures by “M:R:”. The related metric whose threshold was violated for reactive measures is shown as well.

Table 14-10: Proposed quality measures from AGQM

<table>
<thead>
<tr>
<th>Slot</th>
<th>Quality Measure</th>
<th>Related Metric</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>M:P:MAINT:Analyze Reuse Possibilities</td>
<td></td>
<td>m1</td>
</tr>
<tr>
<td>2</td>
<td>M:R::Increase Code Coverage</td>
<td>MET:CC</td>
<td>m2</td>
</tr>
<tr>
<td>3</td>
<td>M:R:Refactor Code</td>
<td>MET:ECB</td>
<td>m3</td>
</tr>
<tr>
<td>4</td>
<td>M:P:MAINT:Review Style Guidelines</td>
<td></td>
<td>m4</td>
</tr>
<tr>
<td>5</td>
<td>M:R:REL:Analyze Error Handling Implementation</td>
<td></td>
<td>m5</td>
</tr>
<tr>
<td>6</td>
<td>M:R:MAINT:Refactor Code</td>
<td>MET:TMM</td>
<td>m6</td>
</tr>
<tr>
<td>7</td>
<td>M:P:PERF:Do Profiling</td>
<td></td>
<td>m7</td>
</tr>
<tr>
<td>8</td>
<td>M:P:MAINT:Analyze Modularity</td>
<td></td>
<td>m8</td>
</tr>
<tr>
<td>9</td>
<td>M:R:PERF:Do Performance Testing</td>
<td>MET:PTCF</td>
<td>m9</td>
</tr>
<tr>
<td>10</td>
<td>M:R:Refactor Code</td>
<td>MET:CYC</td>
<td>m10</td>
</tr>
</tbody>
</table>

To determine the impact of the strategies in conjunction with the distribution of points to the agents in the proactive section, Table 14-11 shows the agents’ bids for the slots, in which proactive measures were proposed. The numbers in parenthesis indicate the bid an agent would have placed according to its strategy if insufficient points were available.

Table 14-11: Agents bids

<table>
<thead>
<tr>
<th>Slot</th>
<th>Winner</th>
<th>FUNC</th>
<th>REL</th>
<th>MAINT</th>
<th>PERF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MAINT</td>
<td>35</td>
<td>24</td>
<td>24</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>MAINT</td>
<td>31</td>
<td>28</td>
<td>28</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>REL</td>
<td>28</td>
<td>32</td>
<td>32</td>
<td>19</td>
</tr>
<tr>
<td>7</td>
<td>PERF</td>
<td>34</td>
<td>28</td>
<td>37</td>
<td>21</td>
</tr>
<tr>
<td>8</td>
<td>MAINT</td>
<td>34(41)</td>
<td>32</td>
<td>32</td>
<td>23</td>
</tr>
</tbody>
</table>

The results correlate with the expected arrangement of the proposed measures, where maintainability measures should be favored most, followed by reliability and performance measures.

For the sake of simplicity, in the current scenario, only early activity completion is assumed without a defined quality overhead factor. Thus, the creation of q-slots only relies on execution time deviations of human assignments. These execution time deviations are shown in Table 14-12. Positive values indicate that an activity took less time than estimated, whereas negative values indicate longer actual execution times. Grey boxes indicate assignments, after which the measure proposal process is started for the respective developer. For this scenario, it was assumed that a quality measure will be possible if at least two hours are available.

Table 14-12: Execution time deviations

<table>
<thead>
<tr>
<th>Developer</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>dev1</td>
<td>1</td>
</tr>
<tr>
<td>dev2</td>
<td>0</td>
</tr>
<tr>
<td>dev3</td>
<td>1</td>
</tr>
<tr>
<td>dev4</td>
<td>0</td>
</tr>
<tr>
<td>dev5</td>
<td>1</td>
</tr>
<tr>
<td>dev6</td>
<td>-4</td>
</tr>
<tr>
<td>dev7</td>
<td>1</td>
</tr>
<tr>
<td>dev8</td>
<td>0</td>
</tr>
<tr>
<td>dev9</td>
<td>0</td>
</tr>
<tr>
<td>dev10</td>
<td>1</td>
</tr>
</tbody>
</table>

With these values, five q-slots are possible in the considered iteration for developers dev1, dev3, dev5, dev9, and dev10. For each q-slot, a measure from the list provided by the AGQM module has been selected, proposed and assessed after application. The chosen measures, the applying developer, and the chosen extension points are shown in Table 14-13. The measure utility has been initialized to ‘1’
for all measures in the scenario. Table 14-13 further shows the relating KPI used for assessment and the newly calculated ‘measure utility’ of the applied measures. The calculations of the proactive measures have not been included here since the GKPIs could not reflect an impact of the proactive measures in the given limited scenario.

Table 14-13: Applied measures

<table>
<thead>
<tr>
<th>Measure</th>
<th>Developer</th>
<th>Extension Point</th>
<th>KPI</th>
<th>Measure Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1</td>
<td>dev1</td>
<td>21</td>
<td>GKPI:MAINT</td>
<td>1</td>
</tr>
<tr>
<td>m2</td>
<td>dev3</td>
<td>21</td>
<td>QKPI:CC</td>
<td>1.17</td>
</tr>
<tr>
<td>m3</td>
<td>dev5</td>
<td>21</td>
<td>KPI:CSV</td>
<td>1.17</td>
</tr>
<tr>
<td>m5</td>
<td>dev9</td>
<td>8</td>
<td>GKPI:REL</td>
<td>1</td>
</tr>
<tr>
<td>m9</td>
<td>dev10</td>
<td>21</td>
<td>QKPI:PTCF</td>
<td>1.29</td>
</tr>
</tbody>
</table>

While the scenario is not detailed and broad enough to ensure the applicability for the majority of SE real-world use cases, it shows the feasibility and potential of the approach towards addressing automated GQM and software quality management.

14.6. Exception Handling Scenario

This section reconsidered the exception handling shortcomings example from Chapter 4 to demonstrate the applicability of the CPM framework. The example incorporated two scenarios, from which we already took the first one as a running example in Chapter 11 to show the application of our concepts. The second scenario deals with a developer creating source code as part of an assignment: Assume that it is prescribed by the process that he creates and executes a unit test for his code (as indicated by workflow A in Figure 14-8). However, as the process is neither enforced nor supported, he can intentionally or unintentionally omit these activities. If such things re-occur often, a growing portion of the code remains untested. In turn, this endangers the reliability of the code base. Therefore, we re-take the ‘Develop Solution Increment’ workflow of the OpenUP process as illustrated by Figure 14-8A.

![Figure 14-8: Exception handling application](image)

After implementing the solution, the developer directly integrates his source code. The steps the system executes to handle that deviation (cf. Chapter 11) are explained in the following.

- Event detection: Based on sensors in the IDE and the source control system, the system detects that the human checks in certain artifacts.
Event aggregation: From the detected events, the system derives that the human must be executing activity ‘Integrate and Build’ at the moment. Since this is not the next intended activity in the workflow instance, an ‘Activity Omitted’ event is created.

Event classification: That event is contextually classified: the omitted activities relate to testing and thus the event is classified as a ‘Testing Activity Omitted’ event.

Handling determination: According to the latter event, an ‘Activity Omitted Exception’ is raised that includes information about the omitted activities and the executing human from the event.

Responsible determination: For this type of exception, the developer who omitted the activities is responsible for the handling as well.

Target determination: In the given case, the workflow instance of the developer is still running; i.e., the respective work unit for the activity ‘Integrate and Build’ is still not finished. For that work unit, an extension point has been defined that can be used for handling extension integration. Thus, a direct integration into that workflow instance is chosen.

Handling preparation: Utilizing the dynamic capabilities of AristaFlow, the exception handling is integrated into the running workflow instance. This is done by the on-the-fly insertion of a new activity during runtime. The latter is connected to a sub-workflow instance containing the handling as illustrated in Figure 14-8B.

14.7. Knowledge Management Support Scenario

This section covers the application of the CPM knowledge management concepts to deal with issues as discussed in the context of Example 4-6. For this purpose we reconsider the ‘Develop Solution Increment’ workflow. First, we show the extension of this workflow with contextual concepts including knowledge management related ones (cf. Figure 14-9). This extension incorporates human-related extensions situated within the context management component: the assignment activities and two exemplary artifacts (i.e., source code and test code). The example presents a sample selection of possible guidance for such a workflow.

The workflow deals with the creation of new software functions and its goal is represented by the assignment ‘Develop Feature X’. The workflow contains five activities, i.e., designing, implementing, integrating, creating, and executing developer tests. Each of these activities is mapped in the context management component and extended by the human-related assignment activities. The latter, in turn, are extended by the atomic tasks representing concrete actions executed with a specific SE tool and manipulating (a) specific artifact(s) (e.g., checking in source code).

The knowledge provider is automatically notified when the execution of the workflow reaches a point for which a GI is defined. This is accomplished via a GI concept in the context management component. The latter indicates what type of GI can be shown at that point and how it shall be shown to the human. A GI can be defined directly for an assignment, assignment activity, atomic task, or indirectly if a GI is defined for an artifact that is processed by one of these activity concepts. Based on this information, the knowledge provider can search for matching GIs. This is realized by the GI tags, using the information from the context management component as tags (e.g., that a GI applies to a certain type of activity or artifact). The knowledge provider can explicitly request additional information from the context management component to find better matching GIs for the current situation (e.g., the role or skill level of the applying human). After having looked up the information, the knowledge provider sends it to the context management component that presents it to the human as configured in the GI concept (e.g., if it is required or optional).
The first of four GIs applied to this workflow is a tutorial associated to the assignment, which contains introductory information on the specific properties of software development in that organization. This could be tagged as applicable to junior engineers that recently joined the organization. The second GI is a link to external information from the Open UP website containing a guideline for developer testing from that process. The third GI is associated to source code artifacts and contains a specific checklist on the coding style as well as related hints for coding of the particular company. This checklist is dynamic, which allows for the automated compilation of checklist items matching the situation in which the checklist is used (e.g. front / backend development). The last GI is associated to the atomic task of checking in source code. In that case, it is a note from the configuration manager containing important information about the current development branch.

We now consider a practical scenario, in which information is added to the knowledge base that can be automatically used in future checklists. In this scenario, a developer notices that in the database code of the developed application, opened cursors were not always properly closed, resulting in erroneously locked resources. After fixing the problem, he puts a note into the Wiki indicating that developers should be cautious with open cursors. Furthermore, he tags it with ‘Backend’ and ‘Development’. Assuming that it has matching a tag defined, the third GI from Figure 14-9 can now use this information as well. That way, the information can be provided automatically to the developers as illustrated in Figure 14-10. Furthermore, based on sensors, the CPM framework can detect to which component the processed artifacts belong, and thus only provide that checklist item for backend development.
14.8. Workflow Coordination Scenario

This section discusses the practical application of the CPM coordination approach. In the context of Example 4-3, three problematic cases regarding coordination in SE projects were discussed. For all of them we presented a solution based on the CPM framework (cf. Chapter 10). We will now briefly recall these solutions and describe their practical integration into the developers’ everyday work using the GUI implementation of the CPM prototype (cf. Figure 14-11).

The first scenario dealt with humans in multi-project environments that report issues with switching projects: Every time this is done, the human must manually gather information about the situation (e.g., the current project, assignment, concrete activities). To support this, we have created the process navigability feature as described in Chapter 10. Utilizing the various concepts related to human activities in the context management component, useful information can be directly displayed to the human. The realization of this is depicted in the lower part of Figure 14-11, which not only displays the current activities, but also the project, assignment, iteration, and other information related to the current tasks and activity as well as the upcoming activities.

The second scenario is related to cooperatively working on an artifact base: in many situations the changes a human makes to certain artifacts can be of interest for others. However, such information is often obscure, which might impose problems as changes could be made twice or certain changes could interfere with each other. To counteract such situations, we developed a notification concept (cf. Chapter 10). That way, there can be various notification rules in place, informing roles or humans about certain events or changes relating to artifact and activities. Concrete examples are shown in the upper part of Figure 14-11, where a human gets informed about a change to a source code package being of interest for him and about the completion of a certain activity of one of his colleagues.

The third scenario dealt with activities executed by a certain human in a project that requires a particular follow-up activity by another human. An example was already shown in Chapter 10: the adaptation of an interface source code component could have an impact on the architectural specification. The latter could also have to be adapted to mirror the current state of the source code. Such relations can be automatically detected using the approach developed in Chapter 10. With this approach, such follow-up activities can be automatically issued and be shown to the respective human directly in the GUI like all other activities (see the lower section in Figure 14-11).
14.9. Sample Application: Software Modernization

To further demonstrate the applicability of the CPM framework, we have applied it to another field within SE: software modernization. Corresponding projects deal with a major issue many companies face; i.e., they have a set of software systems that contain important business knowledge, but cannot be extended or changed easily to conform to new requirements. Such systems are called legacy systems [PaLa06, vdHe09]. We have applied the CPM prototype to such a software modernization setting. This involves, in particular, the implementation of a software modernization process model as well as the support of involved humans through automated knowledge management assistance. We will now briefly depict that. For a more thorough discussion see [GOR14].

The modernization process XIRUP [MOMO08] was applied to demonstrate these abilities. XIRUP defines four main activities for software modernization. All of them are further refined via separate workflows. That way, the modernization process is separated into four main phases: Preliminary Evaluation, Understanding, Building, and Migration. These four phases contain different activities as described in the following.
Preliminary Evaluation: The architecture of the system to be modernized is recovered. Furthermore, a preliminary analysis of the system is conducted, which can influence the modernization decision as well.

Understanding: The architecture recovery is continued to build a model of the system to enable further analysis. Furthermore, a modernization and transformation definition is created to enable the creation of a component model of the system. Based on this model, an in-depth analysis is conducted. Finally, code generation for prototypes takes place.

Building: Code is generated for all components required. To be able to test the latter, model evaluation is continued.
Migration: In this phase, concrete code generation for specific platforms takes place. To test the latter, model evaluation is continued once more.

XIRUP was tailored to include a manual development activity, since we assume that any non-trivial transformation and generation to a modernized system cannot be accomplished automatically without some manual coding involvement. Figure 14-12 shows the XIRUP implementation including the four main activities and their sub-processes.

Having integrated XIRUP into the CPM framework, various features for holistic process and project support can be utilized for executing this process. In the following, a concrete example from the knowledge management area is presented:

Example 14-1 (Knowledge management in software modernization projects):
A junior engineer is provided a GI (cf. Chapter 12) containing appropriate checklist items as shown in Figure 14-13 when the ‘Programming’ activity is detected to be completed via a commit event for a backend component within the Code Generation step of ‘XIRUP Migration’. A senior engineer, in turn, might be bothered by some of the “obvious” checklist items, and some items are not applicable to GUIs or databases.

Figure 14-13: Knowledge provision in software modernization projects

14.10. Lessons Learned from a Preliminary Industrial Application

We have applied the CPM framework in two industrial settings to test its applicability and utility. However, we could not apply a full-scale evaluation for several reasons:

- Manpower: We simply lacked the manpower necessary to develop all aspects of the CPM framework to full industrial maturity.
- Technical issues: Many aspects regarding industrial application and maturity do not relate to the approach itself. Therefore, we kept the focus on the approach itself. Examples of such include networking, computing power, and latency issues. Thus, we had to carefully pick and optimize scenarios to not disturb the team members.
- Company interests: In each of the companies we worked together with a team involved in real projects. We had a scientific prototype and it was not clear whether monetary benefits could result when using this prototype. Thus, we could not test all functions as they did not fit in the current projects or were not mature and stable enough to be used in a productive environment.
Comprehensive scenario: The CPM framework is targeted at the holistic support of SE projects, including many aspects and involving humans. Thus, we believe that it is simply not possible to have a hard (i.e., statistically significant) validation of all aspects. There is a myriad of factors in SE projects we cannot influence. Moreover, many of them depend on the executing humans. Thus, it would be impossible to control all factors and execute projects with and without our approach to directly prove its utility.

With that background in mind, selected functions were tested in the companies. Some of them were optimized to be applicable in a productive environment. We managed to apply our prototype with many of its features. Through this we gained insights into the basic applicability of the framework and its key features. In the following, we sketch the lessons learned from this industrial application.

Environment integration: First, we needed to integrate the CPM framework in a productive environment. This slightly differs from the scientific environment in which we had already tested the approach. Therefore, we had to apply the servers for OSGi and process management and establish connections to the teams’ desktops computers to gather information from them and support them with other types of information. With regard to this, we learned that despite technical obstacles, it is possible to integrate the CPM framework in a productive environment. Moreover, as both companies used a different technology stack, we could show this integration in a Java-dominated environment as well as in one where mostly Microsoft technologies were used.

Workflow guidance: Second, we needed to establish basic workflow guidance for team members. This included workflow enactment on the process management server and enhancing workflow information coming from the latter on our main server with the contextual concepts we developed. Finally, we had to integrate this information as seamlessly as possible into the work environments of the team members. In this context, we learned that this is possible. However, we also learned that, to be really integrated into the developers daily work, it was not feasible to have only a web-based GUI run in the browser. Therefore, we adapted this GUI and integrated it into the developers’ IDEs. Again, this was possible for both the Java and the Microsoft environment by developing plugins for the Eclipse and the Visual Studio IDEs. With this practical application, we learned that, at the operational level, humans reject rigid workflows and, in particular, the way they are usually presented to humans: they can only see the current activity and must provide the system with information to be used for workflow variables such that the system can choose the activity to be executed next. Frequently, humans perceive such questions as unrelated to their current situation. Further, they have the feeling to not have control of the workflow instance they are currently processing. This could lead to a rejection of the workflow and the entire system. In interviews with developers, we could gather information about how they would rather accept such workflows. Out of this information, the ‘abstraction of internal workflow logic’ feature (cf. Chapter 7) emerged. With the latter, humans do not encounter questions marginally related to their current activity. They can rather see their current activity as well as the upcoming ones. Further, they can directly choose which one to process next. This was perceived as transferring the control of the workflow back to the humans, which raised the acceptance of the system significantly.

Processes implementation: After establishing basic integration and workflow guidance we had to understand, model and integrate the processes the companies were using with the CPM framework. Both companies used a slightly different process, i.e., Scrum and V-Model XT respectively. Achieving this created two main insights: First, companies do not use process models exactly as they are specified, i.e., they adapt them to their needs and use the adapted model in their projects. In turn, such adapted processes are often partly obscure. In many interviews with managers and developers, we uncovered these processes and implemented them in the CPM framework. Second, we learned that, despite the complexity of the CPM framework, it is possible to implement such processes within it. The efforts to technically implement the process within CPM are far smaller than those to really gather the process from
the companies. The latter efforts are relatively high. However, they are independent from the CPM framework.

- **Contextual integration:** To be able to not only implement the companies’ process, but to also enact this process dynamically matching the situation at hand, we had to enable contextual integration as well. This requires to establishing sensors for the different tools used by the companies as well as also to gather meaningful information from them. When implementing contextual integration, we learned that sensor integration must be applied carefully as it can, under certain circumstances, slow down the related tools. In some cases, for example, the number of events has to be limited. However, despite these issues, we managed to implement sensors for all relevant tools in the two environments. That way, we learned that it is possible to implement sensors for various tools in both Java and Microsoft environments with reasonable effort. After having integrated the sensors we had to evaluate the quality of the provided events. To achieve this, we calibrated the sensors in the following way: initially, we implemented them in the teams and had a testing period where all developers wrote fine-grained diaries indicating the different tasks they executed and the time when they did it. That way, we tested the sensors and events and adapted them such that they could deliver valid and usable results. We further learned that for many tasks a high number of atomic events are generated. To properly detect such tasks in the right granularity, CEP must be applied in the following way: a higher level event (which marks the execution of a task) should only be created when a certain number of atomic events relating to it have occurred.

- **Quality management:** After having applied all basic features of the CPM framework, we could focus on the main functions supporting the different areas of an SE project. It turned out that for the companies, the most important function was quality management. We could apply the automatic integration of software quality measures into the running human’s workflows (cf. Chapter 9) without major obstacles. In particular, we learned that it is not sufficient to only apply such measures when humans finish their activities or assignments early. This does occur, but not often enough to have an impact with the quality measures. Therefore, we developed the quality overhead feature to enable the reservation of a certain amount of time for quality measures. In particular, this worked within the companies. However, we learned that it is crucial for such a functionality to have it communicated very clearly to the developers as well as to have support from the project manager for it. There must be an awareness of the importance of software quality measures, otherwise no time will be reserved the measures will not be properly executed.

- **Knowledge management:** The second advanced functionality we managed to implement in the companies was the contextual knowledge management approach. For this purpose, we replaced the Wikis they used with our semantic Wiki. In this context, we learned that it can be difficult to motivate humans to enter information as well as to tag this information to be usable by the CPM framework. However, it turned out that when humans start to see this entered information in use as proposed to them by the CPM framework, the motivation to enter and tag it increases.

- **Seamless and cautious integration:** When applying the different features and functions, we learned that it is crucial not to be invasive and not to disturb the normal working schedule of humans. Only if this is guaranteed, a tool will be accepted. First of all, we struggled with such issues, but with some slight modifications, we managed to integrate the CPM framework into the daily work of the teams: A crucial factor was the small and non-invasive GUI we developed and integrated directly into the IDEs. Another factor was the slow and cautious integration: At first, we integrated the GUI without many of the features. That way, it did not yield much benefit, but humans learned that it does not disturb them. Then, we started to integrate additional functions incrementally and the humans utilized them.
14.11. Summary

This chapter has demonstrated the applicability of the CPM framework:

First we have discussed the basic capability of the model to implement an entire process model. In detail, we have modeled and implemented three different process models: Scrum, OpenUP, and V-Model XT. Each of these models has different goals, a different granularity, different properties, and different support features. They all could be implemented to a practical extent and can thus be executed based on the CPM framework.

Second, we have presented practical and comprehensive scenarios for the different features of the CPM framework. These scenarios pick up again the scenarios presenting shortcomings of contemporary projects. The scenarios demonstrate the applicability of the developed concepts for practical situations occurring in real projects. In addition, we have exemplarily applied the CPM framework to another domain within SE; i.e., software modernization projects.

Third, we have implemented a prototype framework realizing the concepts of the CPM framework. We have presented insights and lessons we learned from the real industrial application of the framework. In the latter context, we were able to conduct a preliminary industrial test of the framework as a whole as well as selected CPM features. Thus we could demonstrate the real world applicability of the approach. Finally, we have learned various lessons that lead to practical improvements of the CPM framework.
15. Discussion

This chapter comprehensively discusses our approach and achievements. Furthermore, it compares them to the different categories of other approaches that aim for similar goals. The focus of our approach is neither on solving every possible problem in an SE project nor on providing a highly specific solution covering every possible aspect of a narrow problem area. Instead, we want to provide an extensible framework enabling holistic support for SE projects, and thus to solve a set of important issues discussed at the beginning of this thesis.

First of all, we discuss the achievements of the CPM framework. In particular, we show how the different parts of this thesis contribute to answering the research questions posed in Chapter 2. Then, we discuss the different parts of the approach and compare it to other approaches. As opposed to the discussions in the solution chapters, we compare the approach as a whole to related approaches. For this purpose, Section 15.2 recapitulates the relevant categories of related work as mentioned in the solution chapters. Following this, we discuss the different topics covered throughout this work with regard to these approaches in Section 15.3. Then, Section 15.4 provides a compact comparison of all related approaches involving all these topics. Section 15.5 discusses threats to validity of our research whereas the major findings of the thesis are summed up in Section 15.6.

15.1. Enabling Comprehensive Software Engineering Process Support

SE is a complex discipline with various specific properties. On one hand, the created product – the software – implies special properties: complexity, conformity, changeability, and invisibility [Broo87]. On the other, the associated SE process is difficult to handle and often the source of delayed or failing projects. Many companies struggle to control, manage and support their human-centric SE processes, which impacts the quality of the software as well [BDS+99, Ambi02, Wall07, Dust04, SBBK08]. Having this situation in mind, we have posed three research questions for this thesis in Chapter 2. From these coarse-grained questions, in turn, we have derived a set of more fine-grained requirements in Chapter 4. Further, we have carried out a four-fold evaluation to show that the CPM framework satisfies the requirements: First, the specific solutions we developed in Chapters 8-12 showed the applicability of our basic contextual process approach to various SE areas like quality management or knowledge management. Second, we have implemented a proof-of-concept prototype that shows the technical realizability of the CPM framework. Third, we have implemented four SE process models with rather different properties with this approach. Fourth, we have had a preliminary evaluation in two different industrial settings.

In this section, we will briefly discuss how the different chapters, i.e. different parts of the CPM framework and their evaluation contribute to solving the three research questions posed in the beginning.

Research Question 1: Is it possible to support SE projects by not only documenting but operationally guiding and supporting their processes?

Chapter 7 showed how basic requirements for an approach providing comprehensive SE process support can be satisfied. That way, it is possible to model and enact entire SE process models with the CPM framework. However, many workflows and activities in an SE project are executed extrinsic to
such models based on the CPM framework. Chapter 8, in turn, presented an approach to also integrate such workflows with the SE process models. Chapters 9-12 then introduced various approaches addressing specific SE issues like collaborative work or quality management. That way, the applicability of the basic CPM framework to various SE-related topics was demonstrated. This is backed up by a set of practical scenarios in Chapter 14 as well as the concrete implementation of the CPM framework discussed in Chapter 13. With these points in mind we can state that it is possible to achieve support for SE projects by not only documenting but operationally guiding and supporting their process.

**Research Question 2:** Is it possible to operationalize and guide entire SE process models with (existing) automated tools?

Chapter 6 and 7 showed how to compose a framework out of a set of different tools and technologies. Furthermore, it demonstrated how to build a process support framework capable of covering entire process models. This is evaluated in Chapter 14 by implementing four rather different entire SE process models with this framework. Furthermore, Chapters 9-12 presented more specific approaches based on the implementation of one SE process model. That way we were able to show that the CPM framework not only enables the comprehensive implementation of an entire SE process model, but also fosters a connection of such a model and its enactment to various SE areas like quality or knowledge management. This is also supported by a set of specific scenarios presented in Chapter 14. Altogether we can state that it is possible to operationalize and guide entire SE process models with a framework based on (existing) automated tools.

**Research Question 3:** Is it possible to connect SE process enactment comprehensively to the actual course of the projects including artifacts and humans?

Chapter 7 presented the basic integration of contextual data with process enactment. In Chapters 8-12 this was extended to integrate process enactment with various areas of an SE project like quality or knowledge management. This included a set of approaches to react on the dynamic events occurring in the course of an SE project. In particular, this included automatic contextual adaptations to running workflows as well as automatic detection and handling of various complex exceptions occurring in an SE project and relating to processes, humans, artifacts, or activities. Chapter 14 emphasized that such integration is feasible in practical scenarios as well, e.g., for enabling automatic integration of software quality measures into the SE process or for creation of contextual and process-related checklists for humans on-the-fly. With these points in mind we can state that it is possible to connect SE process enactment to the actual course of the projects including various factors like artifacts or humans.

### 15.2. Related Approaches

Chapter 6 introduced the overall approach applied to build an environment capable of supporting different aspects of SE projects. In SE there is a long tradition of tools and environments with similar goals. There has been an evolution of such tools comprising different phases: Already in the 1980s it was discovered that the complex tasks to be performed in an SE project should be supported somehow: the CASE tools emerged [CNW89, Case85, Sodh91, Wass90]. The latter provided automatic support for different SE tasks and united tools like IDEs or source control systems. The idea of these tools was then improved and expanded with the PCSEEs that were designed to support the whole SE process and to seamlessly integrate SE tools with it. However, after the 1990s, research in this area flattened and other kinds of environments emerged. The latter had the special property that they focused on one specific aspect. Examples include artifact-centered collaboration or knowledge assistance. Besides these three categories of environments, we have discussed other contemporary SE approaches that support different aspects of SE projects. A summary of the discussed approaches is shown in Table 15-1.

Chapter 7 was primarily devoted to the contextual extension of process management concepts as well as contextual process dynamicty and automated process adaptations. With these topics, two areas of 266
related work are of primary interest: process-related approaches focusing on extending and
dynamizing processes (e.g., Provop [HBR08a, HBR10, RHB15], Corepro [MHHR06, MRH08],
Breeze [SMO00], WASA [Wesk00, Wesk01], and VIVACE [ATW+14]), and approaches aiming at
contextual integration of processes (e.g., Context Management [KMK+03], CASS [FaCl04], SOCAM
[GPZ04], and CORTEX [BiCa04]). Regarding the first category, we first discussed plain WfMS and
PAIS approaches without extensions or dynamicity as a basis, and continued with more progressive
ones. An important area are configurable process models enabling the specification of different
variants of a certain model that can be chosen or generated on account of certain facts. Another related
area are approaches allowing for changes to running workflows (e.g., Breeze [SMO00], WASA
[Wesk00, Wesk01], SPADE [BFGL94] ADEPT [DaRe09]). Even more closely related are approaches
that enable such adaptations in an automated fashion.

The second category of approaches deals with contextual integration. An area related to both
categories are artifact-centric processes [KüRe11b]. These extend the process specification by
complex artifact hierarchies and allow the process to react on the context specified by the artifacts.
Other approaches deal with specific aspects of context: creating context models and extending
different notations with these to achieve various goals like process interoperability or compliance.

In Chapter 8 we developed concepts to achieve a greater coverage of the workflows executed in an SE
project. As many of them are far more dynamic than the ones already specified in SE process models,
our approach enabled a declarative activity specification. The latter was connected to contextual facts
to enable an automated configuration of the right workflow for each situation out of the set of
specified candidate activities. Regarding this approach, two areas of related work are interesting:
declarative process models [DHM+96, Mont10, MPA+10, PSSA07, PSA07, Pesi08, SSO01, WBB04,
WPZW10, ZPW12] and process configuration [LDTM11, RSS10, Gott09, RHB15].

Chapter 9 dealt with an important aspect of SE projects: software quality management and its
automated integration with the process. This primarily involved automated code measurements and
their interpretation to automatically distribute software quality measures. Regarding this topic, three
areas of related work are of primary interest: approaches and projects that deal with software metric
application [Dask92, GMK02], software measurement tools [ScJe06, NUS05, LiZh05], and
approaches supporting the GQM technique [FaWu09, HuFa05, STS05]. The latter enables humans to
specify relations between abstract quality goals and concrete measurements of source code artifacts.

In SE various humans must work collaboratively on the source code to achieve project success. Thus,
Chapter 10 developed an approach supporting such collaboration and the coordination of different
humans and teams. In this area, many approaches exist supporting various aspects of human
collaboration [BSV07, JYW07, LeBo07].

Chapter 11 discussed our approach to cope with various types of exceptions that might occur in an SE
project. This includes factors like activities, workflows, humans, or artifacts. Basically, two types of
related approaches exist in this area: first, exception handling approaches that are primarily applied in
WfMS or PAIS [KMK+04]. Second, approaches that apply rules to specify diverse actions to be
performed on account of different facts (like exceptions) [MGR04, CCPP99, DZG10].

Finally, as SE projects are knowledge-intensive [RaTi99, KeHa02], we integrated an approach
supporting knowledge management and dissemination in an SE project in Chapter 12. Again, we
compared this approach to various other related approaches that, in this case, deal with the various
aspects of knowledge management like automated support for storing, managing, and distributing such
knowledge.
Table 15-1: Related approaches

<table>
<thead>
<tr>
<th>Topic</th>
<th>Chapter</th>
<th>Approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Engineering Environment</td>
<td>6</td>
<td>CASE Tools</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PCSEES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modern Environments</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Contemporary SE Approaches</td>
</tr>
<tr>
<td>Contextual / Dynamic Processes</td>
<td>7</td>
<td>WfMS / PAIS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Process Configuration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Artifact-centric processes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Process Adaptations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Automated Process Adaptations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Contextual Process Integration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Business-IT-alignment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enterprise collaboration</td>
</tr>
<tr>
<td>Extended Process Coverage</td>
<td>8</td>
<td>Declarative Processes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Process Configuration</td>
</tr>
<tr>
<td>Automated Quality Management Integration</td>
<td>9</td>
<td>Software Metric Application</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Software Measurement Tools</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GQM Support</td>
</tr>
<tr>
<td>Workflow Coordination</td>
<td>10</td>
<td>Coordination / Collaboration Approaches</td>
</tr>
<tr>
<td>Process Exception Handling</td>
<td>11</td>
<td>Exception Handling Approaches</td>
</tr>
<tr>
<td>Knowledge Management Support</td>
<td>12</td>
<td>SE Knowledge Management Approaches</td>
</tr>
</tbody>
</table>

15.3. Problem Areas

We sketch different areas of problems and requirements presented in this thesis and discuss which approaches satisfy them to which extent. The terms in italics refer to the requirements elicited in Chapter 4 and the concrete solution chapters.

**Basic environment.** A tool or framework that aims to support an entire SE project and its SE process must meet basic requirements to meet the demands of an SE environment. The most basic feature to be in place for automatically assisted process enactment is the capability to govern the process. The latter must not only be specified, but also enacted by the tool meaning that the latter automatically executes the workflows being part of the process. This, in turn, includes the ability to deliver the human tasks specified as part of the workflows to the respective humans. Finally, the tool should guarantee for the correct enactment of the workflows avoiding states like deadlocks.

Besides these capabilities for *basic workflow enactment*, a tool for supporting SE projects must be capable to *support SE processes* in their entirety. This includes special properties like specific task-related information for humans or relations of artifacts to activities. Such a requirement goes far beyond the capabilities of traditional WfMS. However, a tool aiming at comprehensive project support would face a myriad of standard situations. For such situations, it should have a defined behavior, but without creating cumbersome extra work for its humans. Therefore, it should have facilities to easily specify and execute such *configurable automatisms*. Finally, as discussed, SE projects are not easily foreseeable, and the tool must have facilities in place to *support various situations dynamically*.

Besides these capabilities, there are specific requirements for ensuring *adequate process automation* when aiming at the holistic support of SE projects. SE projects involve a high number of interconnected human activities. These activities, their special properties, and their various mutual connections and dependencies must be integrated as well. As discussed, the standard connections between different workflows WfMS provide are insufficient for SE projects. Activities should be allowed to be connected to single activities in other workflows, and there should be the option of arbitrary sub-workflows or –activities. To be able to support projects holistically, a tool should be
capable of (semi-)automating abstract processes like phases as well as low-level operational workflows directly executed by humans. To not distract the latter and burden them with cumbersome extra work, the tools should provide some means of abstracting from the internal workflow logic and provide activities or decisions in a workflow to humans in an easily understandable way.

**Context integration.** In an SE project, various contextual factors influence process enactment, like the involved humans or artifacts. Therefore, a tool that aims at supporting these processes must be aware of this context. This includes facilities to automatically gather and process contextual data. However, to execute and adapt the process according to this data, the processing does not suffice: the process itself must also be prepared for this. Thus, such a tool must provide some means of contextual process integration.

**Process dynamicity.** As discussed, SE projects and processes are dynamic. Therefore, any tool supporting this must enable dynamic processes as well. This comprises the basic ability to change a workflow during run-time. Furthermore, in many SE projects it will often not be feasible to let humans adapt processes manually. Thus, facilities for automated process adaptations should be in place as well. This, in turn, requires a process specification allowing the enacting system to access the various data sets related to the workflows in order to apply adequate adaptations automatically.

**Extrinsic workflows.** SE projects comprise a myriad of situational workflows not covered by SE process models. A tool aiming at holistic SE project support must cover such dynamic extrinsic workflows as well. This not only requires facilities for enacting the latter, but also adequate ways of modeling extrinsic workflows. In particular, modeling must not be cumbersome and extrinsic workflows need to be comprehensible and reusable. This modeling and enactment must both integrate contextual influences to make them usable for various dynamic situations.

**Quality management integration.** Quality management is a crucial part of every software project. A tool that shall support this automatically must provide different capabilities. First, issues existing in the source code must be detected automatically, at least to a certain extent. For respective issues software quality measures should be automatically distributed to be carried out by humans. Thus, awareness of human activities and their state is crucial to know when a measure could be applied without delaying important other activities. Furthermore, a quality measure tailoring should take place before distributing the measures to ensure that they fit to the context of the respective human. Finally, not all distributions might be appropriate or effective. Thus, an assessment must be carried out on the applied measures to improve future distribution actions.

**Collaboration support.** In complex knowledge-worker environments such as SE, projects largely depend on the collaborations of humans involved. However, such collaborations are prone to various issues like forgetfulness of the involved humans. Therefore, a comprehensive SE support tool should be capable of providing support for such collaborations. This comprises facilities for automatic collaboration information distribution to inform a human about the progress of their colleagues. Yet this is only a basic feature. For many cases full automatic collaboration support is desirable and feasible as well: a tool should detect changes and automatically distribute the right follow-up action to the party concerned by such a change.

**Exception handling.** In complex SE projects, various issues and exceptions might occur involving humans, activities and processed artifacts. A tool aiming at comprehensive SE project support should support such situations, and hence be able to detect complex exceptions, involving humans, activities and artifacts, as well as to automatically distribute exception handlings to responsible humans.

**Knowledge management.** SE is a knowledge-intensive discipline and various types of knowledge are a crucial success factor for every SE project. To be able to support knowledge management and distribution in SE projects, a tool must offer different capabilities: First, to be manageable, knowledge must be collected and stored. Second, to be able to effectively distribute it to humans, knowledge must be selected in alignment with their context. Third, it must be automatically provided to the humans matching not only their context but also the state of their workflows and activities.
15.4. Overall Comparison

This section discusses the different categories of related approaches. In this context, we refer to the problem areas introduced in Section 15.3. Further, we summarize the capabilities in Table 15-2.

Table 15-2: Overall comparison

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic tool support for workflows</td>
<td>X</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>●</td>
<td>X</td>
<td>X</td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IC process support</td>
<td>X</td>
<td>○</td>
<td>X</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configurable Automatisms</td>
<td>○</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>○</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic situation support</td>
<td>X</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>○</td>
<td>X</td>
<td>X</td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adequate process automation</td>
<td>X</td>
<td>□</td>
<td>□</td>
<td>X</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context processing</td>
<td>X</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Contextual process integration</td>
<td>X</td>
<td>□</td>
<td>□</td>
<td>X</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>●</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Dynamic processes</td>
<td>X</td>
<td>□</td>
<td>□</td>
<td>X</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>●</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Auto. process adapt.</td>
<td>X</td>
<td>□</td>
<td>□</td>
<td>X</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>●</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Cover dynamic extrinsic workflows</td>
<td>X</td>
<td>□</td>
<td>□</td>
<td>X</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>●</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Modelling of extrinsic workflows</td>
<td>X</td>
<td>□</td>
<td>□</td>
<td>X</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>●</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Integrate contextual influences</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Quality issue detection</td>
<td>○</td>
<td>□</td>
<td>□</td>
<td>X</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>●</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>User activity awareness</td>
<td>X</td>
<td>□</td>
<td>□</td>
<td>X</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>●</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Auto. quality measure distrib.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>●</td>
</tr>
<tr>
<td>Quality measure tailoring</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>●</td>
</tr>
<tr>
<td>Quality measure assessment</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>●</td>
</tr>
<tr>
<td>Collaboration info. distribution</td>
<td>X</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>●</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Active collab. support</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>●</td>
</tr>
<tr>
<td>Complex exception detection</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>●</td>
</tr>
<tr>
<td>Automatic exception handling</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>●</td>
</tr>
<tr>
<td>Collection of knowledge</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>X</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contextual knowledge select</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>●</td>
</tr>
<tr>
<td>Auto. knowledge provision</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>●</td>
</tr>
</tbody>
</table>

The most comprehensive approaches, which focus on the support of entire SE projects, are SE environments. However, none of the categories of SE environments offer such a comprehensive automatic and contextual process support as the CPM framework does. CASE tools offer punctual support for some of the problem areas described, but the big picture is missing. PCSEEs extend this support, primarily by combining the capabilities inherited from the CASE tools alongside a process.
However, process support is still very basic. Both modern environments and contemporary SE approaches strongly focus on a specific area and provide good support there, but disregard other areas.

The discussed process approaches support the process-related problem areas well. Basic WfMS and PAIS provide basic workflow support, but lack dynamic capabilities. These are better integrated within the more sophisticated dynamic process approaches. However, all these approaches fall short in the other problem areas as this is not their focus.

The different approaches dealing with contextual process integration, including FUSION (dealing with enterprise collaboration) [FUSI15] and SUPER (dealing with business-IT-alignment) [SUPE09], primarily aim at extending process specifications. This includes process interoperability [BEK+06, LSH+06], cross-organizational processes [KRFR13, KRL+13], or uniting different views on the processes [PDB+08, ThFe06a, ThFe06b, AFKK07]. They, however, neglect the aspect of process enactment and thus fall short in providing support for this.

Declarative process approaches provide good facilities for specifying and enacting dynamic operational processes where a rigid imperative specification would be not suitable [ReWe12]. They are, however, not suitable for implementing and supporting of an entire SE process model. In particular, they lack sophisticated modeling facilities for the various entities being part of such models.

Approaches dealing with automated software quality management mostly relate to metrics, measurement application, and GQM [Dask92, GMK02, NUS05, FaWu09]. Their strengths are in specifying quality goals or metrics, their relations, and the automated measurement of the source code. However, none of these approaches considers software quality measures and how they could be distributed to the executing humans in an appropriate manner. Furthermore, as these approaches focus on the measurement aspect, they neglect all the other areas described.

Collaboration approaches in SE try to support individual activities and their relations to the activities of other humans involved in the SE project [JYW07, LeBo07, SQTR07]. They support collaboration through information distribution. Automatic activity distribution as a way of automatic support is not prevalent yet, and other project areas are neglected by respective approaches as well.

There exist various approaches dealing with process exception handling. One category of approaches are PAIS having capabilities to directly specify the handleings of exception happening during process enactment [KKL+04]. However, these capabilities are only applicable to exceptions directly related to the process itself. Rule-based approaches are more advanced, as they are able to incorporate various events in rules that, in turn, trigger specific behavior [DZG10, MGR04]. However, they are rather limited in their capabilities related to appropriate distribution of exception handling a human must perform.

Knowledge management approaches in SE aim at supporting knowledge exchange in the projects [BjDi08]. However, they focus on the collection and management of such knowledge. The retrieval of the knowledge remains a challenge. Manual retrieval is often cumbersome for the involved SE engineers, and automatic information distribution is not prevalent.

In contrast to the mentioned approaches, the CPM framework is capable of uniting a vast set of capabilities and areas in order to provide holistic SE project support. As discussed, it does not aim at full SE project automation. However, it provides humans with support for many aspects of such a project.

15.5. Threats to Validity

In a complex setting and environment such as an SE project, validity cannot be guaranteed as it can for narrowly focused research, where simple experiments with big numbers of participants can be conducted. The success of a software project depends on a large number of explicit and implicit
Discussion

Factors. Many of them neither can be measured nor can their influence be estimated. Thus, concrete and easily repeatable experiments at a holistic scale in industrial projects were infeasible. Instead, we have applied selected parts of the approach to software projects in two different settings (i.e., companies). Furthermore, we applied the approach to three scenarios based on rather different process models. Although we have received positive feedback, there remains a set of threats to validity of these results we will discuss in the following:

- **Generalization:** The first threat concerns generalization. We have applied the approach to only two limited scenarios involving small numbers of participants. However, as discussed, SE projects are very complex and, with the limited resources of a small research project, a comprehensive evaluation in real world SE projects is not feasible.

- **Acceptance:** A factor that might influence the results of such an evaluation is the acceptance by humans in the concrete settings. In a research project, in the first place, there is no direct gain or return for the companies; i.e., they have to be convinced to explore such an approach. In our case, the managers were convinced to do so, but their requirement was that the CPM framework will not interfere with project execution. Therefore, we tweaked it to be even less intrusive than it was before. On one hand, we managed to apply it in real productive projects that way. On the other, we gained many important experiences being useful for future applications of the framework.

- **Usability:** Another factor that might influence the applicability of the CPM prototype is the end-user acceptance, which correlates with its usability. As we had no other frameworks covering the same goals we had to experiment with GUIs and their integration into humans’ everyday work. Together with them, we found a way to integrate it smoothly into their environments so they could easily use it without perceiving it as cumbersome.

- **Correctness:** We were unable to prove correctness for every aspect of modeling and automated actions. This is just not possible in such a complex and heterogeneous environment. However, we perceived many practical issues interfering with project execution and the CPM framework and, in most cases, it behaved as expected. We started to optimize the CPM framework to be able to cope with such practical issues.

- **Context:** A potential issue regarding the application of the CPM framework is the provision of sufficient context data to enable the framework to automatically come to the right decisions. In an SE project, there is a myriad of factors influencing project and process enactment. Surely, many of them cannot be measured by a technical solution. However, the approach taken was bottom-up in this case: Start gathering the context data available and provide basic guidance, then apply it in the projects and tweak it to be usable. That way, we could add and test several functionalities step by step.

- **Control:** There is an important issue threatening the validity of our research: Is such a high level of control wanted and reachable? The trend towards agile SE, for example, postulates less control and rigidity. The humans and teams are to decide most things on the fly. However, approaches such as CMMI [CMMI10] clearly show the advantages of process orientation. Furthermore, we have adapted our framework to this situation: It is relatively un-intrusive and provides practical guidance for SE engineers at reasonable time points via its contextual abilities. The experiences we made in the two projects support this claim. Furthermore, we were able to model, integrate, and execute the Scrum process in one of the projects based on the CPM framework.

- **Abstract processes:** There is another point related to the ‘control’ issue: Processes often remain rather abstract. However, on the operational level, their rigid and imperative application seems problematic. Is automated process implementation desirable and feasible anyway? The experiences we made were indeed that it is not, at least not in a rigid and imperative manner. Having that in mind, we developed two approaches for such workflows: the declarative contextual workflow generation and the abstraction from internal workflow logic feature. Both contributed to making operational workflows more dynamic, in giving the humans more control over these workflows and in letting them model and capture the workflows in an easy and comprehensible way. Of course, the applicability of these approaches has to be evaluated in more projects, but first results are already promising.
Discussion

Privacy: One looming issue in applying such a comprehensive approach in an SE project is surveillance and privacy. If humans are not sure about the data that will be recorded and the ways it will be processed, they might simply reject the approach and the results of the application would be worthless. In the two settings we solved the problem in the following way: we applied basic anonymizations and guaranteed that human-specific data will only be kept at the university and not be accessible to the managers of the companies. For further industrial application, however, a more comprehensive and trustable privacy approach will have to be developed so that humans will not reject the approach.

15.6. Major Findings

This section briefly summarizes the major findings of this thesis and the practical settings associated to it.

- It is possible to create an approach that fosters the comprehensive implementation of various SE process models. However, SE projects comprise large sets of activities and workflows enacted extrinsically to SE process models. Such activities and workflows must be covered by any approach that seeks to provide comprehensive SE project support. Furthermore, the implementation of an entire SE process model requires incorporating various factors beyond workflows and activities to be applicable and usable.
- The context and situation of the different humans working in an SE project is a crucial factor for their work. Thus, it constitutes one of the most important factors for successful SE projects. It is, however, possible to integrate contextual data into SE process enactment to achieve effective and efficient SE project support.
- It is possible to integrate various technologies and tools into a framework enabling comprehensive SE process and SE project support. It is further possible to apply the related technologies in a way they are applicable and usable in real SE projects.
- As the human is the central component of any SE project, many aspects largely depend on him. Software engineers are knowledge workers executing a large set of complex activities. Due to this it is clear that any automated approach for contextual SE project support can only take a supportive role and automated support must not be too rigid or prescriptive. Furthermore, SE projects comprise a high number of variables and possible expected and unexpected events that might occur during their execution. Therefore, any supportive approach must be flexible and allow human interference with the planned process.
- SE processes are rarely enacted exactly as planned. Dynamic changes and flexible processes are a crucial factor for automated SE process enactment.
- It is possible to automatically connect various areas of an SE project like quality or knowledge management to SE process enactment.
- Quality management is a crucial part of any SE project. However, it is one of the most important factors that lead to delays or deviations regarding the SE process. Comprehensive SE process support must be aware of this fact and intervene both quality and process management.
- Coordination and collaboration is a crucial and often underestimated part of SE projects. In this area omissive errors can introduce severe problems to an SE project if its participants are not properly supported.
- In any SE process, complex exceptions may occur affecting the process, but stemming from a combination of various sources like humans, artifacts, tools, or the process itself. If such exceptions shall be handled, all these factors must be taken into account.
- Knowledge management is a crucial area of any SE project. However, in many projects it is not properly applied and knowledge dissemination is impeded. SE knowledge support should comprise contextual factors as well as the SE process to be effective.
15.7. Summary

In this chapter, we have comprehensively discussed the CPM framework. First, we have recapitulated the three research questions and discussed how the CPM framework enables SE project and process support. We have further compared the CPM framework in its entirety with other approaches. To achieve this, we have briefly recapitulated all other approaches having at least partially similar goals. Then, we discussed the different problem and requirement areas our approach addresses. Having these two sets of approaches and problem areas, we have confronted both comprehensively. This comparison revealed that there are related approaches that address at least partly similar requirements as the CPM framework. For specific problem areas, these approaches provide solutions. However, as major advantage, our framework unites all these different areas of an SE project. It not only supports SE process models, but integrates support for SE quality management, knowledge worker collaboration, or knowledge management as well. In addition, it provides sophisticated facilities for automation, context processing, and process dynamicity by seamlessly integrating state-of-the-art technologies. This enables a new kind of comprehensive SE human guidance that no other framework provides to the best of our knowledge.

Furthermore, we have discussed threats to validity and major findings regarding the described research. Summing up this discussion, we can state that: The goal of our research is comprehensively supporting and guiding an entire SE project. This necessarily incorporates a set of different problem areas and technologies. Furthermore, SE projects largely depend on the participating humans as well as a myriad of other factors that cannot be easily measured. Consequently, one cannot provide a formal proof of the benefits provided by our research. We further did not have the resources to apply large scale industrial applications with dozens of companies and hundreds of participating humans. However, we have implemented the entire approach as an SE framework. Furthermore, we have applied the basic approach to various areas of SE projects like quality or knowledge management, and provided practical scenarios demonstrating its benefits. Finally, we have applied the CPM framework in two industrial settings, where it was used in real productive projects. The results of this application were promising and show the potential of the CPM framework.
Part IV

Conclusion
16. Summary and Outlook

SE projects are complex, long-running, and knowledge-intensive, depending on a myriad of different factors that are not easily controllable. Furthermore, the developed product, i.e., the software, constitutes an intangible asset whose quality state cannot be measured easily. This places pressure on the knowledge workers in SE projects. Many aspects of the projects, their process, and the produced product are implicitly managed and prone to forgetfulness or other errors.

Due to these various issues, SE projects have always been problematic. From the beginning of SE until today many projects have exceeded their budgets and schedules, delivered low-quality erroneous software, or even failed completely. To make projects more repeatable as well as to support their execution, SE process models have been developed. This started in the 1970s with classical models such as the Waterfall Model [Royc70] or the Spiral Model [Boeh88]. However, these SE models often were too rigid and could not mirror the dynamic SE project execution in reality. More recently, the agile trend took account of this property and agile processes like Scrum [ScBe01] have been popularized. These developments have improved the situation, but still many projects struggle with time, resources and software quality.

A remaining problem concerns the operational support for the projects, their processes, and, first and foremost, the involved software engineers. Projects and their processes are often planned up-front and their execution does not match this plan, resulting in an ever-growing gap between plan and reality. Software engineers utilize a large set of SE tools supporting different tasks like IDEs, source control management systems, or bug trackers. However, the complexity of SE projects keeps growing; e.g., the sizes of projects keep growing, the different tools are often rather complicated, and their number grows as well. Moreover, holistic SE support is missing. Tools may comprehensively support a specific task, but are not connected well to the other tools. Much is still left to the software engineers without providing any guidance to them. Although their collaboration is crucial, it has not been properly supported or governed yet. Crucial project knowledge remains only in the heads of the humans, and is not properly stored, managed and disseminated among the project participants.

This work presents a holistic approach to support both SE projects and, especially, SE processes. In the following we will briefly summarize the core contributions of this thesis:

**Automated process support:** The CPM framework provides an infrastructure for comprehensive SE project and SE process support. It unites different state-of-the-art technologies encapsulated in loosely-coupled components. The set of components comprises, among others, dedicated components for process enactment, context integration, or knowledge management. Thus, it not only enables the modeling and enactment of workflows, but also the extension of the workflows with a myriad of additional data sets that support the implementation of entire SE process models. As SE process enactment is known to be complex and dynamic, the CPM framework comprises additional basic components, enabling the simple definition and execution of configurable automatisms to support humans in recurring standard situations. It further enables CPM to cope with various dynamic situations whose exact configuration and course might not be estimated a priori.

**Context integration:** The execution of an SE project and SE process depend on various contextual factors, like the properties of the executing humans or the states of involved artifacts. The CPM framework integrates different facilities to deal with such information. First, it features a set of sensors that can be integrated into various SE tools to automatically gather information. Second it enables the automatic processing of such information to derive meaningful information from the numerous events happening in an SE project. Third, by an extended process specification, it enables the direct and tight integration of process enactment with the context of the project.
Process dynamicity: SE project execution is rather dynamic and mostly differs to what was planned. Therefore, the CPM framework incorporates dynamic processes. Thus, the different workflows executed in the context of an SE process can be dynamically changed during run-time to adhere to changing situations. However, as the projects comprise many different areas and the set of influential context factors is high, it can be challenging for a human to apply a process adaptation on account of this data. To support this, the CPM framework not only enables manual process adaptations, but also automated and context-aware ones.

Extrinsic process coverage: Process models cover a substantial portion of the work done in an SE project. However, many workflows cannot be covered by them for various reasons. Such workflows are characterized by three main properties. First, they cannot be completely foreseen. Second, they are rather dynamic. Third, they depend on their context even more than the ones belonging to the SE process models. Therefore, the CPM framework incorporates a declarative and dynamic way of modeling such workflows that allows directly integrating contextual influences. Furthermore, it enables a uniform way of enacting them similarly to imperative workflows.

Quality assurance integration: Quality assurance is a crucial part of any SE project. However, many projects struggle with bad source code quality. Therefore, the CPM framework integrates facilities to automatically measure the source code quality and to distribute software quality measures to the software engineers in case of quality problems. This comprises a monitoring of the human activities to be able to find the right point in the process for inserting a software quality measure as well as a dynamic tailoring of the latter to select the right measure for the right human and situation. Finally, the CPM framework automatically assesses the applied measures to optimize the measure distribution over time.

Collaboration and coordination: The collaboration of the involved knowledge workers constitutes a crucial part of any SE project. This collaboration might get complicated and error-prone in large projects. Therefore, the CPM framework integrates facilities to support such collaboration in two ways. First, it fosters automated information distribution, informing one human about important changes to their environment as, for example, the status of the activities of their colleagues. Second, it is capable of automatically initiating follow-up activities for certain changes in a project impacting other humans.

Process exception handling: In an SE project many things do not work exactly as planned. Many exceptions might occur relating to the process, its activities, the involved humans, or the processed artifacts. The CPM framework uses its contextual infrastructure to detect as many of these complex exceptions as possible. Furthermore, it is capable of automatically determining an exception handling procedure and distributing it to the appropriate human to apply it.

Knowledge provisioning: SE projects largely depend on the knowledge of the humans involved possess. However, the management and distribution of such knowledge remains a challenge. The CPM framework fosters gathering, storing, and managing of such knowledge. Furthermore, due to its process- and context-related capabilities, it is capable of automatically distributing knowledge to project participants that matches their current situation and problems.

The CPM framework delivers a set of functions we believe to be unique. It unites various areas like dynamic process management, human assistance, and quality management. In these areas, however, there exist specific approaches as well. For example, [HMMR14] focuses on supporting the human by providing process visualizations and additional information. Others support enactment of parts of processes for specific humans based on process views [KoRe13]. The knowledge worker is supported by various approaches as well. Examples include flexible checklist support [MuRe14] or mobility support of knowledge workers by approaches like [PMR14]. Another area is quality management for processes with approaches like [LoRe15]. All of these approaches have their strengths and go beyond the capabilities of the CPM framework in a specific area. The main strength of the latter is, however,
is the comprehensive, applicable and usable integration of a large set of different areas to better support humans in SE projects.

The CPM framework presented in this work solves many problems of SE projects as it provides holistic SE project and SE process support. However, there exist various options for further improving and extending the approach. In the following we will highlight some of the most important ways, the CPM framework might be extended.

We have already discussed and created a set of extensions and additions to the CPM framework not directly being part of this work. One of these extensions is related to the modeling of the contextually extended processes. In the CPM framework, humans model the workflows directly in the WfMS and the different extensions in a web GUI. Such modeling would be simplified if humans had modeled the complete process in one tool and notation. To enable this, we have already created a preliminary approach for a SE workflow language comprising all necessary properties and can then be automatically transformed into the workflows of a WfMS and the additional contextual extensions applied in the CPM framework. For further reading on this topic, we refer to [GOR11a].

In SE, the assessment of processes and their improvement is a crucial topic as well. Process assessment and improvement approaches like ISO 15504 (SPICE) and CMMI (for more information regarding these two, see [Wall07]) have therefore received much attention. Thus, an integration of such approaches into an approach for SE project and SE process support is desirable. We have already created such an extension of the CPM framework enabling the semi-automatic assessment of an executed SE process with models like CMMI or SPICE (see [GOR12e, GOR13]).

Another interesting option will be the application of the CPM framework in other domains as standard SE projects. In the future, with a set of specific other sensors, an application in other knowledge-intensive domains becomes possible. We have already started to investigate such options. As we did not have the resources to develop a completely new set of sensors, we investigated a type of project that can rely on the sensors we already have. In [GOR14] we have discussed the application of the CPM framework in the context of specific software modernization projects.

The preliminary evaluation showed that our concepts have great potential for really supporting SE projects. Regarding the industrial application, however, a larger scale industrial evaluation remains as a future task. To enable the latter, we will need to add various features. For being usable in large scale productive projects, a consistent privacy approach will be necessary. Furthermore, some of the applied technologies will have to be adapted to ensure performance and scalability in larger industrial environments.
Bibliography


Bibliography


289


Bibliography


# Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALM</td>
<td>Application Lifecycle Management</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BPMN</td>
<td>Business Process Modeling Notation</td>
</tr>
<tr>
<td>CASE</td>
<td>Computer-Aided Software Engineering</td>
</tr>
<tr>
<td>CEP</td>
<td>Complex Event Processing</td>
</tr>
<tr>
<td>CPM</td>
<td>Context-aware Process Management</td>
</tr>
<tr>
<td>ECA</td>
<td>Event Condition Action</td>
</tr>
<tr>
<td>GI</td>
<td>Guidance Item</td>
</tr>
<tr>
<td>GKPI</td>
<td>Goal Key Performance Indicator</td>
</tr>
<tr>
<td>GQM</td>
<td>Goal Question Metric</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IS</td>
<td>Information System</td>
</tr>
<tr>
<td>KPI</td>
<td>Key Performance Indicator</td>
</tr>
<tr>
<td>MDA</td>
<td>Model-Driven Architecture</td>
</tr>
<tr>
<td>MDE</td>
<td>Modell-Driven Engineering</td>
</tr>
<tr>
<td>MVC</td>
<td>Model-View-Controller</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>PAIS</td>
<td>Process-Aware Information System</td>
</tr>
<tr>
<td>PCSEE</td>
<td>Process-Centered Software Engineering Environment</td>
</tr>
<tr>
<td>SEE</td>
<td>Software Engineering Environment</td>
</tr>
<tr>
<td>QKPI</td>
<td>Question Key Performance Indicator</td>
</tr>
<tr>
<td>SME</td>
<td>Small and Medium-sized Enterprise</td>
</tr>
<tr>
<td>SPEM</td>
<td>Software &amp; Systems Process Engineering Metamodel specification</td>
</tr>
<tr>
<td>SOA</td>
<td>Software Quality Assurance</td>
</tr>
<tr>
<td>WfMS</td>
<td>Workflow Management System</td>
</tr>
</tbody>
</table>
Part V

Appendices
A. Ontology

The ontology has been modeled and accessed only using the Protégé ontology editor. In the following, we will be present a small set of exemplary concepts of the ontology as modeled in OWL XML. Some of the properties have been renamed due to the fact that names in an OWL ontology are unique and one such property as e.g. ‘workUnitContTempl’ should not be used to connect a work unit container template to a project template and to a work unit template but rather two distinct properties are required here, in this case, ‘workUnitContTempl’, and ‘containingWUCT’.

A.1. Imperative Process Concepts

This section presents the discussed concepts for the imperative processes.

A.1.1. Template Concepts

This section presents the template concepts. First, Figure A-1 gives an overview of them (barely readable and just as visual overview) directly from the Protégé ontology editor. After that, for an exemplary concept, the XML definition from the ontology is shown (cf. Listing A-1). Keep in mind that the properties of the concepts are not directly part of the concepts. However, the restrictions on a selection of properties can be seen.
Figure A-1: Imperative template concepts in ontology
Listing A-1 (Work unit template)

```xml
<owl:Class rdf:ID="WorkUnitTemplate">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
        1
      </owl:maxCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="workflowUserInfo"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="assignActTempl"/>
      </owl:onProperty>
      <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
        1
      </owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
        1
      </owl:cardinality>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:ID="repeatable"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
        1
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">
        1
      </owl:maxCardinality>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:ID="omittable"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Class rdf:about="TemplateConcepts"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">
        1
      </owl:maxCardinality>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:ID="extensionPointTemplSet"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:ID="actTempl"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">
        1
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
        1
      </owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="containingWUCT"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```
A.1.2. Individual Concepts

This section presents an excerpt of the individual concepts discussed (cf. Listing A-2), preceded by a (barely readable) visual overview of the concepts and their connections in Figure A-2.
Figure A-2: Imperative individual concepts in ontology
Listing A-2 (Work unit)

```xml
<owl:Class rdf:about="WorkUnit">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int" >1</owl:maxCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="assignAct"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger" >1</owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="basisWUT"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int" >1</owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="WUfutureExec"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int" >1</owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="WUprimRole"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int" >1</owl:cardinality>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:ID="singleExec"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger" >1</owl:cardinality>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:ID="actInst"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int" >1</owl:cardinality>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:ID="finalized"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int" >1</owl:cardinality>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:ID="WUpriRole"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```
A.2. Declarative Process Concepts

This section presents an excerpt of the concepts related to the declarative modeling approach from Chapter 8 (cf. Listing A-3 and Listing A-4) preceded again by a visual overview in Figure A-3. The building block template shows only one restriction because the cardinality restrictions on properties ‘info’ and ‘problem’ are realized in its super concept, the declarative modeling element. The declarative container template, in turn, is a sub concept of the declarative modeling element and the work unit container template.
Figure A-3: Declarative concepts in ontology
Listing A-3 (Building block template)

```xml
<owl:Class rdf:about="BuildingBlockTemplate">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="containedIn"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="DeclarativeModelingElement"/>
</owl:Class>
```

Listing A-4 (Declarative container template)

```xml
<owl:Class rdf:about="#DeclarativeContainerTemplate">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class>
          <owl:intersectionOf rdf:parseType="Collection">
            <owl:Class rdf:about="#DeclarativeModelingElement"/>
            <owl:Class rdf:about="#WorkUnitContainerTemplate"/>
          </owl:intersectionOf>
        </owl:Class>
        <owl:Restriction>
          <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:minCardinality>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#containedBBset"/>
          </owl:onProperty>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf rdf:resource="WorkUnitContainerTemplate"/>
  <rdfs:subClassOf rdf:resource="DeclarativeModelingElement"/>
</owl:Class>
```
B. Conceptual Framework

In this appendix, we discuss the concepts of the CPM framework and provide exemplary formal definitions. The appendix is separated into different sections to improve readability. First, different concepts for entities are shown followed by concepts for consistency checks and algorithms.

B.1. Entity Concepts

In this section, concepts for various CPM entities are discussed. For the sake of brevity we only show a selection of interesting concepts.

B.1.1. Basic Concepts

This section deals with the basic concepts of the CPM framework. We only show a selection of the definitions of these concepts. However, for completeness, Table B-1 gives an overview about all of these basic concepts. Extensions for topics like quality management are not included here.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifiers</td>
<td>All valid identifiers over a given alphabet. All concepts have a name ε Identifiers</td>
<td>Types</td>
<td>All definable object types. All concepts have a distinct type ε Types</td>
</tr>
<tr>
<td>TemplateConcepts</td>
<td>All template concepts in the framework used for defining workflow structures.</td>
<td>IndividualConcepts</td>
<td>All individual concepts in the framework used for individual enactments of processes defined by the template concepts.</td>
</tr>
<tr>
<td>WFTemplates</td>
<td>All workflow templates within a WfMS.</td>
<td>WFinstances</td>
<td>All workflow instances within a WfMS.</td>
</tr>
<tr>
<td>ActivityTemplates</td>
<td>All activities within workflow templates in a WfMS.</td>
<td>ActivityInstances</td>
<td>All activities within workflow instances in a WfMS.</td>
</tr>
<tr>
<td>AreaTemps</td>
<td>All area templates. A set of area templates can be used to define abstract categories (or disciplines) for projects like, e.g., ‘Implementation’ or ‘Testing’.</td>
<td>Areas</td>
<td>All definable areas used to categorize activities and artifacts in concrete projects as applied in many processes (e.g., the disciplines of the OpenUP process).</td>
</tr>
<tr>
<td>ProjectTemps</td>
<td>All project definitions within the framework, which have (among other properties) a defined type and a defined process (that is defined by work unit container template). The process depends on the type of project. Project templates also have defined area templates.</td>
<td>Projects</td>
<td>All concrete projects in the framework.</td>
</tr>
<tr>
<td>WorkUnitContTemps</td>
<td>All definable work unit container templates.</td>
<td>WorkUnitConts</td>
<td>All definable work unit containers.</td>
</tr>
<tr>
<td>WorkUnitTemps</td>
<td>All definable work unit templates.</td>
<td>WorkUnits</td>
<td>All definable work units.</td>
</tr>
<tr>
<td>WorkUnitContTemplDeps</td>
<td>All definable work unit container template dependencies.</td>
<td>WorkUnitContDep</td>
<td>All definable work unit container dependencies.</td>
</tr>
<tr>
<td>WorkUnitTemplDeps</td>
<td>All definable work unit template dependencies.</td>
<td>WorkUnitDeps</td>
<td>All definable work unit dependencies.</td>
</tr>
<tr>
<td>Concept</td>
<td>Description</td>
<td>Concept</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Milestonetemplates</strong></td>
<td>All definable milestone templates, which can be used to define abstract milestones of a process and are attached to a certain work unit template.</td>
<td><strong>Milestones</strong></td>
<td>All definable work milestones used to model the milestones of a concrete project and store information about their achievement.</td>
</tr>
<tr>
<td><strong>Assigntemplates</strong></td>
<td>All definable assignment templates.</td>
<td><strong>Assigns</strong></td>
<td>All definable assignments.</td>
</tr>
<tr>
<td><strong>Assignacttemplates</strong></td>
<td>All definable assignment activity templates.</td>
<td><strong>Assignacts</strong></td>
<td>All definable assignment activities.</td>
</tr>
<tr>
<td><strong>Atomiictasktemplates</strong></td>
<td>All definable atomic task templates.</td>
<td><strong>Atomictasks</strong></td>
<td>All definable atomic tasks.</td>
</tr>
<tr>
<td><strong>Tooltemplates</strong></td>
<td>All definable tool templates, which can be used to define tools types as e.g., IDE within the framework.</td>
<td><strong>Tools</strong></td>
<td>All definable tools used to capture concrete tools used in concrete projects.</td>
</tr>
<tr>
<td><strong>Projcomptemplates</strong></td>
<td>All definable project component templates, which are used to model a hierarchy of artifacts within the framework.</td>
<td><strong>Projcomps</strong></td>
<td>All definable project components capturing the structure of concrete artifact instances used in projects.</td>
</tr>
<tr>
<td><strong>Artifacttemplates</strong></td>
<td>All definable artifact templates, which are sub-concepts to the project component templates for defining the artifacts within the hierarchy.</td>
<td><strong>Artifacts</strong></td>
<td>All definable artifacts, which are sub-concepts to the project components for capturing the artifacts within the hierarchy.</td>
</tr>
<tr>
<td><strong>Sectiontemplates</strong></td>
<td>All definable section templates, which are sub-concepts to the project component templates for defining the structure of the hierarchy.</td>
<td><strong>Sections</strong></td>
<td>All definable sections, which are sub-concepts to the project components for structuring the hierarchy.</td>
</tr>
<tr>
<td><strong>Roolertemplates</strong></td>
<td>All definable role templates, which can be used to define roles as e.g., 'Quality Manager' within the framework.</td>
<td><strong>Roles</strong></td>
<td>All definable roles used to concretely connect humans with their tasks, responsibilities, or artifacts.</td>
</tr>
<tr>
<td><strong>Eventtemplates</strong></td>
<td>All definable event templates, which can be used to pre-define certain events within the framework, including a relation to the tool that triggered them, as e.g., the check-in of a certain source code artifact with a source control framework.</td>
<td><strong>Events</strong></td>
<td>All definable events used to capture concrete events and their data occurring in projects.</td>
</tr>
<tr>
<td><strong>Problemtemplates</strong></td>
<td>All definable problem templates that can be used to pre-define certain problems that might occur relating to certain events, e.g., the fact that the complexity of a source code artifact becomes too high due to code changes by different humans.</td>
<td><strong>Problems</strong></td>
<td>All definable problems capturing concrete problems and their data occurring in projects.</td>
</tr>
<tr>
<td><strong>Extensionpointtemplates</strong></td>
<td>All definable extension point templates.</td>
<td><strong>Extensionpoints</strong></td>
<td>All definable extension points.</td>
</tr>
<tr>
<td><strong>Extensiontemplates</strong></td>
<td>All definable extension templates.</td>
<td><strong>Extensions</strong></td>
<td>All definable extensions.</td>
</tr>
<tr>
<td><strong>Workuserinfos</strong></td>
<td>All definable workflow user information.</td>
<td><strong>Workflowvars</strong></td>
<td>All definable workflow variables.</td>
</tr>
<tr>
<td><strong>Decalternatives</strong></td>
<td>All definable decision alternatives.</td>
<td><strong>Resources</strong></td>
<td>All human resources within the framework comprising humans and teams of humans. Teams consist of one or more humans and have a leader which is also a human. All humans within the framework.</td>
</tr>
<tr>
<td><strong>Varvalues</strong></td>
<td>All definable workflow variable values.</td>
<td><strong>Persons</strong></td>
<td>All human resources within the framework comprising humans and teams of humans. Teams consist of one or more humans and have a leader which is also a human. All humans within the framework.</td>
</tr>
<tr>
<td><strong>Vartemplates</strong></td>
<td>All definable workflow variable templates.</td>
<td><strong>Teams</strong></td>
<td>All human teams within the framework.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Skilllevels</strong></td>
<td>All definable skill levels humans can possess.</td>
</tr>
</tbody>
</table>

Two properties shared by all concepts are type and name. The former denotes the type of concept, like work unit container, whereas the latter is a unique identifier for each concept. As both are common for all concepts, they are not further mentioned in the definitions.
Work Unit Dependency

As discussed in Chapter 7, we have added a new dependency between different workflows to the one already existent in WfMS. The template concept for this new dependency is defined in Definition B.1:

**Definition B.1 (Work Unit Template Dependency)**

A work unit template dependency is a tuple \( \text{workUnitTemplDep} = (\text{type}, \text{name}, \text{source}, \text{target}, \text{async}, \text{behavior}) \) where

- \( \text{source} \in \text{WorkUnitTempls} \) is the source depending on target.
- \( \text{target} \in \text{WorkUnitTempls} \) is the target the source depends on.
- \( \text{async} \in \text{BOOLEAN} \) indicates whether the dependency is asynchronous or synchronous.
- \( \text{behavior} \in \{\text{firstShot}, \text{lastShot}\} \) indicates when the dependency is satisfied.

**WorkUnitTemplDeps** describes the set of all definable work unit template dependencies.

The work unit template dependency connects a work unit template (source) with a work unit template (target). When a work unit, which is based on the source template, comes to enactment, the work unit container containing the source work unit will be created (if it is not already in place). If the async property is set to FALSE, the termination of the source will depend on the termination of the target. As the target is a work unit in that case, it might be executed more than once in a LOOP. Therefore, the behavior property governs whether the source terminates with the first or last termination of the target.

Both dependencies have a related stateful individual concept capturing one individual enactment of the workflows defined by the templates. The work unit dependency (cf. Definition B.2) allows for storing the information whether its target has been executed using the property finalized. As the target is a work unit it might have multiple iterations. Therefore, this dependency has two properties, one indicating that the target has been executed (executed), the other indicating that the final execution of the target has happened (finalized).

**Definition B.2 (Work Unit Dependency)**

A work unit dependency is a tuple \( \text{workUnitDep} = (\text{type}, \text{name}, \text{source}, \text{target}, \text{async}, \text{behavior}, \text{executed}, \text{finalized}, \text{basis}) \) where

- \( \text{source} \in \text{WorkUnits} \) is the source depending on target.
- \( \text{target} \in \text{WorkUnits} \) is the target of the dependency the source depends on.
- \( \text{async} \in \text{BOOLEAN} \) indicates whether the dependency is asynchronous or synchronous.
- \( \text{behavior} \in \{\text{firstShot}, \text{lastShot}\} \) indicates when the dependency is satisfied.
- \( \text{executed} \in \text{BOOLEAN} \) indicates if the target has been executed at least once.
- \( \text{finalized} \in \text{BOOLEAN} \) indicates if the target has been executed for the last time.
- \( \text{basis} \in \text{WorkUnitTemplDeps} \) is the template that workUnitDep is based on.

**WorkUnitDeps** describes the set of all definable work unit dependencies.

Human Activity Management

The human activity concepts (assignment, assignment activity, atomic task) require a particular set of runtime properties; therefore, we have added individual concepts for them. In the following we show Definition B.3 for the assignment.

**Definition B.3 (Assignment)**

An assignment is a tuple \( \text{assign} = (\text{type}, \text{name}, \text{responsible}, \text{assignActSet}, \text{workUnitCont}, \text{basis}, \text{state}, \text{guidanceSet}, \text{plannedStart}, \text{plannedEnd}, \text{actualStart}, \text{actualEnd}, \text{area}, \text{contentInfo}) \) where

- \( \text{responsible} \in \text{Resources} \) is the resource that is responsible for assign.
- \( \text{assignActSet} \) is a finite set of human activities with \( \text{assignAct} \in \text{AssignActs} \) that are crucial to complete the assignment.
- \( \text{workUnitCont} \in \text{WorkUnitConts} \) is the work unit container assign is attributed to.
- \( \text{basis} \in \text{AssignTempls} \) is the assignment template assign is based on.
- \( \text{state} \in \{\text{Inactive}, \text{Active}, \text{Finished}\} \) is the state of assign.
- \( \text{guidanceSet} \) is a finite set of guidances(cf. Chapter 12) used to support assign.
- \( \text{plannedStart} \in \text{DATETIME} \) is the planned start time for assign.
- \( \text{plannedEnd} \in \text{DATETIME} \) is the planned end time for assign.
- \( \text{actualStart} \in \text{DATETIME} \cup \text{NULL} \) is the actual start time for assign or undefined.
- \( \text{actualEnd} \in \text{DATETIME} \cup \text{NULL} \) is the actual end time for assign or undefined.
- \( \text{area} \in \text{Areas} \) defines the concrete area assign is attributed to.
- \( \text{contentInfo} \in \text{STRING} \) contains information for the human on assign.

Assigns describes the set of all definable assignments.

In order to keep track of its planned and actual execution, the assignment has four properties storing its planned and actual start and end times. In addition, it contains information on the assignment useful for the human processing it (\( \text{contentInfo} \)) and relations to guidances (cf. Chapter 12) for further support. Finally, it has a finite set of states whose transitions are depicted in Figure B-1.

**Figure B-1: Assignment states**

When a work unit container is created, its related assignment is created with state ‘Inactive’ as well. It then enters state ‘Active’ when one of its assignment activities is started by the human. If he switches to another assignment, it becomes inactive again until he switches back to it. The assignment enters its final state ‘Finished’ when its work unit container is finished. The assignment activities that are part of the assignment have similar properties as well as a finite set of states. The transitions between the states are depicted in Figure B-2.

**Figure B-2: Assignment activity states**

When an assignment is created, the related assignment activities get created, having state ‘Created’. When the work unit related to the assignment activity starts, the activity is available for the human, therefore it enters state ‘Inactive’. When the human starts processing it, it becomes ‘Active’. If he switches to another assignment it becomes ‘Inactive’ again until he switches back. When he finally completes it, it enters final state ‘Finished’.

The assignment activity is the planned human activity with the finest granularity in the CPM framework. However, it has connections to the more fine-grained activities, the atomic tasks. As opposed to the other activity concepts, the atomic task has no properties for planned times, but an actual start and end. However, atomic tasks are fine-grained and our experiences in real projects have shown that while processing an activity, a human frequently switches between the different tasks. Therefore, not only the absolute start and end times are recorded, but the overall duration
Conceptual Framework

(taskDuration) as well. The atomic task also has a set of states whose transitions are depicted in Figure B-3.

![Figure B-3: Atomic task states](image)

When an assignment activity is created, the corresponding atomic tasks (as defined by the template concepts) are also created and enter state ‘Inactive’. A task enters state ‘Active’ when the CPM framework detects its enactment or the human explicitly selects it. It becomes inactive again when another task is selected in the same way. From both states, there is a transition to final state ‘Finished’ in case the human finishes the corresponding assignment activity.

Artifact Management

Artifacts being part of the SE process as well as their mutual connections cannot be modeled properly within a WfMS using the data elements being part of the workflows. To add facilities to model artifact structures, we consider the concept of the project component template (cf. Definition B.4).

**Definition B.4 (Project Component Template)**

A **project component template** is a tuple projCompTempl = (type, name, reference, subCompSet, superCompSet, roleTemplSet, responsibleRoleTempl, reqCompTemplSet, stateSet, relatedCompTemplSet, areaTempl, compTemplType) where

- reference ∈ STRING ∪ NULL is a reference to the template of a real artifact (e.g., Specification) or undefined.
- subCompTemplSet is a finite set of project component templates that are subordinate to projCompTempl with projCompTempl ∈ ProjCompTempls.
- superCompTemplSet is a finite set of project component templates with projCompTempl ∈ ProjCompTempls that the projCompTempl is subordinate to.
- roleTemplSet is a finite set of role templates with roleTempl ∈ RoleTempls used to define one or multiple human roles according to projCompTempl.
- responsibleRoleTempl ∈ RoleTempls defines the main role template according to projCompTempl.
- reqCompTemplSet is a finite set of project component templates that projCompTempl requires with reqCompTempl ∈ ProjCompTempls.
- stateSet is a finite set of STRINGS used to define the possible states for projCompTempl.
- relatedCompTemplSet is a finite set of project component templates that has a content-related relation to projCompTempl with projCompTempl ∈ ProjCompTempls.
- areaTempl ∈ AreaTempls is the area template projCompTempl is associated to.
- compTemplType ∈ STRING is concretization of the type of projCompTempl.

The project component template is an abstract concept that generalizes more concrete sub concepts. Therefore, ProjCompTempls is defined as follows:

ProjCompTempls = ArtifactTempls ∩ SectionTempls. ArtifactTempls and SectionTempls are disjoint subsets of ProjCompTempls that are defined by:
- \( \text{artifactTempl} \in \text{ArtifactTempls} \) is a project component template for which the following applies:
  \( \text{artifactTempl}\).reference \( \neq \) NULL \( \land \) \( \text{artifactTempl}\).subCompTemplSet = \( \emptyset \).
- \( \text{sectionTempl} \in \text{SectionTempls} \) is a project component template for which the following applies:
  \( \text{sectionTempl}\).subCompTemplSet \( \neq \) \( \emptyset \) \( \land \) \( \text{sectionTempl}\).reference = NULL.

The project component template has a set of basic properties starting with a reference to the real entity it models (\( \text{reference} \)). In addition, it enables the definition of a set of role templates (\( \text{roleTemplSet} \)) and one role template responsible for the project component (\( \text{responsibleRoleTempl} \)). That way, a CPM framework can determine which human to inform (e.g., when there is a problem with an artifact). It further allows for content-related categorization by referring to an area template (\( \text{areaTempl} \)) and type (\( \text{compTemplType} \)). The latter might be for example ‘PDF file’ or ‘Java artifact’. Based on this type, the CPM framework can issue activities matching the project component (cf. Chapter 10). As opposed to the other concepts, the project component template allows defining a set of states the project components based on it may have during execution. This option has been introduced since many different types of artifacts in projects with a myriad of different states exist. Note that these states are not controlled by the CPM framework, but must be set by humans during enactment.

Another feature of the project component template is the possibility to add various relations to other project component templates. Such relations can be used to model various dependencies of artifacts as required by SE process models like the OpenUP [EcFo15]. On one hand, this enables a hierarchy of project component templates with the properties \( \text{subCompTemplSet} \) and \( \text{superCompTemplSet} \). On the other, content-related connections can be established using the \( \text{relatedCompTemplSet} \). Finally, the property \( \text{reqCompTemplSet} \) allows one project component template to require the presence of others.

Dynamic Processes

The concepts for defining dynamic events and reactions to them are discussed in this section. The most important concepts are, in this context, the extension point and the extension (cf. Chapter 7). For both of these, we present the template concepts in Definition B.5.

**Definition B.5 (Extension Point Template)**

An extension point template models templates for extension points to the work unit in a project. It is represented as a tuple \( \text{extensionPointTempl} = (\text{type, name, extensionType, extensionSubType, abstractionLevel, parallelInsertion}) \) where

- \( \text{extensionType} \in \text{ExtensionTempls} \) marks the type of extension template applicable to \( \text{extensionPointTempl} \).
- \( \text{extensionSubType} \in \text{STRING} \) marks the sub-type of extension template applicable to \( \text{extensionPointTempl} \).
- \( \text{abstractionLevel} \in \text{STRING} \) is the level of abstraction of extension templates applicable to this point.
- \( \text{parallelInsertion} \in \text{BOOLEAN} \) marks whether the extension template shall be inserted in parallel to the workflow instance the former is attached to or sequentially after it.

\( \text{ExtensionPointTempls} \) describes the set of all definable extension point templates.

The extension point template features content- and process-related information: the type of extension can be specified using the two properties (\( \text{extensionType, extensionSubType} \)). Further, there is property \( \text{abstractionLevel} \), which defines the abstraction level of the workflow in the entire process (e.g., operational development workflow vs. a workflow representing a phase of the process) to distinguish which extensions can be feasible. The extension point template corresponds to a marking of a change to a potentially running workflow instance. As discussed in Chapter 7 we apply a simple insertion into the workflow instance (i.e., Pattern AP1 from [WRR08]). For this pattern, three options for insertion exist: serial insert, parallel insert and conditional insert. The third option is redundant, as the added activity would be contemporarily inserted into the workflow instance matching the properties of the
situation. In such a case, no further condition is necessary. To distinguish between option one and two, the property \textit{parallelInstertion} is applied.

To classify the extensions made to the process, we further introduce the concept of the \textit{extension template} in (cf. Definition B.6).

\textbf{Definition B.6 (Extension Template)}

An \textit{extension template} models templates for extensions to process enactment. It is represented as a tuple $\text{extensionTempl} = (\text{type}, \text{name}, \text{assignmentTempl}, \text{extensionPointTemplSet}, \text{extensionSubType}, \text{abstractionLevel}, \text{skillLevelSet})$ where

- $\text{assignmentTempl} \in \text{AssignTempls}$ defines the concrete human assignment that marks the content of the extension based on extensionTempl.
- $\text{extensionPointTemplSet}$ is the set of extension point templates, to which extensionTempl is applicable with $\text{extensionPointTempl} \in \text{ExtensionPointTempls}$.
- $\text{extensionSubType} \in \text{STRING}$ marks the sub-type of extension applicable to $\text{extensionPointTempl}$.
- $\text{abstractionLevel} \in \text{STRING}$ is the level of abstraction for which extensionTempl is applicable.
- $\text{skillLevelSet}$ is a finite set of skill levels, one of which a human executing the extension shall possess with $\text{skillLevel} \in \text{SkillLevels}$.

The extension template corresponds to an abstract concept that generalizes more concrete sub concepts. Therefore, $\text{ExtensionTempls}$ is defined as follows:

$$\text{ExtensionTempls} := \text{FollwowActTempls} \cap \text{MeasureTempls} \cap \text{ExcHandTempls}.$$ $\text{FollwowActTempls}$ is the set of all definable Follow-up Activity Templates (used for activity coordination and detailed in Chapter 10), $\text{MeasureTempls}$ is the set of all definable Quality Measure Templates (used for software quality management and detailed in Chapter 9), and $\text{ExcHandTempls}$ is the set of all definable Exception Handling Templates (used for exception handling and detailed in Chapter 11). These three are disjoint subsets of $\text{ExtensionTempls}$.

The extension template features, same as the extension point template, a sub-type ($\text{extensionSubType}$) and abstraction level ($\text{abstractionLevel}$). Furthermore, it features a set of extension point templates for which it is applicable ($\text{extensionPointTemplSet}$) and a relation to an assignment template ($\text{assignmentTempl}$) that captures the human activity to be used to extend the process. In addition, it can also be specified, what skill level the human executing the extension should have ($\text{skillLevelSet}$). For more information on these properties and a detailed discussion of their application for integrating software quality measures into the process, we refer to Chapter 9.

The extension of a process can become necessary in many cases. We have discussed different cases for that in Chapter 4: task coordination (requirement $R:\text{Coord}$), process exception handling (requirement $R:\text{Exc}$), and software quality management (requirement $R:\text{Qual}$). In alignment with these cases and requirements, we have introduced three concrete sub-types of the abstract extension concept. These have been discussed in detail in the Chapters 9, 10, and 11.

\section*{B.2. Consistency Checks}

This section discusses the consistency checks and conditions we created for the CPM concepts. It is split up regarding the different areas the CPM framework covers. These checks are extensible and do not claim to be complete. They are a starting point influenced partly by sources from literature and experiences from practical settings.

\subsection*{B.2.1. Basic Concepts}

This section discusses consistency checks for the basic concepts applied for extending workflows.
Template and Individual Concepts

This check deals with the relation of template and individual concepts. Both concept sets share similar properties and the former set is used to pre-define the relations between concepts of the latter one. Therefore, individual concepts must not ignore these definitions. Figure B-4 illustrates a concrete case prohibited with this check. In this case, atomic task template ‘Coding’ is connected to the tool template ‘IDE’. However, a concrete individual has a connection to the static code analysis tool PMD instead.

Figure B-4: Consistency check: template properties

Work Unit Containers

For the work unit containers we apply consistency checks for various problems. Figure B-5 illustrates cases where properties of work unit containers have been set erroneously. Case a) deals with a work unit container without any work unit. In turn, case b) shows a work unit container requiring another one not contained in the same project. Such a container is out of control of the current project and hence does not contribute any results to it. Cases c) and d) concern work units that read or write project components not read or written by its container. The CPM framework’s definition implies that such components are exchanged with the container and distributed to its work units. Therefore, cases c) and d) should be prevented. In case e), a work unit container is defined to have no workflow instance but still has a connection to one. This collides with the definition of the ‘noWorkflow’ property and the workflow instance is redundant.

Figure B-5: Consistency check: work unit containers
Work Units

The definition of work units might contain certain erroneously set properties. Figure B-6 illustrates an undesired case for it. It concerns the usage of the work unit: It should be connected to a human-centric activity (assignment activity) or to a sub work unit container or work unit. If none of them is applied, the work unit will terminate right after its activation and would thus be useless.

Dependencies

The dependencies between work units and containers may imply erroneously specified properties interfering with correct execution. Figure B-7 illustrates three undesired cases. Cases a) and b) show different examples of circular dependencies with work unit dependencies (a) and work unit container dependencies. Such cases might produce deadlocks and should thus be prevented. A special case for the work unit dependency is shown in case c): If such a dependency is set to a work unit that is omittable, the dependency will not be satisfied if the work unit is be omitted. Therefore, we also prevent the setting of such dependency.

Variables

The variables used for governing the execution trace of the workflow instances are modeled in the context management component. This includes our concept for abstraction of internal workflow logic (cf. Chapter 7). The involved concepts might also imply erroneously set properties interfering with correct execution. Figure B-8 illustrates various cases for that. The connection to the variables in the WfMS can only be established if all variables are correctly mapped. Therefore, incorrect naming (case a) or incomplete mapping (case b) should be prevented. As the CPM framework does not monitor the correctness of all read and write operations on the variables and we have want to enable a standard trace for each executed workflow instance, each work unit container template must supply initial values for all variable templates (violated in case c). Similarly, each modeled human decision must have at least one decision alternative, otherwise the human might not make the decision (violated in
case d). For each of these decisions, a standard alternative may be defined to unburden the human from the decision. To prevent ambiguities, for each decision, there must be exactly one standard alternative (violated in case e). The decision alternatives are modeled as abstraction of the workflow variables. Therefore, each alternative must set at least one of the variables (violated in case f). Otherwise, the alternative will have no effect at all.

![Diagram of workflow concepts](image)

**Figure B-8: Consistency check: variables**

### B.2.2. Extrinsic Workflows

This section discusses modeling conditions and checks for the concepts realizing extrinsic workflows for SE issue processing.

**Modeling Conditions**

This section presents the modeling conditions enforcing properties on the building blocks that enable the creation of block-structured workflows from them.

**Condition C1**: Each workflow shall not have multiple start or end points. This promotes simple and understandable models as suggested in [MRv10]. Such a start or end point can be a single building block template or multiple building block templates that are connected in parallel.

**Condition C2**: Each activity shall have at least one connection to other activities. This condition ensures that workflows are buildable, as a workflow cannot be built from unconnected activities since it cannot be determined when to execute this activity. The exception from this condition are containers with only one contained activity. The latter shall have no connection to other activities as they are outside the container.

**Condition C3**: No cyclic sequencing shall be specified, as this is error-prone: It might be impossible to determine start and end point of a cyclic workflow. Furthermore, if a cycle were integrated in a workflow, there will be no clear exit condition for that cycle making execution nondeterministic. If activities are to be executed more than once, this shall be specified using the loop template.

**Condition C4**: The activity structure shall be simple. An activity shall have only one successor and one predecessor. If multiple successors are needed, one can be defined as successor and the other shall be specified as parallel to that successor. This limitation is introduced to support simplicity and understandability of the models. Furthermore, the specification of multiple successors of an activity without specifying how they should be executed (in parallel? conditional?) results in nondeterministic models. However, complex workflow modeling is enabled in a defined way using the specialized building block templates.

**Condition C5**: A building block template shall not be sequentially connected to another building block template to which it is also connected in parallel. Such a connection is inconsistent, specifying that it should be executed after (before) and parallel to the other building block template at the same time.
**Condition C6**: The different specialized building block template concepts (*sequence template, parallel template, loop template, and conditional template*) enable hierarchical specification of declarative workflows. The constraints utilized to structure the building block templates (hasSuccessor and hasParallel) shall be defined in a way that does not violate this hierarchical specification as this would make the structure more complicated and may even introduce inconsistencies. This implies that a building block template is not contained in two different other building block templates and that it has no connections to other building block templates that are not contained in the same building block template. Figure B-9 shows inconsistently specified examples. The inconsistent *loop template* shows a constraint (between activity 3 and 4) that violates hierarchical specification. Generation of a block-structured workflow is not possible, as the system would generate LOOP nodes around the activities 2 and 3, and activity 3 would have a connection with activity 4 that violates the block structure.

![Inconsistent concept examples](image)

**Condition C7**: A *loop template* shall only contain one building block template. This can be a simple activity or any other building block template, enabling the looping of any structures. This constraint prohibits inconsistent specification as shown with the inconsistent *loop template* in Figure B-9. That specification lacks a connection between activity 2 and 3. Simple modeling is again supported by defining that the *loop template* is for repetitive execution of a contained activity or a structure that is represented by another building block template.

**Condition C8**: A *parallel template* shall contain at least two building blocks. This condition is introduced to support simple and readable process models. A *parallel template* with only one contained building block template does not endanger workflow correctness. However, it would add unnecessary AND-splits and joins to the workflow.

**Condition C9**: A *parallel template* shall contain only building blocks that are connected in parallel. This constraint again supports simple hierarchical modeling, prohibiting confusing and error-prone structures as shown by the inconsistent *parallel template* shown in Figure B-9.

**Condition C10**: A *sequence template* shall contain at least two building blocks. This condition avoids specification of unnecessary building block templates, since a *sequence template* containing only one activity is similar to only specifying that contained activity without the *sequence template*.

**Condition C11**: A *sequence template* shall contain only sequentially connected building blocks. As with Condition C9, this condition supports a clear definition of the building block templates. A structure as shown by the inconsistent *sequence template* in Figure B-9 is thus prohibited as it also contains the parallel activities 3 and 4. On the other hand, it also has no specified connection between the parallel activities 3 and 4 and the sequential activities 5 and 6.

**Condition C12**: A *sequence template* shall contain a clear start and end point. This condition avoids cyclic dependencies of the activities in the *sequence template*.

**Condition C13**: A *conditional template* shall only contain unconnected activities or building block templates. This condition is applied because there will be only one or none of the contained building block templates selected for execution, and connections between them would thus produce inconsistencies.

**Condition C14**: A *conditional template* shall contain a minimal number of activities / building block templates: If the *conditional template* is defined as optional, it must contain at least one activity, else it would only add complexity to a workflow generating XOR-splits and joins with no contained activities as shown in Figure B-9. If the *conditional template* is not defined as optional, it must contain...
at least two activities since it would otherwise produce an inconsistent XOR pattern in the workflow containing only one branch.

Besides the sequencing constraints that are always only checked locally for the container or current building block template, there are also the existence constraints. These are in place for checking the soundness of a subset of activities that has been chosen due to contextual properties and are checked recursively for one container. However, to prevent modeling of containers that are inconsistent or foster inconsistent activity subsets, two conditions regarding the existence constraints are added to the build time checks:

**Condition C15**: One activity shall not both require and mutually exclude the same activity.
**Condition C16**: If an activity requires another activity, the latter must also be part of that container. If this is not the case, every activity subset containing the first activity will necessarily be inconsistent.

An additional constraint for the mutual exclusion constraint is not needed, as it is possible to integrate two mutually exclusive activities in the candidate set of one container. All activity subsets not containing both of them will then be consistent.

In the following, we describe a mapping of these conditions to concrete checks applied on the different concepts. These checks have been implemented as exemplarily shown in Chapter 13 for the sequence template.

The conditions for the sequence template realize the following subset of the aforementioned modeling conditions (cf. Figure B-10): hierarchically separated modeling (cf. C6) is checked (cf. case c). The other checks deal with the conditions that directly apply to the sequence template: The correct number of contained building block templates (cf. C10) is enforced (cf. case b) and the correct connections between these (cf. C11) is governed (cf. case a). Finally, the presence of a single start and end point within the sequence template (cf. case d and e) is enforced (cf. C12).

![Figure B-10: Consistency check: sequence template](image)

Similar checks are applied for the parallel template (cf. Figure B-11). Again, hierarchically separated modeling (cf. C6) is enforced (cf. case c). In addition, the correct connections between contained building block templates (cf. C9), and their correct number (cf. C8) is also checked (cf. case a and b).
The checks applied to the *loop template* also implement C6, enforcing hierarchically separated modeling (cf. Figure B-12 case b). In addition the correct number of contained building block templates (cf. C7) is also checked (cf. case a).

Concerning the *conditional template*, no separate check is required for implementing hierarchically separated modeling. A *conditional template* shall only contain unconnected building block templates (cf. C13 and Figure B-13 case b). The correct number of contained building block templates (cf. C14) is also checked (cf. case a).

Concerning the declarative container template, the presence of a single start or end point (cf. C1) is checked (cf. Figure B-14 case a and b). As defined in C1, both start and end point may contain multiple building block templates if they are connected in parallel. The presence of an unconnected building block template within a declarative work unit container is prohibited as well. This will only be permitted if the container contains exactly one building block template. In that case, the building block template will have to be unconnected (cf. C2 and Figure B-14 case c). Another check prohibits cyclic dependencies between contained building block templates (cf. C3 and case d). Furthermore, a consistent container must only contain consistent building block templates (cf. case e). Two other checks deal with the existence constraint. It is ensured that no building block template in a container requires and excludes the same activity (cf. C15 and case f). Finally, no building block template in a container shall require another building block template that is not part of the container or one of its contained building block templates (cf. C16 and case g).
B.2.3. Quality Management

This section discusses the realization of the agent structure utilized in Chapter 9 for automatic software quality measure prioritizing. The agent structure must be capable of both realizing the bidding process for the proactive measures and the voting process for the reactive measures. The bidding process shall favor agents whose goals are not in a good state. If this is the case, an agent takes place in the bidding process. If this applies for none of them, all can take place. If an agent wins one round, it may place one of his proactive measures in the list from which, at a later time point, measures for application will be selected. The voting process is different. Here, different agents vote on all measures in the reactive measure list that are attributed to their goal. That way, measures supporting multiple goals will have a higher probability to come to execution.

To be able to realize these two prioritizing processes, the agent structure is defined as depicted in Figure B-15. The AGQM agent is responsible for managing the multi-agent system component. It instantiates the other agents and determines whether a reactive or proactive measure will be proposed. For each defined goal, a goal agent is instantiated. In the proactive section, the goal agents communicate with the session agent to realize the bidding process. Thereby, the session agent takes the role of the “buyer” and thus selects the proactive measure from the goal agent with the highest bid. Each goal agent places bids according to its strategy. Initially, we have included three basic strategies. The strategies ‘offensive’, ‘balanced’ and ‘defensive’ influence the starting bid of the agents as well as win-or-lose adaptation based on the last bidding session. If insufficient points are left for the intended bid, the agent bids all points it has left. If an agent has no points left, it cannot place bids anymore until all agents have no points left, whereupon all points are reset to their initial value. Each agent has a list of proactive measures it could offer. Goals known to be at risk due to GKPI deviation are elevated to participation status in the bidding. If no report containing GKPI violations is received, all agents participate.
The reactive section is realized by the vote agent. Each time a report is received, the vote agent creates a weighted list of reactive measures using the report. To elicit the weight of each measure, the vote agent communicates with the goal agents. For each measure, a goal agent evaluates whether that measure is associated to its goal via the aforementioned connection of measures, metrics, KPIs, and goals. In each voting process, a goal agent distributes all of its points (initially allocated at the beginning of the iteration) uniformly to all measures in the current report that are associated to its goal. If multiple agents vote on one measure, the points are aggregated. If no report has been received yet, the voting process cannot be conducted. In that case, a proactive session is substituted. That way, the multi-agent system component creates a new ordered list of measures that mirror the predefined importance of the project’s quality goals.

**B.3. Algorithms**

This Chapter includes a set of additional algorithms not discussed in the main chapter of this work.

**B.3.1. Basic Workflow Enactment**

This section deals with algorithms needed for contextually extended workflow enactment as discussed in Chapter 7.

**Activity Marking**

This section shows algorithms for marking omittable and repeatable activities.

**Omittable Activities.** Activities in a workflow can be omitted due to the XOR pattern. In that case, there are points in the execution when it is clear that the execution of the respective activity will not happen in this instance of the workflow. These points correspond to the execution of other activities called *terminator activities* as described in Chapter 7. Algorithm B-1 is used to mark omittable activities and establish connections between an omittable activity and its terminator activity.

**Algorithm B-1: markOmitatable** (Pseudo Code for marking omittable activities)

```plaintext
Require: Decomposed Workflow list P {Blocks, Activities}, List targetBranch, List activitiesToConnect
1: for all elements in targetBranch do
2:     if not activitiesToConnect.empty() then
3:         connectNodes(element, activitiesToConnect, 'omittable')
4:     end if
5: if element ∈ blocks
```
The algorithm is explained in the following and graphically illustrated in Example B-1. The algorithm takes the decomposed workflow list discussed in Chapter 7 as input as well as a decomposed workflow list representing the point in the workflow where this execution of the algorithm should operate on. For the initial execution on a workflow, this will be the whole workflow. It also expects a list of activities, whose execution triggers the deactivation of a particular activity that is empty at the beginning (called terminator activities). The algorithm iterates through the workflow list and when there are terminator activities \((\text{activitiesToConnect})\) it marks the current activities as omittable and connects it bidirectionally with its terminator activities (Line 3) (cf. connectNodes() and the activities 1-5 in the example). This is needed for each activity when a workflow is executed later on. However, the algorithm also adds the markings to the blocks. These markings will be used to facilitate the making of activities that are inserted into the workflow when it is already running (cf. Algorithm B-2).

If the algorithm encounters a block, a new list is created (Line 5 and 6). This new list is used for new terminator activities of the encountered block and other blocks within it. This is done since the lists are passed as call-by-reference so that each level of the recursion has its own list that can also be used for further levels of the recursion but does not change the lists of upper levels of the recursion. That way, in Line 6 only the values in the list are copied (cf. e.g. in recursion Rec1 in the example). This is done because activities can be deactivated by multiple other activities. Consider e.g., multiple nested XOR patterns: An activity within an inner XOR pattern can be deactivated by activities of other branches of each of that XOR patterns. If a XOR block is encountered, the next step is the determination of the terminator activities for the current branch of that XOR pattern (Line 10 and 11) (cf. the initial call and the Rec2 recursion in the example). This is done by the algorithm getTerminatorActivities already described in Chapter 7. The algorithm is then called recursively for the current branch with the current childConnectActivities list and the new terminator activities for the current branch (Line 12). The same happens if another pattern as XOR is encountered (Line 16-20) (in that case without new terminator activities).

Example B-1 (markOmittable Steps):
For this example, the workflow used for Example 7-15 in Chapter 7 has been slightly adapted to contain two nested XORs to better demonstrate the XOR handling. Therefore, Figure B-16 and Figure B-17 show the adapted workflow and the concrete steps executed, both indicating the different recursion levels.
Since no activities follow the XOR1 pattern, the termination of the whole workflow is taken as terminator activity for all comprised activities. For activity 4, also the succeeding activity 5 is added.

<table>
<thead>
<tr>
<th>Call</th>
<th>Actions</th>
<th>childConn</th>
<th>termActs</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitCall</td>
<td>element = XOR1</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(Workflow L, targetBranch L,</td>
<td>childConn = activitiesToConnect</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>List activitiesToConnect AT)</td>
<td>termActs = new List()</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>termActs = getTA(L,L1,termActs,false,true)</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Rec1(L, L1, childConn+termActs)</td>
<td>element = AND1</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>childConn1 = activitiesToConnect</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Rec1.1(L, L1.1, childConn1)</td>
<td>element = 1</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>connectNodes(1, L)</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Rec1.2(L, L1.2, childConn1)</td>
<td>element = 2</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>connectNodes(2, L)</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Rec2(L, L2, childConn+termActs)</td>
<td>termActs = new List()</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>termActs = getTA(L,L2,termActs,false,true)</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Rec2.1(L, L2.1, childConn2)</td>
<td>element = 3</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>connectNodes(3, L)</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Rec2.2(L, L2.2, childConn2)</td>
<td>element = 5</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>connectNodes(5, L)</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Repetable Activities.** Due to the LOOP pattern, activities specified in a process model can be repeatable and can occur more than once in its execution. Therefore, they have to be marked so that the context management component is aware of this fact and can create new instances of the relating concepts when an activity is repeated. When activities are repeatable it may also be of interest to know when another execution of these can no more happen for a given workflow instance. This is a somehow similar case to the omittable activities and the XOR pattern: At certain points in the execution, it is clear that the respective looped activity will not be executed another time. This point is

![Figure B-16: markOmittable workflow](image-url)

![Figure B-17: markOmittable steps](image-url)
the execution of the first activity after the LOOP. In the case of multiple nested LOOP patterns this applies to the outer LOOP pattern. Due to the similarity to the markOmittable Algorithm we refrain from separately discussing the markRepeatable algorithm.

**Adaptation Markings.** As discussed in Chapter 7, newly inserted activities are analyzed and marked by a separate algorithm instead of re-running all initial marking algorithms. This involves different cases. First, there are different markings: the ‘repeatable’ marking, the ‘omittable’ marking, the list of activities an activity terminates, and the list of activities that are terminated by the activity. The first three markings apply for all activities of one branch while the last one, indicating an activity as a terminator activity, only applies for the first activity in a branch or the first activity in a branch after a XOR or LOOP pattern. Different situations require that the algorithm adopts the marking in different ways. These situations are explained in the following and illustrated in Figure B-18, starting with the generic case and showing the more specific cases afterwards (where this is a refinement, i.e., the generic cases also apply to the more specific cases):

1. **Inserted into a list, not as first element** (i.e., the list represents the workflow instance or a branch of a pattern): In this case, only the markings (repeatable, omittable, and the connection to the terminator activities) have to be adopted from any other activity in the branch. It is assumed that in this case, the list in which the activity has been inserted cannot have been empty before because it can only be the entire workflow instance, a branch of an AND pattern, or a LOOP pattern. None of these would make sense without any contained activities.

2. **Inserted as first element into a list** (representing the workflow instance or a branch of a pattern): Being the first element in the list, the insert activity can be the terminator activity for other activities. As it is assumed (as in case 1) that the list was not empty before, the connections to activities that are terminated by the current activity can be acquired from the former first activity in the list, which is now the second activity. This activity might also be a pattern containing multiple activities. However, for this algorithm this does not matter as the marking that have been previously applied to the workflow lists treat patterns (blocks) from the outside like simple activities and apply the same markings to them.

3. **Inserted into a list after a LOOP pattern in the same branch** (i.e., the list represents the workflow instance or a branch of a pattern): The activities in the LOOP pattern are repeatable and need terminator activities to indicate that they will not be repeated again. Therefore, the LOOP and all containing activities have to get the inserted activity be added as a terminator activity. Taking a naive approach, one might assume simply taking the markings from the successor of the inserted activity suffices. However, it is possible that it is not a successor in the workflow instance.

4. **Inserted into a list after a XOR pattern in the same branch** (i.e., the list represents the workflow instance or a branch of a pattern): In principle, this is the same case as the previous one. However, XOR patterns have one special property: If they have an empty branch it is possible that no activity of the XOR pattern comes to execution. This, in turn, implies that the newly inserted activity must also be added to the activities of a XOR or LOOP pattern directly before the XOR pattern that is the predecessor of the inserted activity.

5. **Inserted into a list that represents the empty branch of a XOR pattern**: In this case, no activity is in place in the list to adopt the markings from. Therefore, the markings can be adopted from an activity in another branch and mutual terminator activity markings have to be established between the branches.
In the following, Algorithm B-2 is presented that applies the markings for a newly inserted activity:

**Algorithm B-2: markInsertedActivity** (Pseudo Code for marking newly inserted activities)

```plaintext
Require: List targetBranch, Activity target, Block surroundingPattern
1: if not targetBranch.size == 1
2:   Element element ← targetBranch.getPreviousElement(target)
3:   if element == NULL
4:     element ← targetBranch.getNextElement(target)
5:   end if
6:   adoptMarkings(target, element)
7: end if
8: if targetBranch.firstElement == target
9:   if not targetBranch.size == 1
10:   connectTerminatorActivity(target, targetBranch.getNextElement(target))
11: end if
```

Figure B-18: Marking cases for inserted activities
Algorithm B-2 takes as input a newly inserted activity, its branch, and the pattern surrounding that branch. First, the algorithm deals with case 1, which is the simplest case: In Line 1-8 it inherits the markings (repeatable, omittable, and potential terminator activities) from another activity in the same branch. This is only done if the new activity is not the only one in the branch, which might be the case if the surrounding pattern is a XOR pattern. Case 2 is dealt with in Lines 9-12: if the new activity is the first in the target branch, the list of activities it terminates is taken from the former first activity in the branch that is now the second one. The next case processed involves insertion within the empty branch of a XOR pattern (case 5). In this case, all markings are adopted from the first activity of another branch (Lines 13-20). This is done to establish connections to the other activities that are outside of the XOR pattern because the first activity of each branch of a XOR pattern has equivalent relations to activities that are outside of the XOR pattern. The mutual marking of the activities of the different branches in the XOR pattern are then applied in Lines 21-26: First, the first activity of each branch is added to the terminator activities of the newly inserted one. The latter is then added to the terminator activities of all activities in the other branches in the XOR. The final part of the algorithm (Lines 30-44) deals with cases 3 and 4. It takes the predecessor of the inserted activity and, if it is a LOOP or XOR, it adds the new activity to its terminator activities. In case of an XOR, this action is repeated. The function used to add the terminator activity (in this case addAsTerminatorActivity()) also applies the marking recursively to all activities contained in the LOOP / XOR.

Computational Complexity of the Algorithms

To conclude this section regarding algorithms, we will elaborate briefly on their computational complexity. For most of them, however, this is not a critical issue as they are applied during build time. Furthermore, their complexity depends on the elements in the modeled workflows and the

```java
13: if surroundingPattern == XOR and targetBranch.size == 1
14: 
15: Boolean outerMarkings ← false
16: for all surroundingPattern.branches do
17: 
18: if not branch == targetBranch
19: 
20: adoptMarkings(target, branch.getFirstActivity)
21: outerMarkings ← true
22: end if
23: target.omittableTerminators.add(branch.getFirstActivity)
24: branch.getFirstActivity.terminatesActivity.add(target)
25: for all branch.activities do
26: activity.omittableTerminators.add(target)
27: target.terminatesActivity.add(activity)
28: end if
29: end for
30: end if
31: Element prevEl = targetBranch.getPreviousElement(target)
32: while not prevEl == null
33: if prevEl == (LOOP or XOR)
34: addAsTerminatorActivity(prevEl, target)
35: if prevEl == XOR and containsEmptyBranch(prevEl)
36: prevEl ← targetBranch.getPreviousElement(prevEl)
37: else
38: prevEl ← NULL
39: end if
40: else
41: prevEl ← NULL
42: end if
43: end while
44: end if
```
number of these elements is recommended to be kept rather small for various reasons. For example, [MRv10] recommends to keep the number of nodes in a workflow below 50. Our practical experiences show that it is very uncommon that a huge number of workflows or workflows with a huge number of elements are created in a modeling session. Also, only one of the algorithms (markInsertedActivity) is to be executed during runtime and this might impact operational performance. Therefore, we have put emphasis on a low complexity for this algorithm. In Table B-2 the complexity of the different algorithms is shown.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>decomposeWorkflow</td>
<td>$O(#\text{nodes in workflow})$</td>
</tr>
<tr>
<td>markOmittable</td>
<td>$O(#\text{elements in list} \times #\text{XORs.branches})$</td>
</tr>
<tr>
<td>markRepeatable</td>
<td>$O(#\text{elements in list} \times #\text{LOOPS})$</td>
</tr>
<tr>
<td>getTerminatorActivities</td>
<td>$O(#\text{elements in list} \times #\text{element.branches})$</td>
</tr>
<tr>
<td>markInsertedActivity</td>
<td>$O(#\text{surroundingPattern.branches} \times #\text{branch.activities} + #\text{preceding LOOPS or XORS})$</td>
</tr>
</tbody>
</table>

The algorithm ‘decomposeWorkflow’ directly depends on the number of nodes in the analyzed workflow. The other three build time algorithms depend on the number of elements in the output list of the first workflow, as well as on the number of branches of the workflow patterns. However, as the build time algorithms are executed in a row or respectively call each other, an overall computational complexity for analyzing one modeled workflow can be expressed as follows:

$$O(#\text{nodes} + (#\text{elements in list} \times #\text{XORs.branches}) \times (#\text{elements in list} \times #\text{element.branches}) + (#\text{elements in list} \times #\text{XORs.branches}) \times (#\text{elements in list} \times #\text{element.branches})) = O(#\text{nodes} + (#\text{elements in list})^2 \times #\text{XORs.branches} \times #\text{element.branches} + (#\text{elements in list})^2 \times #\text{LOOPS} \times #\text{element.branches}) = O(#\text{nodes} + (#\text{XORs.branches} + #\text{LOOP}) \times (#\text{elements in list})^2 \times #\text{element.branches})$$

Having the properties of the workflows just discussed in mind, this complexity seems quite adequate and should not hamper modeling. Nevertheless, we have managed to realize ‘markInsertedActivity’ with a much smaller complexity, as it is to be executed during runtime for every activity inserted into a potentially running workflow instance. For brevity, we omit a separate discussion of the algorithms for extrinsic workflow generation as they operate on similar structures. Furthermore, extrinsic workflows are mostly smaller than intrinsic ones as they are enacted.

**B.3.2. Extrinsic Workflow Generation**

This section presents algorithms related to the generation of workflows from the declarative specification we have introduced in Chapter 8.

The algorithm $\text{BBtreatment()}$ is utilized to convert building blocks into parts of an executable workflow. The conversion is abstracted (from the creation of context and process management concepts) using simple functions as, e.g., $\text{insertNode()}$ for the insertion of one activity into a workflow.

**Algorithm B-3: BBtreatment**

(Pseudo Code for inserting building blocks into workflow)

```plaintext
Require: Building Block BB, Work Unit Container skeleton, Arc marker
Return: String errorCode
1: String errorCode ← empty String
2: if BB ∈ Activities
3: insertNode(BB, skeleton, marker)
4: else if BB ∈ Sequences
5: errorCode ← sequenceTreatment(BB, skeleton, marker)
```
6: else if $BB \in \text{Parallels}$
7:     $\text{errorCode} \leftarrow \text{parallelTreatment}(BB, \text{skeleton}, \text{marker})$
8: else if $BB \in \text{Loops}$
9:     $\text{errorCode} \leftarrow \text{loopTreatment}(BB, \text{skeleton}, \text{marker})$
10: else if $BB \in \text{Conditionals}$
11:     $\text{errorCode} \leftarrow \text{conditionalTreatment}(BB, \text{skeleton}, \text{marker})$
12: end if
13: return $\text{errorCode}$

The algorithm expects a building block as well as the workflow skeleton to be extended including a position marker as input. If that building block is a simple activity, it is inserted into the workflow. If it is of another type, the insertion is handled by specialized algorithms. One of these, $\text{parallelTreatment}()$ is exemplarily discussed in Algorithm B-4.

Algorithm B-4: parallelTreatment
(Pseudo Code for inserting a parallel into a workflow)

Require: Building Block BB, Work Unit Container skeleton, Arc marker
Return: String errorCode
1: String $\text{errorCode} \leftarrow \text{empty String}$
2: if $BB.\text{parallelBBset}$ is empty
3:     return "$\text{emptyParallel}\$
4: else if $BB.\text{parallelBBset}.\text{size} < 2$
5:     $\text{errorCode} \leftarrow \text{BBtreatment}(\text{parBB}, \text{skeleton}, \text{marker})$
6: else
7:     $\text{AndSplit} \text{ split} \leftarrow \text{insertParSplit}(\text{marker}, \text{skeleton})$
8:     List $\text{branches} \leftarrow \text{new List}()$
9:     for all $BB.\text{parallelBBset}$ do
10:       $\text{insertBranch}(\text{split}, \text{marker})$
11:       $\text{errorCode} \leftarrow \text{BBtreatment}(\text{parBB}, \text{skeleton}, \text{marker})$
12:       $\text{branches}.\text{add}(\text{marker})$
13: end for
14: $\text{insertParJoin}(\text{marker}, \text{split}, \text{skeleton}, \text{branches})$
15: end if
16: return $\text{errorCode}$

parallelTreatment inserts no pattern if no building block is contained in the parallel. If it contains only one building block, no pattern is needed either but only the building block is inserted. If multiple building blocks are contained, each is added in a separate branch of an AND pattern.

For brevity, we will refrain from discussing the computational complexity also for the extrinsic workflow generation. As stated in Chapter 8, the modeled workflows can contain many activities. However, for the workflow generation algorithms, the number of activities that are really in place for a specific situation is important. Usually, this is a rather small subset of the modeled activities.
C. Basic Actions for Process Enactment

This appendix discusses concrete actions to be performed in order to enact processes with the CPM framework.

Create Project

When a project and its process realized by a structure of workflows have been created with the template concepts, that structure can be used for concrete project executions. Therefore, a concrete project including its process must be created in the CPM framework. The action for this is shown in the following. Note that we assume that a project only has one defined process. This is an abstraction that may not hold for all projects, however, multiple processes for one project can be added with low effort within the CPM framework.

This action is applied to create a new project with its associated work unit container.

Preconditions: -
Input: project template ∧ roles, project components, and tools required by the assigned work unit container template.

Actions:
- Create project concept.
- Create contained areas as defined in template
- Apply: Create Work Unit Container

Output: project with work unit container in state ‘Created’.

The ‘Create Project’ action implies the creation of its associated process that is captured by one basic work unit container (and its potential sub work unit containers). The latter is created by the following action ‘Create work unit container.

Create Work Unit Container

This action is applied to create a new work unit container from a work unit container template. As opposed to WfMS where workflow instances are directly started from their templates, the containers in the CPM framework are created without starting them (or the relating WfMS workflow instances). Thus, a workflow structure for the complete process of a project can be created without having to start each of the future workflow instances.

Preconditions: -
Input: work unit container template ∧ values for roles, project components, and tools

Actions:
- Create work units as defined in template and assign to work unit container.
- Create assignment as defined in template and assign to work unit container.
- Create assignment activities as defined in template and assign to assignment.
- Create atomic tasks as defined in template and assign to assignment activities.
- Assign concrete tools to atomic tasks as defined in template.
- Set process variables as defined in the template.
- Assign concrete humans for the container roles.
- Assign concrete inputs/outputs for container (including structure of project components as defined in super/subCompsSet properties).
Basic Actions for Process Enactment

- Assign main human with main role also to the assignment. Distribute the humans filling the roles of the container to the work units. Add responsible party of each work unit to the relating assignment activity.
- For all defined dependencies defined by the template for work units, apply the action ‘Create work unit container’ to create the containers (and work units) that are the targets of the dependencies and then connect them via the ‘Add work unit to work unit dependency’ and ‘Add work unit to container dependency’ actions.

Output: work unit container in state ‘Created’.

After a concrete work unit container has been created, it remains in the state created and also does not automatically initiate the start of its relating workflow instance. This has the advantage that for a project, its whole process can be prepared with a workflow structure without having to start one or more of the involved work unit containers or workflow instances. When all concepts and information is in place, a work unit container can be explicitly started including the creation / start of its relating workflow instance. To start a project, its top-level container thus has to be started. The action to execute such a start is shown in the following.

Start Work Unit Container

This action is applied to start a work unit container.

Preconditions: work unit container must be in status ‘Created’.

Input: work unit container

Actions:
- Instantiate a new workflow instance from a workflow template that is connected to the template of the current work unit container.
- Connect the workflow instance and the work unit container.
- Set work unit container status to ‘Started’.

Output: work unit container in state ‘Started’.

When a container and its associated workflow instance is running, its progress is governed by the work units that are the mappings of the activities in the workflow instance. With these, connections to sub-containers or human tasks (assignment activities) are managed as discussed in Chapter 7. Thus a sound management of their states and especially their termination is crucial as the workflow instance can only continue when one or more active work units terminate. Therefore, the action for checking if a work unit may terminate is explicitly defined in the following.

Check Work Unit Termination

This action is applied to check if a work unit can terminate.

Preconditions: work unit must be in state ‘Started’.

Input: work unit

Actions:
- Check if associated assignment activity is finished.
- Check if required guidance has been used, i.e. guidance in ‘guidanceSet’ (of related assignment activities or project components or, if it is the final work unit of the assignment, also the assignment) are satisfied.
- Check dependencies of work unit are satisfied, i.e. if they are ‘finalized’ (or, in case of a work unit dependency with ‘oneShot’ behavior ‘executed’).

Output: work unit in state ‘Started or ‘Finished’.

As discussed in this chapter, the CPM concept applies multiple instances of the concepts relating to a looped activity in the WfMS. That means, if a WfMS activity is executed repeatedly due to a loop, the relating work unit and related concepts are already finished. So a new work unit instance has to be created, supplied with the values for markings and human activities from the prior work unit instance,
and has to be linked with the latter as well as with other containers and work units the prior work unit instance had dependency connections to. This is managed explicitly by the following action.

**Create new Work Unit Instance**

This action is applied to create a new instance of a work unit if the relating workflow activity is executed multiple times in a loop.

*Preconditions:* work unit must be in state ‘Finished’ and relating WiMS activity comes to execution again.

*Input:* work unit

*Actions:*

- Create new work unit and relating activity concepts and adopt values from the prior work unit.
- If the work unit is defined for single execution by the `singleExec` property only create the work unit without any other concepts and start it. That way, the new instance will terminate immediately like a ‘blind activity’ that will have no effect on the container and be invisible to the human.
- Check if work unit has a dependency. If yes, create the same dependencies for the new work unit. If the target container (or the container containing the target work unit) has a planned successor iteration (via the `futureExec` property), take that container as the target and link it with the new work unit. If there is no future iteration, create a new container with the values in place and link it with the new work unit.
- Check for a dependency who had the prior work unit instance as a target. If there are one or more of these having the `lastShot` behavior, link the dependency to the new work unit instance.
- Start the work unit.

*Output:* new work unit in state ‘Started’.

As discussed many times in this work, SE process enactment is rather dynamic and thus changes to the workflow structure of a project might often be necessary. For example, a new workflow instance / work unit container may have to be added due to a changed or new customer requirement that needs to be realized. Such a new container has to be integrated into the workflow structure by adding new dependencies between that new container and a container that is part of the workflow structure. Therefore, in the following, concrete actions for adding such dependencies are shown starting with the dependency of a work unit to another container.

**Add Work Unit to Container Dependency**

This action is applied to create a new dependency between a source work unit and a target work unit container.

*Preconditions:* Source work unit and target work unit container must be in state ‘Created’ or ‘Started’.

*Input:* source work unit ∧ target work unit container

*Actions:*

- Create dependency.

*Output:* newly connected work unit and container as illustrated in Figure C-1.

![Figure C-1: Add work unit to container dependency](attachment:image.png)
The addition of a dependency from a work unit to another work unit in another container is also possible as shown in the following.

**Add Work Unit to Work Unit Dependency**

This action is applied to create a new dependency between a source work unit and a target work unit.  
*Preconditions:* source and target work units in state ‘Created’ or ‘Started’. 
*Input:* source work unit ∧ target work unit ∧ definition of behavior of new dependency 
*Actions:*  
- Create dependency.  
*Output:* newly connected work units as illustrated in Figure C-2.

![Figure C-2: Add work unit to work unit dependency](image)

Changes to the workflow structure of a project may not only imply adding new requirements and additional workflow instances. It is also possible that, for example, a requirement can be canceled because its realization turns out to be unfeasible or too expensive. In such a case one or more containers might have to be excluded from the workflow structure. This implies removing dependencies between containers and / or work units. The actions for this are shown in the following starting with the removing of a dependency to a container.

**Remove Container Dependency**

This action is applied to remove mutual dependencies between work units in a source work unit container and a target work unit container. 
*Preconditions:* source and target in state ‘Created’ or ‘Started’. 
*Input:* source work unit container ∧ target work unit container 
*Actions:*  
- For all dependencies of work units in the source container to work units in the target container where the source and target work units are in state ‘created’ or ‘running’, apply: **Remove work unit dependency**.  
*Output:* -.

Same as for a container, a work unit can also have a dependency to another work unit that also might need to be removed. The relating action is shown in the following.

**Remove Work Unit Dependency**

This action is applied to remove a dependency between a source work unit and a target work unit or a container. 
*Preconditions:* source and target in state ‘Created’ or ‘Started’. 
*Input:* source work unit ∧ (target work unit container ∨ target work unit) 
*Actions:*  
- Delete dependency 
- For the source work unit apply: **Check Work Unit Termination**.  
*Output:* -.
Another frequent change we have perceived in the projects of our industry partners is the moving of an activity / requirement / workflow instance from one point in the process to another. This happens in iterative development, when an activity is to be executed within one iteration but cannot be finished therein. Iteration deadlines are mostly firm and thus the activity (and its workflow instance) is transferred to another iteration. To facilitate this, the following action shows the moving of dependencies.

**Move Work Unit Dependency**

This action is applied to move a dependency between an old source work unit and a target work unit or a container to a new source work unit.

*Preconditions:* old and new source and target in state ‘Created’ or ‘Started’.

*Input:* old source work unit ∧ new source work unit ∧ (target work unit container ∨ target work unit)

*Actions:*

- If target is a work unit apply for new source and target: **Add Work Unit to Work Unit Dependency**, else apply **Add Work Unit to Container Dependency**.
- If target is a work unit apply for old source and target: **Remove Work Unit Dependency**, else apply **Remove Container Dependency**.

*Output:* newly connected concepts as illustrated in Figure C-3 (upper half for container dependency and lower half for work unit dependency).

![Diagram](https://via.placeholder.com/150)

Figure C-3: Move work unit dependency

One thing that also happens frequently in projects is the situation that activities must be handed over from one human to another. The cause for this might be for example the unavailability of one human or that an activity has been assigned to a whole team and the team leader then passes it on to a concrete human best suitable for the activity. For such cases, we have applied two different actions for distributing human activities. The first one, shown in the following, deals with the distribution of one concrete assignment activity from one human to another.

**Distribute Activity**

This action is applied to change the executing human of an assignment activity.

*Preconditions:* assignment activity must be in status ‘Created’, ‘Active’, or ‘Inactive’.

*Input:* assignment activity ∧ human

*Actions:*

- Remove executing human.
- Set new human as executor.

*Output:* assignment activity with new executor.
Another specific case is the distribution of a more complex activity, an assignment, from one human to another. In this case, the assignment might already be started and all comprised assignment activities must still be transferred to the new executor as shown in the following.

**Distribute Assignment**

This action is applied to change the executing human of an assignment and all of his related assignment activities belonging to that assignment.  
*Preconditions:* assignment in state ‘Active’ or ‘Inactive’.  
*Input:* assignment ∧ old executor ∧ new executor  
*Actions:*  
- Remove executing human of assignment.  
- Set new executor as executor of assignment.  
- For all assignment activities having the old executor and that are in state ‘created’ or ‘running’ apply **Distribute activity** with the new executor.  
*Output:* assignment with new executor.