Providing Automated Holistic Process and Knowledge Assistance during Software Modernization

Gregor Grambow
Ulm University, Germany

Roy Oberhauser
Aalen University, Germany

Manfred Reichert
Ulm University, Germany

ABSTRACT

Software modernization remains a difficult, highly intellectual, labor-intensive, collaborative, and risky undertaking involving software engineers interacting in knowledge-centric processes. While many tools and several methodologies are available, current modernization projects lack adequate automated and systematic operational process support. This chapter provides an introduction to the topic of automated process and knowledge assistance for software modernization, giving background information on related work in this area, and then expounds on various problems. To address these, a holistic solution approach and guidance framework called the Context-aware Software Engineering Environment Event-driven framework (CoSEEEK) is described, which can support developers on software modernization projects, addressing such aspects as process dynamicity, extrinsic processes, process exception handling, coordination, quality assurance, and knowledge provisioning. Subsequently, future research directions are discussed and a conclusion is drawn.

INTRODUCTION

Software applications are continuing to grow in size, with one recent study indicating that applications typically double in size as measured in lines of code every 4-5 years (van Genuchten & Hatton, 2012). Size growth results in increasing complexity and an increase in the total number of defects (Koru, Zhang, El Emam, & Liu, 2009). Over time, legacy applications continue to face additional challenges (Mens et al., 2005), including increasing software maintenance costs, quality assurance issues, and architectural and technological erosion (Ducasse & Pollet, 2009; Tan & Mookerjee, 2005). Software modernization is challenged with modernizing the language and/or technologies used in a legacy application, while retaining its previous functionality, and as such has a key place within the Software Maintenance knowledge area of the Software Engineering Body of Knowledge (SWEBOK) (Abran & Bourque, 2004). Yet many reengineering efforts fail because they are unsystematically executed using ad-hoc processes (Weiderman, Bergey, Smith, & Tilley, 1997). Various software modernization strategies, processes, methodologies, and techniques have been developed over the years, and the appropriate selection,

tailoring, and application to a given modernization situation is dependent on many factors, including organizational constraints, budget, time, and resource constraints, experience, available competencies, risk, urgency, criticality, etc. (ISO/IEC 14764, 2006). Aspects involved in modernization can include business process archaeology, process mining, business process modeling, reverse engineering, software architecture, model-driven software engineering, knowledge management, and static and dynamic code analysis.

Software modernization remains a risky, difficult, and highly intellectual process incorporating various automated and manual tasks performed by specialized knowledge workers. The process lacks sufficient automated and systematic operational process support. The intangibility of the product represented in its source code artifacts poses a significant comprehension challenge to software engineers. To support the latter in their modernization tasks, various tools such as tools related to Model-Driven Architecture™ (MDA) (Mukerji & Miller, 2003) from the Object Management Group (OMG), and various processes such as the (ISO/IEC 14764, 2006), Service-Oriented Migration and Reuse Technique (SMART) (Lewis, Morris, & Smith, 2005), and eXtreme end-User dRiven Process (XIRUP) ("D31b", 2008) from the MOdel driven MOdernisation of Complex Systems (MOMOCS)[i] project. Mostly, the former are utilized for supporting the project participants' individual tasks while the latter aim to help organize the necessary collaboration and sequencing of work. However, this lack of integration may also be a crucial obstacle for holistic and technically supported process enactment in the software engineering (SE) domain: On the one hand, projects and their processes are planned using abstract process models, on the other hand the concrete tasks are executed by different people working with different software tools and techniques in order to manipulate a large number of different artifacts (e.g., specifications, user documents, source code artifacts, or tests). The tools are crucial for the successful completion of the different tasks, yet their usage is neither directly connected to the process nor to each other. This promotes an ever-growing gap between the planned and the actually executed process. This disparity has negative effects for modernization projects and their produced software: First of all, proactive and reactive software quality techniques are often not systematically and satisfactorily integrated, since the actually executed operation process is unknown. Project planning with management tools (e.g., in visual form such as a Gantt chart) typically does not provide process governance nor process execution support. Since quality assurance is not systematically and automatically integrated, important quality actions (perhaps involving tools for static code analysis, profiling, or test remediation) may not be executed in a timely fashion, and quality deterioration may go undetected.

An important part of modernization is recovering specific knowledge from existing software assets. Business Process Archeology (Pérez-Castillo, de Guzmán, Piattini, & Ebert, 2011) extracts abstract models of these legacy systems, including the company and the company operation supported by this system. For example, business process models, which can be quite complex and be tied to various artifacts and should be well understood during the modernization effort by the software engineers. For this, knowledge management and provisioning is also a crucial factor for successful modernization, yet it is rarely if ever systematically integrated at the operational level. E.g., knowledge may be recorded in wikis or documents, but since it is not automatically situationally associated with source code, tools, and processes, it must currently be manually retrieved by a software engineer when he or she perceives the need and triggers a search or retrieval, and effective and timely situational knowledge dissemination opportunities are missed.

Besides these concrete problems, more general issues exist caused by disparity of the abstractly planned process and the actually executed process incorporating various humans, activities, tools, and artifacts. Some important tasks executed within a modernization project may not be explicitly covered by the process (e.g., debugging) and therefore remain neither planned nor traced. The same applies for dependencies between activities that are part of the process and others that are extrinsic to the process (e.g., open issues or questions). This also has an impact on the ability of the projects or organizations to cope with process-related exceptions. These can be of complex nature involving multiple actors, tools and artifacts, thus

concealing the relation between cause and effect. Additionally, for cost and resource availability reasons, the trend towards global software development may also affect modernization projects, which could benefit from automated support for global collaboration in the modernization processes.

Towards addressing the aforementioned challenges, a tighter connection between the planned process and concrete tooling and artifacts is desirable. To achieve this, we pursue an abstract holistic concept and implementation framework named CoSEEEK (Context-aware Software Engineering Environment Event-driven frameworK). Its main idea is the contextual, semantic extension of adaptable operational processes, collaboration support, and the automatic integration of SE tools, knowledge, and guidance to be able to consistently manage how people execute this process and manipulate various artifacts via tools. This is illustrated in Figure 1:

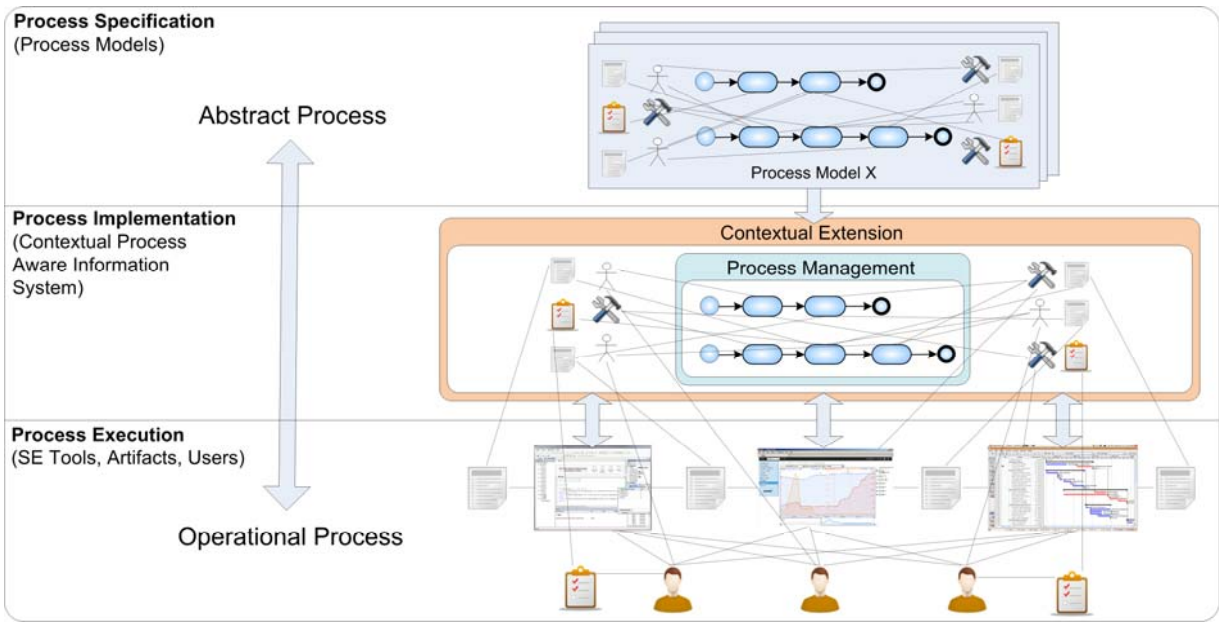

Figure 1. Connection of abstract and operational processes

On the abstract level, processes are specified by the role of a process engineer by means of process models. These models contain the processes to be executed as well as specifications of utilized artifacts, tools, roles, and additional support information such as checklists. On the operational level, there are users utilizing SE tools such as IDEs, configuration management tools, static code analysis tools, and artifacts like specifications, source code, or checklists. Our concept and system enable a direct connection of the abstract and the operational areas, integrating knowledge such as process guidance with SE tools and artifacts for concrete projects and organizations. This is achieved by contextually extending a Process-Aware Information System (PAIS), which not only enables the explicit modeling of artifacts, tools and users, but also their direct connection to the activities governed by the PAIS (Reichert & Weber, 2012). Furthermore, via a sensor framework, the actual SE tools in use are integrated together with target artifacts connected to the model in the contextually extended PAIS.

Overall, this chapter demonstrates an approach to automatically, dynamically, and systematically support modernization process execution and collaboration. Required tools, knowledge, and artifacts are directly interwoven within abstractly defined processes and, in turn, operationally governed and supported by our

framework. Various publications from the authors and a forthcoming doctoral thesis of Gregor Grambow are utilized, summarized and applied to the software modernization domain.

BACKGROUND

State-of-the-Art

Modernization as a form of software evolution is differentiated from maintenance and from replacement in (Comella-Dorda, Wallnau, Seacord, & Robert, 2000b), which describes various modernization techniques for user interfaces, logic, and data. Software engineers involved in modernization projects typically utilize a potpourri of various methodologies, processes, tools, and techniques. To provide an overall understanding of the modernization undertaking, various modernization-related approaches and aspects are described.

Model-centric standards and associated model and code-generation tooling can support modernization. MDA (Mukerji & Miller, 2003) seeks to standardize portable and interchangeable design models that can be technology and platform independent. To support model transformations, a Meta-Object Facility (MOF) is specified. According to the MDA approach, legacy systems would have a platform-independent model (PIM) of the application or specification's business functionality and behavior that could be more easily modernized by transforming this PIM to newer platform-specific models (PSM) for new platforms and technologies. Model-Driven Software Development (MDSD) is broader, including both additional alternative approaches to MDA and various MDSD tooling, for instance the Eclipse Modeling Framework (EMF) project[ii].

Various approaches to modernization include Software Evolution, Refactoring and Improvement of Operational & Usable Systems (SERIOUS)[iii], which studied evolutionary software development techniques, tools and processes. Additional model-driven approaches include Risk-Managed Modernization (RMM) (Seacord, Plakosh, & Lewis, 2003) and XIRUP from the MOMOCS project.

Knowledge acquisition during modernization typically encompasses various analysis techniques. One form of categorization that distinguishes the degree of internal knowledge required is black-box vs. white-box recovery and transformation. Black-box (Comella-Dorda, Wallnau, Seacord, & Robert, 2000a) focuses on awareness of the external component properties and interfaces while white-box recovery includes awareness and analysis of internal structures and possibly code source when available (Weiderman, Bergey, Smith, & Tilley, 1997).

Various types of tools support knowledge acquisition during modernization as described in (Pérez-Castillo, de Guzmán, Piattini, & Ebert, 2011). To facilitate modernization tool interoperability and data interchange, the OMG's Architecture Driven Modernization (ADM)[iv] initiative standardizes metamodels in seven modernization-related areas. (Pérez-Castillo, de Guzman, Piattini, & Avila-Garcia, 2009) makes use of ADM to contextualize the data about legacy source code. MARBLE (Pérez-Castillo, 2012; Pérez-Castillo, de Guzman, & Piattini, 2011) is an example of a tool that supports business process model recovery while (Putrycz & Kark, 2007) addresses recovery of business rules. For extracting models, the use of domain-specific programming languages such as Gra2MoL (Cánovas Izquierdo & Molina, 2009; Izquierdo, Cuadrado, & Molina, 2008) has been proposed. The proper utilization of acquired knowledge plays a key role in decision-making during modernization (Koskinen et al., 2005; Koskinen, Lintinen, Ahonen, Tilus, & Sivula, 2005).

For building an environment supporting collaborative work, CASDE (Jiang, Ying, & Wu, 2007) and CooLDev (Lewandowski & Bourguin, 2007) utilize activity theory. CASDE features a role-based awareness module managing mutual awareness of different roles. CooLDev is an Eclipse IDE plug-in that manages activities performed with other plug-ins in the context of global cooperative activities. CAISE (Cook, Churcher, & Irwin, 2004) is a collaborative SE framework with the ability to integrate SE tools and develop new SE tools based on collaboration patterns.

The aforementioned approaches and tools primarily focus on knowledge extraction within modernization, and lack holistic, process-oriented, and collaborative knowledge and quality assurance support for the entire modernization lifecycle.

Terminology

This section discusses and clarifies important terms and basic concepts that will be used throughout the chapter.

Process and Workflow. By process we mean a systematic series of actions or steps, which can involve one or more workflows, taken towards achieving some end state or result that adds value to an organization. By workflow we mean a more detailed series of related tasks required to complete some portion of a process, usually detailing the concrete tasks, procedures, steps, knowledge, inputs, outputs, people, and organizations involved to process a piece of work. It can be viewed as the partial or complete technical implementation of a process.

Context / Contextual Integration. The notion of context, as used in this chapter, refers to the various facts, events, and entities that are part of a software project. They have in common that the projects participants have to interact with them and that they have an impact on each task a human conducts as part of the process and using specific tools. Contextual integration refers to an integration of these tasks, the process, the tools, and the humans with the context to be able to incorporate all necessary and important information in process execution and make the whole project more effective and efficient.

Knowledge. In this chapter it refers to project-, company-, or technology-specific facts and information needed to effectively and efficiently complete the tasks a person must conduct during a project. Examples include information about the applied development process, the concretely used tooling, or specifics of the technology used.

Holistic. In the context of this chapter, the term holistic is to be understood in relation to the entirety of a software project. Holistic project support means support that incorporates different areas of such a project as, e.g., knowledge management, quality management, or software development. Furthermore, holistic support means that it also incorporates the abstract planned process as well as the concretely executed activities.

MAIN FOCUS OF THE CHAPTER

Issues, Controversies, Problems

Software modernization and business process archeology are complex undertakings. Software tools such as those previously mentioned support many of the related tasks. Yet two crucial issues remain: The first is that support provided by these tools focuses primarily on automating certain modernization tasks such as transformation. However, despite advances, in our opinion major software modernization projects cannot yet be fully automated and thus various manual tasks remain to be executed by software engineers like coding, bug fixing, and adjusting tests for the modernization to succeed. While advanced tools are available, contextual support and guidance in congruence with knowledge and process awareness remains unaddressed. The second crucial issue is that tools that support various tasks are typically specialized for these tasks. What is still missing is a holistically supportive system that unites the various aspects of a project with the related tasks, tools, artifacts, knowledge, operative processes, and software engineers. This section identifies different problem areas of these crucial issues and elicits requirements for each area. To illustrate these different problems, a practical example from a software project is applied in the following:

Example 1 (Software Project Problems): *Imagine a company producing software. This company suffers from problems that many companies in this domain share: the do not have real control over the approach and process of their projects, be they software development, modernization, or other maintenance projects. Projects can thus not be sufficiently monitored and controlled, resulting in budget, schedule, or quality deficiencies. To counteract these problems, the company has introduced the usage of process models for all projects to make them more controllable and repeatable. However, as these high-level process models have shown to have relatively little impact on the actual operational activities conducted in the projects, the company has introduced a PAIS to implement and support the process models automatically. Yet the success of this approach was also limited. Parts of the process models could be automatically executed using workflows in the PAIS, but these workflows were still too far removed from the concrete tasks conducted in the projects. The project participants (e.g., software engineers or requirements analysts) reported various issues: The activities in the PAIS were too abstract and had no real relation to the real tasks conducted with their real SE tools. Furthermore, the workflows were too rigid and didn't allow for as hoc deviations that are usual in such a dynamic domain. This also applied to problems resulting in exceptional situations during project execution. Therefore, the prescribed workflows were more and more disregarded by the participants and even viewed as burdensome rather than helpful. The workflows could also cover only a small part of the work done in the projects. Many activities remained unmodeled and unsupported. In addition to this, the participants reported other remaining issues: Often multiple people from multiple departments worked on artifacts that related to each other (like an architecture specification and the source code), but this relation was not properly managed and inconsistencies emerged. The same applied to the knowledge needed for each project. Besides some project specific wikis, there was no management of such information and people often had to spend much time in gathering or developing it. Finally, as the quality of the source code was not continuously managed and quality management was not tightly integrated with the process, the participants complained that they often had to apply bug fixes or software quality measures under high time pressure at the end of a project.*

In the following, concrete problem areas are elicited for these common problems, Therefore, Figure 2 depicts these problem areas abstractly:

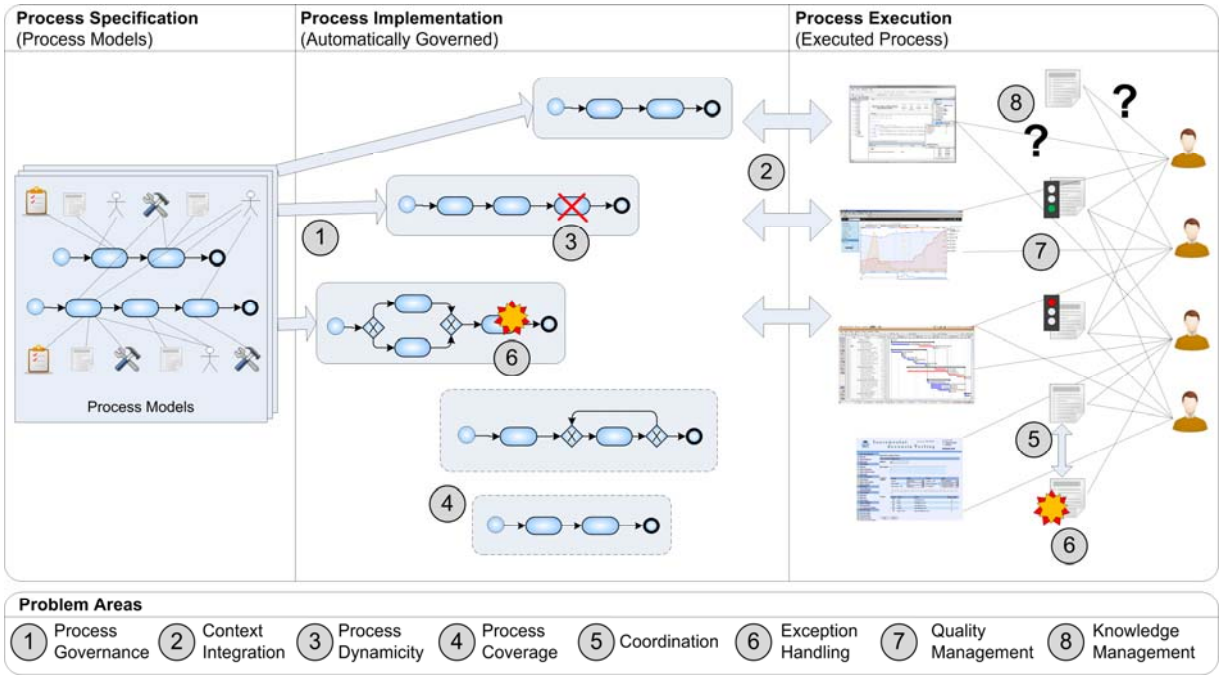

Figure 2. Problem areas in contemporary projects

Automated Process Governance. One problem area is process tracking and guidance, referred to here as automated process governance (cf. Figure 2-1). 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 (Gibson, Goldenson, & Kost, 2006). Many process models have been developed, some specifically for software modernization projects like XIRUP, or other common SE process models such as Scrum (Schwaber & Beedle, 2001), the Unified Process (Jacobson, Booch, & Rumbaugh, 1999), or the V-Model XT (Rausch, Bartelt, Ternité, & Kuhrmann, 2005). The problem with these models is that they typically exist only on paper or web pages and are only used for process specification and documentation. Their impact on actors and concrete activities remains low (Wallmüller, 2007). Automated support for executing such process models is desirable. There are numerous tools capable of automated workflow governance, but the automatically-assisted implementation of a whole process model with such tools remains a challenge. Requirements for automated process support include the following:

- A tool or framework must be in place that governs the process. In order to assist and not interfere, the tool user interface should remain on the sideline, not require extra effort, and not be cumbersome to use. This should be supported by easily accessible user interfaces that are well integrated with everyday work.
- The ability to cover the utilized SE process models like those previously mentioned. These processes have many different special properties that represent the nature of each process. A tool or framework that aims at providing automated process enactment support must thus be capable of modeling these properties and cover the processes to a large extent.
- The automatic distribution of human tasks to the respective persons.
- The correct execution of the modeled workflows must be guaranteed by the system executing them.

- Process implementation requires that an arbitrary number of sub-workflows can be connected to the activities of a workflow, e.g., to model a number of different guided activities by different people that are part of a development iteration of a process.
- A system providing comprehensive support for the SE process must implement the process in a consistent way, including the connection of the abstract process areas (e.g., project, phase, iteration) with the operational workflows that directly concern the executing persons.
- Automated process support should not create cumbersome extra work for the persons involved. Therefore, a tool aiming at comprehensive process support should be capable of easily executing automatisms to support repetitive tasks associated with process execution. These automatisms should, if possible, be easily configurable for the users to avoid 'hard coding' procedures and its associated inflexibility.
- The process of creating software is very dynamic and many factors in SE projects are not easily foreseeable. Therefore, a tool supporting that process should be able to deal with dynamic situations to a reasonable extent.

Context Integration. A second important problem area is 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 executing processes align. The latter relies mainly on different actors using different tools (e.g., requirement management tools or IDEs) to manipulate various artifacts that are crucial for the process. These activities and tool interactions are not directly modeled in the process models since they are too fine-grained. Thus, a dichotomy between the planned and actually executed process may exist. Therefore a tool or system enabling automated process support must be able to deal with such contextual information with the following requirements:

- To have access to contextual information, a tool should be able to gather and integrate information from various sources in its environment. This should be possible in an automated fashion and without disturbing users or other tools.
- The contextual information must be processed and combined to be able to gain viable information from data that is gathered in the SE environment.
- To enable comprehensive process implementation support for SE, a system should be able to integrate the process contextually into the project. Therefore, contextual data should be made available to the system. However, this data must be usable for process execution, and thus the specification of the implementing workflows should support the connection to and integration with contextual data.
- To support the software development process as it is actually executed, while mapping the various interconnected types of human activities that appear both in reality and in the SE process models, a system should support activities with different granularities and properties.

Process Dynamicity. Reality has shown that project execution often does not go exactly as planned. A planned process is a start, but to bring this process together with the real issues occurring in everyday project work remains a more difficult issue. A tool or system seeking to provide holistic process support must thus be capable of coping with dynamic changes to the process (Reichert & Weber, 2012):

- To be able to react to the changing situations in the dynamic software development area, a system should be able to incorporate changes to workflows implementing the process. These changes

should be possible while the workflows are executed, since the need for these changes will not always be known prior to workflow initiation.

- As SE project execution involves many different factors, not all information necessary to adapt a workflow with maximum effectiveness might be available to an individual. Therefore, a system should support automatic adaptations to running workflows dependent on contextual factors (Grambow, 2013; Grambow, Oberhauser, & Reichert, 2011a).

The above three areas deal with general problems and basic requirements to a system that seeks to solve them. All of the three areas are covered by the section 'Approach and Framework to support Software Modernization Projects' that introduces the foundations of our solution. In the following, more specific problem areas are described, which are then dealt with in separate sections.

Extrinsic Process Coverage. One crucial problem with process models is the fact that they cannot cover all workflows that are actually executed in a project. Hence, we distinguish between *intrinsic workflows* (part of the process) and *extrinsic workflows* (unforeseen in the process, cf. Figure 2-4). The latter can be executed based on exceptional situations, but can also be recurring common tasks (e.g., bug fixing or technology evaluation). These activities rely heavily on the properties of the current situation, remain mostly unplanned and untraced, and may impact timely process execution. Concrete requirements as well as our proposed solution are discussed in the section 'Extended Process Support'.

Process Exception Handling. During the execution of a complex project, many unforeseen and exceptional situations occur. This poses a big challenge to any tool 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 (Reichert & Weber, 2012). In reality, process exceptions are often not that simple and also not easily detectable. Further, they may relate to processed artifacts even without the introducing person noticing them. Furthermore, to select an exception handling suitable for both the situation and person is challenging. Concrete requirements as well as our proposed solution are shown in the section 'Exception Handling'.

Collaboration and Coordination. In a complex project, there are always activities and artifacts that relate to each other. That implies that one 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. An example of a relation specifically affecting modernization projects includes architectural specifications and relating source code artifacts. As some of these activities may be covered by the process while others are not, this might result in problematic artifact states if many related adaptations by different people are applied in an uncoordinated manner. Concrete requirements as well as our proposed solution are shown in the section 'Tasks Coordination'.

Quality Assurance. One problem affecting many SE projects is the quality of the software produced (cf. Figure 2-7). Hence, quality assurance is a crucial factor for any SE project. However, in many projects quality assurance is understood as applying some bug fixes at the end of the project when time allows. Studies have shown that this is very ineffective and that quality measures should be applied systematically during project execution (Slaughter, Harter, & Krishnan, 1998). This involves proactive as well as reactive quality measures. The challenge in this area is to effectively and efficiently integrate the application of these quality measures with the development process. Concrete requirements as well as our proposed solution are discussed in the section 'Quality Assurance'.

Knowledge Provisioning. The creation and modification of software is a complex and knowledge-intensive task since software is an intangible asset. It involves knowledge from different sources, all of which are crucial for the success of the task. This includes information on the process, the coding style and other

specifics of the company, the used framework or area (frontend or backend development), etc. Companies often neglect this fact and do not implement proper knowledge management. This often leaves software engineers without all required knowledge and thus makes their work ineffective and error prone. Concrete requirements as well as our proposed solution are shown in the section ‘Knowledge Provisioning’.

Approach and Framework to support Software Modernization Projects

To achieve the connection of abstract and operational processes and holistic project support for software modernization projects (and SE projects in general), we propose the following concept and framework. The concept is centered on a contextual extension of applied process management in conjunction with facilities to automatically gather and process contextual data. The idea is realized within a modular framework called CoSEEEK (Context-aware Software Engineering Environment Event-driven frameworK). In the following, the concept as well as the realization within CoSEEEK will be briefly explained and illustrated (cf. Figure 3). For further reading, see (Grambow, 2013; Grambow, Oberhauser, & Reichert, 2011a; Grambow, Oberhauser, & Reichert, 2012a; Oberhauser, 2010).

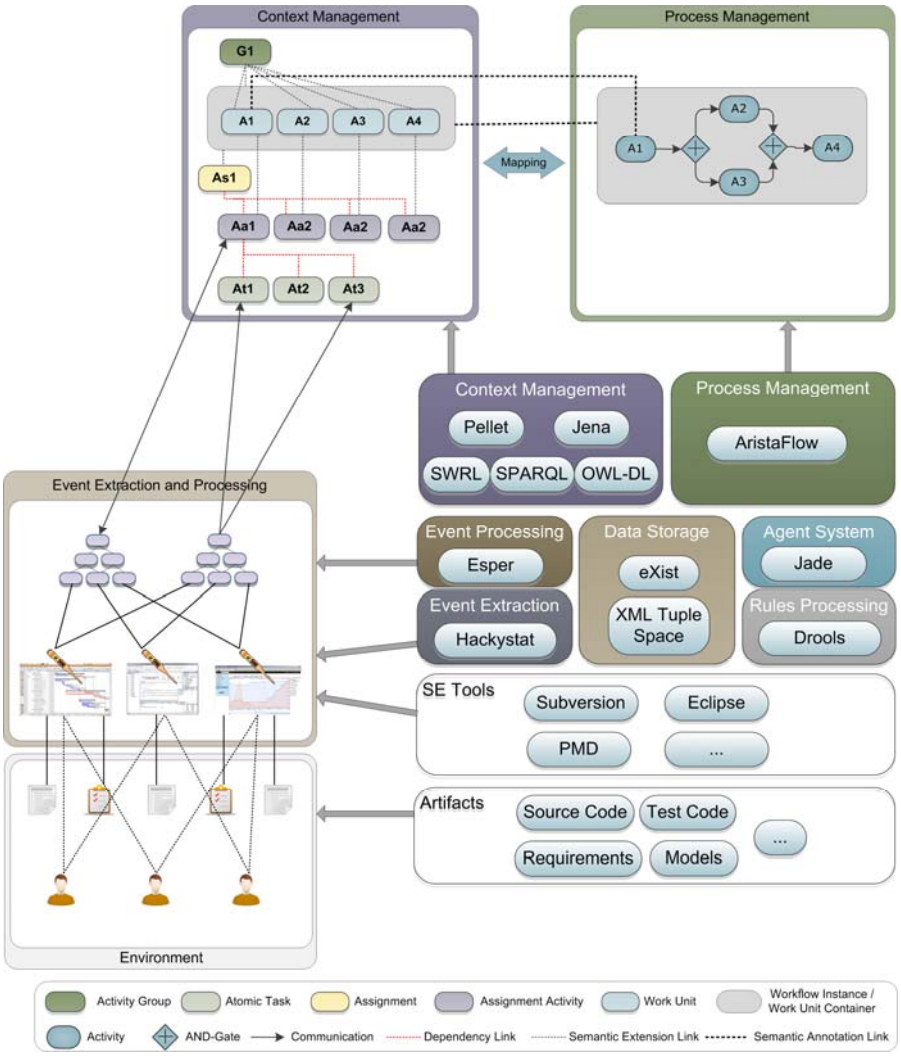

Figure 3. Conceptual architecture and implementation

The concept consists of a set of loosely coupled components targeting different requirements. Communication between the different components is event-based. These events are stored and distributed by a *Data Storage* component organizing the events in different collections for different topics. Each component may register for these collections to automatically receive events of other components relating to that topic. That type of communication is realized by an implementation of the tuple space paradigm (Gelernter, 1985). Event communication is realized with XML events with the tuple space utilizing the native XML database eXist (Meier, 2009).

Basic process implementation and automation is realized via the *Process Management* component. The latter is in charge of managing and executing workflows that are part of a process. As this component is implemented utilizing the AristaFlow BPM Suite (Dadam & Reichert, 2009; Lanz, Reichert, & Dadam, 2010), it is capable of addressing many basic requirements. AristaFlow provides process management technology that is notable with respect to the flexible support of adaptive and dynamic workflows. New workflow templates can be composed out of existing application services in a plug-&-play like fashion, and then serve as schema for the robust and flexible execution of related workflow instances. In particular, during run-time, selected workflow instances can be dynamically and individually adapted in a correct and secure way; e.g., to deal with exceptional situations or evolving business needs (Reichert, Rinderle-Ma, & Dadam, 2009). Examples of workflow instance changes supported by AristaFlow include the dynamic insertion, deletion, or movement of single workflow activities or entire workflow fragments respectively (for a discussion of the adaptation patterns supported by AristaFlow, see (Weber, Reichert, & Rinderle-Ma, 2008)). For integrating these change functions and other AristaFlow services (e.g., for managing user work lists or for defining workflow templates) with a domain- or application-specific PAIS as in our case, the AristaFlow Open Application Program Interface (Open API) can be utilized (Reichert et al., 2009). For example, for dynamically inserting activities at the workflow instance level, the application developer can make use of the following system functions provided by the Open API:

- Querying the activity repository for available activity templates,
- Marking those activities of the workflow instance after which the selected activity shall be inserted (i.e., after completing these activities the newly added one shall be enabled),
- Retrieving the set of activities selectable as “end” activities for this insertion,
- Marking the activity (or set of activities) which shall serve as end activity (activities),
- Performing (tentatively) the insertion based on this information,
- Checking the AristaFlow report on detected errors (e.g., missing values for input parameters), and
- Making the instance change persistent.

Note that dynamic workflow instance changes can be conducted at a high level of abstraction. In particular, all complexity relating to dynamic workflow instance changes (e.g., correct transformations of the workflow schema, correct mapping of activity parameters, state adaptations) are hidden to a large degree from end users and application developers respectively (Reichert & Dadam, 1998). Further, AristaFlow provides techniques for learning from past experiences and workflow instance adaptations, respectively, and for evolving workflow schemes accordingly (Weber, Reichert, Wild, & Rinderle-Ma, 2009).

The second core component of the concept is the *Context Management* component. The latter is the central information management and coordination component of the concept. It realizes the integration of process execution with contextual data acquired from the project environment. To support a high degree of automated and context-aware assistance, a tight coupling of the *Context Management* and the *Process*

Management component is required, which we refer to as *Context-aware Process Management* (CPM). Fundamentally, process management concepts are enhanced with semantic information. This additional information is stored in the *Context Management* component, while the workflows are managed by the *Process Management* module. Since *Context Management* unifies all project knowledge, it can also be used as a management layer around the *Process Management* component, facilitating context-based process management. Thus, all process-related actions are addressed by the *Context Management* component, which, in turn, manages the actions of the *Process Management* module. Figure 3 illustrates these extensions to process management. The *Process Management* component governs the workflows and their activities. These two concepts are mirrored in the *Context Management* component: the activity by the *Work Unit* and the workflow by the *Work Unit Container*. Thus, process management is separated into two areas. On the one hand, the governing of the different activities of one workflow (also denoted as process orchestration) utilizing well-established workflow patterns like AND, SPLIT, or LOOP. This is done within the *Process Management* module. The Context Management component, in turn, is in charge of extending process management concepts with various other concepts supporting users and project execution. One example is activity-related concepts: *Work Units* are connected to three other concepts, enabling advanced task management. The *Assignment* is used as a coarse-grained top-level task, which is also estimated and scheduled from the business side in a project. The *Assignment Activity* then describes the tasks that are necessary to accomplish the *Assignment*. The most fine-grained level is described by *Atomic Tasks* that denote low-level tasks a person conducts, e.g., using a SE tool. Various tasks can be automatically detected by the *Event Extraction* component, which is described later in this section. Combining the *Context Management* and the *Process Management* modules enables the automatic adaptation of running workflows based on the current context. The application of this capability will be shown in the section explaining quality assurance integration. Since total automated process governance is not viable for all situations, a user-centric abstraction of workflow logic provides users a capability to provide necessary inputs that influence the workflow (Grambow, 2013; Grambow, Oberhauser, & Reichert, 2012d).

The *Context Management* module employs semantic technology to enable high-level utilization of all project knowledge. This technology has several advantages, including enhanced interoperability between different applications, extending reuse possibilities, and the option for advanced content consistency checking (Gasevic, Djuric, & Devedzic, 2006). It also provides a vocabulary for the modeled entities including taxonomies and logical statements about the entities. Ontologies also provide the capability of reasoning about the contained data and inferring new facts. As an ontology language, OWL-DL (Web Ontology Language Description Logic) is used due to its proliferation and standardization (World Wide Web Consortium, 2004b). For simple RDF-based (World Wide Web Consortium, 2004a) queries to the ontology, SPARQL (Prud'hommeaux & Seaborne, 2006) is used. Operations that are more complex are executed using the reasoner Pellet (Sirin, Parsia, Grau, Kalyanpur, & Katz, 2007) that also executes SWRL (Horrocks et al., 2004) based rules. Programmatic access via DAO objects to the ontology is provided by the Jena framework (McBride, 2002). Thus, different semantic concepts can be created and manipulated as needed.

A crucial part of the utilized context knowledge comes from the *Event Extraction* and *Event Processing* components. The *Event Extraction* component utilizes the Hackystat framework (Johnson, 2007), which provides a rich set of sensors that can be integrated into various SE tools. The sensors enable the *Event Extraction* component to automatically generate events in different situations, as, e.g., checking in a source code file in Subversion or switching to the debug perspective in Eclipse. The events generated and collected in the *Event Extraction* component are basic and low-level. The *Event Processing* component, in turn, utilizes *complex event processing* (CEP) (Luckham, 2001) to process these events, providing high-level events with enriched semantic value. This is done utilizing the framework Esper[v]. The latter provides a facility to define patterns that govern how certain events are combined to derive other higher-level events, which are then again written to the *Data Storage* component as all other events.

The final two components are the *Rules Processing* and the *Agent System* components. The former enables the automatic execution of simple automatisms and is based on the framework JBoss Drools[vi]. The latter has been applied for enabling automated decisions for dynamic situations during project execution. It employs a multi-agent system (MAS) with different behavior agents. It is implemented utilizing the FIPA- compliant (O'Brien & Nicol, 1998) JADE framework (Bellifemine, Poggi, & Rimassa, 1999). The usage of both of these components will be illustrated in detail in the section discussing automated quality assurance integration.

Extended Process Support

This section deals with small workflows not covered by process models. These workflows are often dynamic and are highly dependent on the context of a particular situation. We call them extrinsic workflows. This is further described in (Grambow, 2013; Grambow, Oberhauser, & Reichert, 2010b; Grambow, Oberhauser, & Reichert, 2011c; Grambow, Oberhauser, & Reichert, 2012a). As mentioned in the problem statement, such workflows can have negative effects on project and process execution, since they are dynamically executed during project execution without prior planning and without traceability. Furthermore, they are difficult to model with the capabilities of imperative workflow technology. A system that aims for holistic process support should include a means for consistently integrating such workflows:

- There should be a facility to support both *intrinsic* and *extrinsic activities* by an automated system or framework.
- Both *intrinsic* and *extrinsic activities* should be executed in a uniform way to support uniform assistance for the user and to enable easy tracking and analysis of executed workflows.
- Compared to intrinsic workflows, extrinsic workflows are more dynamic and less foreseeable. Their modeling should enable coverage of various possible situations without bloating process models or making them too complex.
- The workflow modeling itself should remain easy and foster the reuse of modeled solutions or the parts thereof.
- The workflow modeling should hide the inherent complexity of the workflow models to assist the user with problem-oriented creation of the models.
- There should be facilities to automatically gather information on the current situation from users or the development environment.
- The modeling environment should be capable of modeling contextual influences to be able to use situational information directly.
- A facility to model the connections of contextual properties to workflow activities is required to enable their automated situational selection.

To illustrate the SE requirements, we apply the following example:

Example 2a (Extrinsic Workflows): *As aforementioned, SE issues arise that are not modeled in the standard process flow of defined SE processes. This includes bug fixing, refactoring, technology versioning, or infrastructural issues. Since there are so many different kinds of issues with ambiguous and subjective delineation, 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 4a*

shows such a workflow for bug fixing that contains nearly 30 activities (i.e., steps), many of these being conditionally executed for accomplishing different tasks like testing or documentation. One example is static analysis activities that are eventually omitted for very urgent use cases. Furthermore, there are various reviewing activities with different parameters (such as 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 if a bug fix should be merged into various other version control branches.

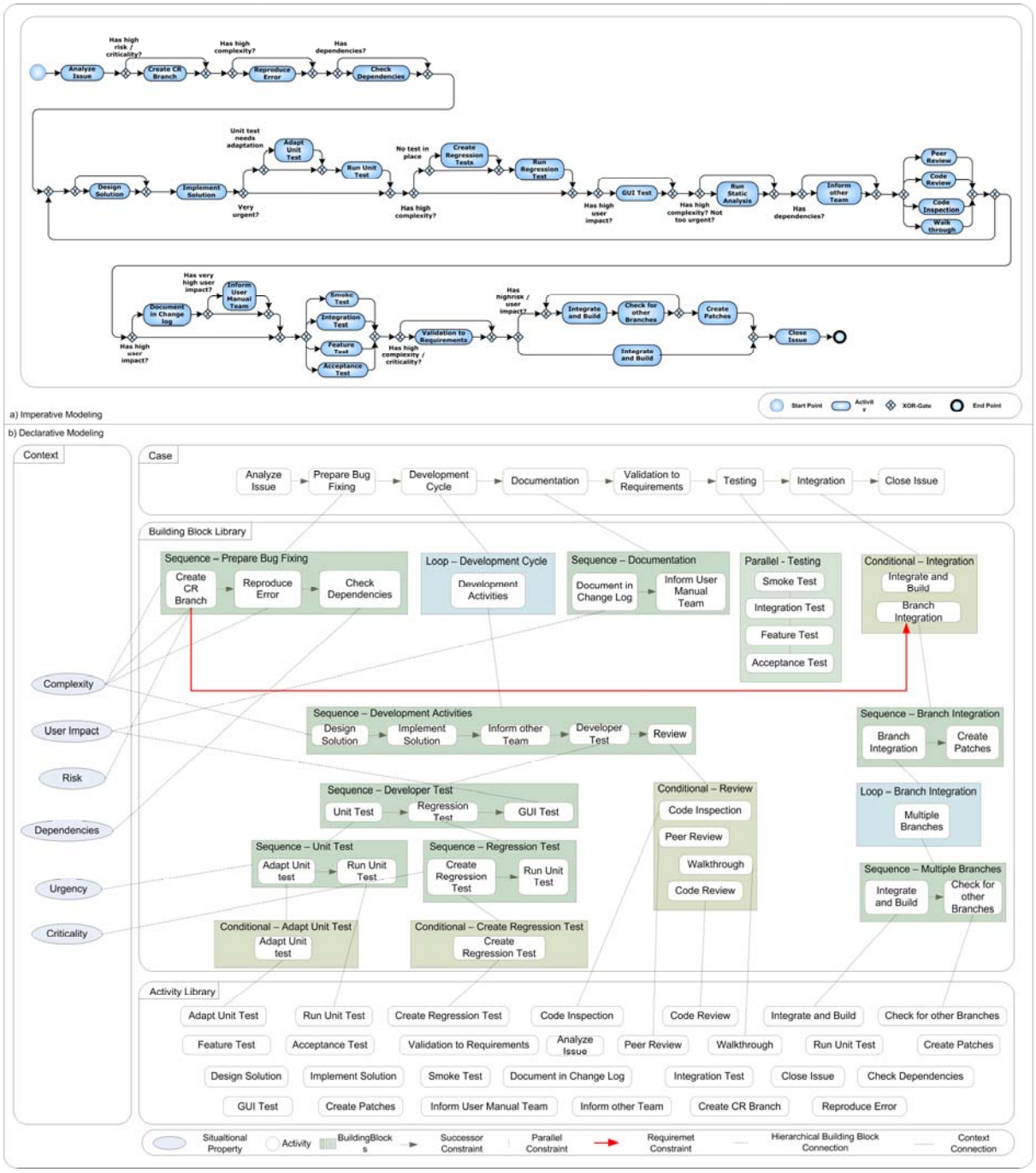

Figure 4. Imperative modeling (a) vs. declarative modeling (b)

This solution incorporates a set of sensors that enable the automatic gathering of contextual information; e.g., state transitions of certain SE tools or SE artifacts that can be recognized as properties of situations. In this section, facilities to model contextual properties that can be used to describe a situation are shown, e.g., ‘Risk’ or ‘Complexity’. In turn, these properties have calculated values that may be derived from various sources as the skill level of a user executing an activity or the measured code complexity of a processed source code artifact. To be able to contextually integrate process execution into the projects and thus enable the process to be influenced by the properties of various situations, explicit connections of process management concepts to context properties are introduced.

As concrete workflow execution is often relatively dynamic in SE, a rigid pre-planning of activity sequences is not always advantageous. Therefore, this section shows a means of declaratively modeling candidate activities for a workflow at build-time that enables a system to automatically select appropriate activities for various situations at run-time. The modeling is designed to be hierarchical, separating workflow models into several nestable blocks. These blocks can be modularized and be logically treated as simple activities, fostering their reuse in multiple workflow models and simplifying these. To support process engineers in modeling declarative context-dependent workflows, an easy way of specifying context properties, workflows, contained blocks, and activities is provided.

Utilizing this modeling method, extrinsic workflows can be addressed. To unite this with traditional imperative process modeling, which is still useful for more predictable processes (Reichert & Weber, 2012), our approach unites both ways of modeling under a common process management concept. In the following, different parts of the concept will be illustrated: modeling of contextual influences, gathering of contextual information, and declaratively modeling processes.

Contextual Properties. To integrate situational influences, Situational Method Engineering (SME) is applied, which adapts generic methods to the actual situation of a project (Ralyté, Brinkkemper, & Henderson-Sellers, 2007). We used two different influence factors: process properties, which capture the impact of the current situation; and product properties, which realize the impact of the product currently being processed (in this context the type of component, e.g., a GUI or database component). To strike a balance between rigidly pre-specified workflows and the absence of process guidance, the idea is to have a basic workflow for each use case, which is then dynamically extended with activities matching the current situation. The construction of the workflows utilizes a so-called case base as well as a method repository. The case base contains a workflow skeleton of each of the use cases. In the following, these use cases, which are associated to an SE issue and have an attributed workflow, will simply be called cases. The workflow skeleton belonging to a case only contains the fundamental activities always being executed for that case. The method repository contains all other activities whose execution 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 SME.

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 is connected to the activity via an n-to-m relation. The activities are connected, in turn, to properties. The latter are concepts used to explicitly model contextual properties of the current situation and case (e.g., Complexity=high, Urgency=low). The selection of an activity can 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 also connected to various properties. The properties 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.

Information Gathering. To leverage the automatic support for extrinsic workflows, the computation of the property values constitutes a key factor. Our approach unifies process and product properties in the concept of the *property*, which can be influenced by various factors. On the 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 (such as PMD[vii]) can be used to execute various measurements on that file, revealing various potential problems. This aspect deals with implicit information gathering. Since not all aspects of a case are necessarily covered by implicit information, and not all options for gaining knowledge about the case are always present, the system utilizes explicit information gathering from the user processing the case. To enable and encourage the user to provide meaningful information, a simple response mechanism is integrated into the user interface. Via this mechanism, the user can directly influence process as well as product properties. To enable the system to utilize explicitly gathered information for workflow generation, the workflow skeletons of the cases always start with an activity 'Analyze Issue'. The latter lets the user gain awareness about the issue and the current situation and set the properties accordingly. To keep the number of adjustable parameters small, the concept of a product category was introduced. The product category unites the product properties in a pre-specified way. The influence of the product categories on the different properties is specified in advance and can be adapted to fit various projects.

Declarative Workflow Modeling. After completing the computation of the property values, activities must be selected and correctly sequenced to enable dynamic construction of the workflow for an SE issue. This is done utilizing the connection between properties and activities. An activity can depend on one or more properties. Examples include selection rules such as:

- 'Choose activity code inspection if risk is very high, criticality is high, and urgency is low' or
- 'Choose activity code review if risk and criticality are both high'.

Declarative workflow modeling approaches incorporate a certain amount of flexibility in the workflow models (Pichler et al., 2011) and thus enable the latter to be applicable for different situations. However, the declarative way of modeling can be difficult to understand (Zugal, Pinggera, & Weber, 2011a) and can produce models that are hard to maintain (Zugal, Pinggera, & Weber, 2011b). Therefore, this declarative workflow modeling approach is based on very simple constraints and so-called Building Blocks that enable further structuring of the workflow and structural nesting.

This modeling type is illustrated and compared to classical workflow modeling in Figure 5. The figure shows the modeling of the Work Unit Containers above and the derived workflows for execution below. 'Work Unit Container 1' shows a simple, imperatively modeled workflow that is also executed in that form (as 'Workflow 1'). 'Work Unit Container 2' illustrates declarative modeling of the same workflow: the exact structure of the workflow is not rigidly pre-specified. There are only simple constraints connecting activities in the workflow. Examples in Figure 5 are 'Requires', expressing that one activity requires the presence of another, and 'Parallel', expressing that both activities should be executed in parallel. The generated workflow for these constraints looks exactly like the imperatively modeled 'Work Unit Container 1'. Activities in the declarative approach also have relations to contextual properties in order to enable the system to select a subset of the pre-specified activities for the execution workflow. Finally, 'Work Unit Container 3' demonstrates the use of Building Blocks. These are used for further structuring the workflow to enable complex workflow structures. Three Building Blocks are shown for sequential, parallel, and repeated execution of the contained elements in Figure 5. 'Workflow 3' shows how a workflow 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 system to the selection of activities '1', '2', '3', and '5' while omitting activities '4' and '6'. Building Blocks enable

hierarchical structuring of activities contained in workflows and can be reused in different Work Unit Containers easily, where they can be treated like simple activities hiding the complexity of the contained activity structure. That way, simple basic modeling is enabled while retaining the ability to model complex structures. One example for this can be activities related to software creation like coding, testing, or documenting. These can be structured by the Building Blocks, e.g., in a Loop enabling multiple iterations of coding, documenting, and testing the new code combined in one Building Block. The latter can, e.g., be called 'Software Development Loop' and then easily be reused as a single activity. This, in conjunction with the simple basic constraints, supports simple and understandable workflow models. Finally, the advantages of imperative and declarative modeling approaches are combined: The imperative workflows generated for execution ensure that users follow the predefined procedures and also aid the users with workflow guidance. However, by declaratively specifying various candidate activities for these workflows and connecting them to situational properties, the system retains the ability to choose the right activities for the users' concrete situation. More information of the concrete concepts involved is shown in (Grambow, 2013; Grambow, Oberhauser, & Reichert, 2010b; Grambow, Oberhauser, & Reichert, 2011c; Grambow, Oberhauser, & Reichert, 2012a).

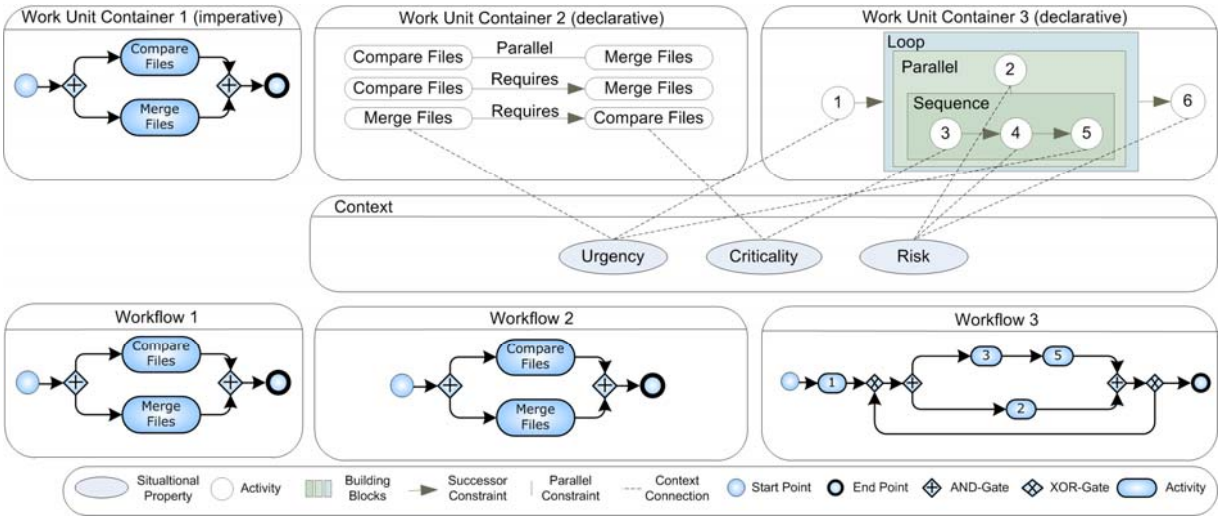

Figure 5. Imperative vs. declarative modeling

The basic sequencing constraints enable easy modeling of simple activity structures that can be enhanced with more complexity utilizing the different Building Block Template types. However, these constraints enable the modeling of structures that endanger the ability to convert them to block structured workflows. Furthermore, as the declarative activity specification only contains candidate activities, several of the latter could be omitted for an executable workflow. This could also create structures that are not convertible to a block structured workflow. Therefore, the so-called auto completion feature is introduced that enables the automatic extension via additional connections between the activities. This is explained and illustrated in the following example.

Example 2b (Extrinsic Workflows): *Figure 4b shows how a bug fixing workflow can be simply yet dynamically modeled using the shown declarative approach: The workflow shown at the top ('Case') contains various Building Blocks out of a Building Block library. One example is the Development Cycle Building Block: It contains various other activities like Implement Solution, Developer Test or Review activities that are executed in a Loop. It is a standard set of activities that can be reused in various*

workflows. To match different situations, the Building Blocks are connected to situational properties, enabling the system to automatically tailor them to various situations. One example is the Review activities that are part of the Development Cycle: According to the situation, a Code Review activity could be chosen, or a Code Inspection or even none at all. All this complexity is hidden from the user and managed by the system, leaving a simple sequential but tailored workflow for the bug fixing issue.

As with most SE projects, software modernization projects suffer from a high complexity and large number of different tasks. Often the latter are executed without awareness of the process and might thus negatively impact it The approach shown in this section satisfies three categories of requirements: Generally, it enables the unification of intrinsic and extrinsic workflows, it supports and simplifies extrinsic workflow modeling, and it offers automation and execution support for these in alignment with properties of various situations. The first of the three is enabled by not only being able to model and execute intrinsic as well as extrinsic workflows, but also by uniformly realizing them as they are executed. The higher level of dynamicity that is inherent to extrinsic workflows is accommodated by a declarative, problem-oriented method of modeling. The hierarchical structure of the declarative modeling approach featuring the concept of the Building Blocks supports the modeling in many ways: complexity is hidden at build-time as well as at run-time. Reuse is fostered since 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 of these: The properties of various situations are automatically detected by the system and are used to select a subset of activities that match that situation. This is enabled by the explicit modeling of these situational properties and their influences on the workflow models.

Task Coordination

SE projects feature a very collaborative process where the communication of various individuals is crucial for project success. In multi-project and multi-team environments, this can be a challenge as there are many diverse information channels and many dependencies. Much of the coordination effort that results from this is manually coped with by the individuals working on the projects. This can result in additional effort for each individual, in confusion, or in missed coordination activities. Thus, automated coordination and collaboration support is desirable. Many different activities by different individuals affect different artifacts. Some of these activities may be included in the process and some not. Some activities may be tool-supported and others may not. This can lead to situations in modernization projects where activities and artifacts are no longer synchronized, such as an architecture specification and the relating source code. As already stated, a system aiming for comprehensive process support should be able to counteract such situations (Grambow, 2013; Grambow, Oberhauser, & Reichert, 2011d; Grambow, Oberhauser, & Reichert, 2012b). To achieve this, an automatic system or tool should satisfy a number of requirements:

- The system shall enable automatic notifications about activities and activity effects. These notifications shall be automatically distributed to the concerned users and be presented to them seamlessly so as not to distract them from their present activity. It shall be flexibly configurable what causes such notifications for whom.
- The system shall be able to automatically determine possible effects of activity execution. This includes finding out which project areas and maybe which artifacts in a SE project are affected indirectly by these activities.
- The system shall be able to automatically determine responsible persons for follow up activities concerning changes needed as result of indirect activity outcomes.

- The system shall be capable of automatically initiating follow up activities for indirect activity outcomes. These follow-up activities shall be automatically distributed to the responsible persons.
- The system shall enable configuration of the way follow-up activities are provided. Users shall not be overwhelmed by numerous micro-activities. Activities of the same type shall be groupable. Thresholds shall be in place for the number of follow-up activities per user to enable collection and consolidated provisioning of the activities.

The following example illustrates the problem and requirements:

Example 3 (Activity coordination): *This example deals with a source code artifact that is part of an interface component: since the file belongs to an interface component, the applied changes possibly not only affect the unit tests of the file, but also other artifacts such as the architecture specification or integration tests. These additional activities are usually neither covered by the SE process nor governed by workflows; 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.*

Support for two different ways of coordinating tasks is desirable, passive and active. Both are elaborated in the following.

Passive Coordination Support. The passive coordination ability provided by the system deals with configurable user notifications. To support users in their collaboration and to counteract forgetfulness, automatic notifications can be beneficial for two situations in SE projects: first, when events happen that relate to activities or artifacts, and second, when status changes occur to the artifacts. For these two cases, different components of our approach are utilized: On the one hand, the *Context Management* component contains information relevant to the process and the project (such as users, roles, activities, artifacts). On the other hand, the *Event Extraction* component automatically forwards events relating to the aforementioned concepts. Examples include the start and end of an activity or a change of the status of an artifact. Based on these facts and an explicit *Notification Template* concept, notifications both for general and individual cases can be easily configured. The *Notification Template* has three main properties: ‘source’ can be connected to arbitrary concepts that shall be monitored by that *Notification Template*. This can relate, for example, to an individual artifact or even to all artifacts of a specific type. ‘trigger’ captures an event relating to the source or a particular state of the source, so that a concrete notification can be created when a particular event occurs relating to the source or when it enters a particular state. The third property ‘target’ describes who shall receive the notification. It can be configured for roles to create more general notification but also for individual persons. The process for such notification works as follows:

- *Notification registration:* The system has general pre-configured *Notification Templates* to which the user can add new personalized ones. This is done by the user creating a new *Notification Template*.
- *Event detection*: As project execution progresses, various events happen and various entities change their state. Some of these events and changes can be relevant for notifications: some of

them are explicitly triggered by users interacting with the system and others are detectable by the *Event Management* component sensors.

- *Notification generation*: The *Context Management* component uses the information in the *Notification Template* concepts to create concrete notifications for the users and assigns these to them.
- *Notification distribution*: The information is displayed to the users via an integrated GUI component.

Active Coordination Support. To support automated active coordination support, a system must be capable of automatically identifying different areas of interest in a project, such as 'Implementation' or 'Architecture'. Therefore, an explicit *Area* concept has been added that, in turn, can be further segregated into so-called *Section*s that enable logical partitioning of an *Area* (e.g., Area: Implementation – Section 1: GUI Development – Section 1.1: Source code package xy). These definitions can be tailored for projects and automatically supported. For example, to split up the 'Implementation' *Area*, the structures of the source code can be scanned, creating sub-*Section*s of the *Area* alongside the package structure of the source code. Automated active coordination comprises a four-phased approach:

1. **Determine projects *Area*s that are affected by an activity**: The first step is configurable and can take into account various facts to determine which *Area*s are affected. For the scenario shown in Example 3, such a configuration can be '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 *Area*s and the target of the applied activity as input. This target can be a concrete artifact as in the given scenario or a more abstract *Section* of the project as, e.g., a module. The concrete target is then determined via relations of the different *Section*s. An example for this can be implementation and testing: the testing (structural or retesting) of a module relates to its implementation. In the given example, the relation does not need to be in place for the concretely processed component, but can also be found if one exists elsewhere in the hierarchy. If there is no direct relation from the processed source code artifact, the system looks for other components the file belongs to, e.g., the module.

3. **Determine the *Person* responsible for the chosen target**: Once the target for the follow-up action is determined, the responsible persons or teams must be discovered. For example, if the target of the follow-up action is a source code file with no direct responsible party defined, the overlying *Section*s are also taken into account, e.g., the encapsulating module. 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 the target and the responsible have been determined, a concrete activity has to be chosen. This is done using properties of the involved artifacts, areas, sections and the activity that was the trigger.

To enable a system to realize automated detection of follow-up actions, different concepts have to be in place in 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 of different relating components must be established as, e.g., the fact that testing a module relies on implementing that module.

(3) Information that may be used to clarify under which circumstances one area affects another must exist.

(4) Different components must be classified, as, e.g., a package in the source code that realizes the interface of a component.

The high degree of dynamic collaboration in SE raises challenges for automated support of process awareness and guidance in SEEs. Currently, SEEs lack contextual information and integration, especially with regard to adaptive collaboration and workflows. The approach presented in this section enables explicit modeling and management of intrinsic as well as extrinsic activities. Active and passive collaboration support is provided for software development. This incorporates two different types of support: Communication and collaboration of different individuals is supported by passive information distribution that is automatically conducted by the system. Configurable mechanisms are in place to enable a wide scope of notifications concerning various events or state changes of various entities or activities. Furthermore, notifications can be general, pre-configured, user-specific, and/or dynamically added by concerned individuals. Additionally, the system enables the fully automatic initiation and governance of related follow-up activities caused by other activities. A dynamic information distribution strategy enables related components to be associated even if no direct relations between the source component and the target component exist.

Exception Handling

To increase the level of standardization (i.e., usage, repeatability, conformance, etc.) of process execution, automated support for SE processes is desirable. To enable this in a holistic way, an automated solution should be capable of some kind of process exception handling so that the occurrence of exceptions does not deteriorate process performance. This is further described in (Grambow, 2013; Grambow, Oberhauser, & Reichert, 2011b). Automated process exception support will only be acceptable if it is not too complex or more cumbersome than manual handling (Ellis, Keddara, & Rozenberg, 1995). Automated handling implies automated detection of exceptions that depends on the capabilities of the system managing the processes (Luo, Sheth, Kochut, & Miller, 2000). However, existing PAIS are still rather limited regarding detection and handling of exceptions (Russel, van der Aalst, & ter Hofstede, 2006). Exceptions can arise for reasons such as constraint violations, deadline expiration, activity failures, or discrepancies between the real world and the modeled process (Russel, Ter Hofstede, Edmond, & van der Aalst, 2004). Especially in the highly dynamic SE process domain, exceptions can arise from various sources, and it can be difficult to distinguish between anticipated and unanticipated exceptions. Even if they are detected, it can be difficult to directly correlate them to a simple exception handler. Exceptions can arise relating to various items such as activities, artifacts, or the process itself. To incorporate automated support for such exceptions into a system, the following requirements should be satisfied at least to some extent:

- The system shall enable automatic detection of the occurrence of various exceptions. This comprises the ability to automatically infer the occurrence of an exception based on various events acquired from the environment of the system.
- The system shall be able to determine situationally appropriate exception handlings. On the one hand, this depends on the correct classification of the exception. On the other hand, it depends on contextual factors the system must incorporate like properties of the current project or situation.
- The system shall be able to automatically determine the person responsible for an exception handling. This is no trivial task since according to the situation it could be a person responsible for an activity, or for an artifact, or perhaps the principal of a person.
- The system shall be able to automatically initiate and govern exception handlings. As all parameters of the exception and the planned handling are determined, the system must have access to the process as it is enacted to automatically initiate a handling, distribute it to the responsible person, and govern its execution.
- The system shall be able to deal with incomplete knowledge about exceptions. In many situations, not all needed data relating to an occurred exception may be in place. The system shall be capable of taking action in these situations as well, utilizing the incomplete knowledge.

Again, to illustrate the requirements, we apply a concrete example:

Example 4a (Exception): *Exceptions can occur that relate directly to executed activities, to artifacts, or to future scheduled activities. 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. E.g., source code complexity might increase if multiple people applied "quick and dirty" fixes. Thus, the understandability and maintainability of that file might drop dramatically and raise the probability of further defects.*

Many of these exceptions may be difficult to detect, especially for a PAIS without direct knowledge of the environment. It may also be unclear when exactly to handle the exception and who should be responsible. Generally, the knowledge about the exception can vary greatly, making unified handling difficult and the application of standardized exception handlers unsuitable. To enable a system to automatically deal with such exceptions and satisfy the requirements already elicited regarding exception detection, exception handling, and the distribution of the handling to a matching human agent, our concept relies chiefly on contextual information, its modeling in the system, and its detection by sensors. To apply a unified and repeatable approach to automated exception handling, we apply a set of concepts and a well-defined procedure. The latter can be roughly understood as an extended flexible variant of ECA (Event-Condition-Action) (Paton, 1999). The three phases are called Recognition, Processing, and Action here. In the following, the involved abstract concepts for exception handling are elaborated.

Exception: The notion of *Exception* is utilized to classify a deviation from the planned procedure that was recognized to have a potential negative impact on the process, and thus should be dealt with to avoid such an impact. According to (Reichert & Weber, 2012), typically there is a distinction between anticipated exceptions, whose occurrence can be easily foreseen, and unanticipated ones. For anticipated exceptions, standard exception handlers can be defined. Usually, this is not possible for unanticipated ones. Since SE projects typically feature a dynamic process and it might be difficult so foresee a multitude of possible exceptions, our approach does not discriminate between anticipated and unanticipated exceptions. It also does not use standard exception handlers tied to specific exceptions. Flexibility is improved through the

explicit separation of events, exceptions, handling of the exceptions, responsible persons, and the point in the process where a handling is invoked. Thus, 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 that. Additionally, the approach manages different levels of knowledge about occurring events. Depending on that level of event knowledge, it can be decided whether a more generic exception shall be raised or rather a specialized one. As stated in (Russel, Ter Hofstede, Edmond, & van der Aalst, 2004), anticipated exceptions occurring during the execution 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. These can be covered by various exception types like *Activity-related Exception*, *Artifact-related Exception*, and *Process-related Exception* that can be explicitly modeled. The included exception hierarchy is not intended to cover every possible exception in every project. It rather presents a basis for frequent exceptions and can be extended:

Handling: The notion of *Handling* is used to describe activities executed as countermeasures for a triggered exception. Since SE exceptions are usually complex and of semantic nature, no simple rollback of the activities that caused the exception can be done. As an example, consider the activity of bug fixing (as shown in Example 4a): While fixing a bug, this activity can also introduce additional problems to the code such as increased code complexity. This can happen when the person applying the bug fix is not the one responsible for the processed artifact. As a countermeasure, an explicit refactoring can become necessary. *Handling* neither comprises the person to execute these activities nor the time or point in the process where they are to be executed.

Responsible: *Responsible* captures the responsible person for a *Handling*. As in Example 4, this can be the one who executed an activity introducing the exception or the one responsible for an artifact related to an exception.

Target: *Target* is the point in the process where the *Handling* is executed. For certain exceptions, it can be suitable to integrate *Handling* directly into the workflow where the exception occurred, whereas in other cases a separate exception handling workflow has to be executed.

Exception Handling Procedure. The procedure enabling automated exception handling is separated into three phases named recognition, processing, and action phase. Each of these comprises several activities conducted by the system. The procedure is illustrated in the following, beginning with the activities of the recognition phase.

- *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 also to supporting tools. Therefore, the Event Management component gathers a multitude of events from various tools like IDEs or source control management tools.
- *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 file) and provide no information about the complex activity a person is performing. Therefore, these atomic events need to be processed and aggregated to derive higher-level events of more semantic value (like the application of a bug fix).

The processing phase comprises a set of activities, differentiating between immediate and deferred exception handling cases:

- *Event Classification*: Event classification can be used to gain more knowledge about events or other new information to be able to find a specific handling later. The new information can also be related to the current project and its properties, like e.g., its quality goals or properties of the current situation.
- *Handling Determination*: When an exception has occurred, it has to be decided when and how to take countermeasures, which also depends on the current project situation. The situation can be classified using different parameters like risk or urgency. If urgency is high, meaning there is a high schedule pressure on the project, one might 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 might also be decided to retain an exception if the knowledge about it does not suffice for immediate automatically-supported handling. Furthermore, different types of exceptions with different handlings can be connected to different events relating to different levels of knowledge. Thus, relative generalized exception handlings are also possible for situations in which only a small amount of knowledge about the exception is present.
- *Responsible Determination*: If it is decided to take immediate action in case of an exception, the person responsible for that action has to be determined. There can be different possibilities: For example, if an exception relating to an activity occurred, the processor of that activity can be responsible or, if an exception occurred relating to an artifact, the responsible person for that artifact (or, e.g., source code package) can also be responsible for handling the exception. There may not be a direct party responsible for each processed artifact, but responsibilities can be hierarchically structured to simplify determination of the responsible party.
- *Target Determination*: When the responsible party for handling the exception is determined, the concrete point in the process has to be determined where the handling is applied. In certain situations it may be appropriate to directly integrate a handling in a running workflow. In other cases a new workflow would be initiated.
- *Exception Retainment*: If, due to various parameters of the situation, no immediate handling is favored, 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 final phase is called action phase and comprises the following activities:

- *Handling Preparation*: After all parameters for the handling of an exception are determined, the concrete handling has to be prepared, i.e., a new workflow instance has to be created or the handling has to be integrated seamlessly into a running workflow instance.
- *Handling Execution*: Finally, the prescribed handling is executed by the chosen person.
- *Deferred Handling*: When exceptions are retained, a human can decide for which exceptions a deferred handling is preferred. Therefore, an additional GUI is needed, presenting a list of retained exceptions and enabling manual determination of a handling or discarding of the exception.

To illustrate the practical application of this approach to exception handling, Example 4a is reconsidered in the following example:

Example 4b (Exception handling): *As aforementioned, consider the following scenario: the code complexity of a source code artifact is very high and was introduced by some activity. The problem may*

be detected much later and relate more to the artifact than to the activity in that case. Furthermore, the appropriate person to deal with the problem could be the one responsible for the entire artifact rather than the last person who worked on it. To counteract the new problem, an explicit refactoring can become necessary. Automatic detection of the deterioration of the state of this artifact is possible in the approach via static analysis tool reports, which can cause a process exception to be raised. A context-matching handling can then be automatically chosen (the appropriate refactoring). If the person is still working on that artifact, the handling can be directly integrated with his work. If not, the handling is forwarded to the person responsible for the artifact and can be queued as a work item.

The approach explained in this section enables a higher level of automatism and support relating to exception handling. It 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 the semantic capabilities also enables the discovery of links between activities and exceptions that have no direct connection. These features also support the determination of a situationally matching handling for an exception. Finally, the flexibility of the handling is enhanced by separating the determination of the handling, the responsible party, and the target of the handling.

Quality Assurance

As described in (Grambow, 2013; Grambow & Oberhauser 2010; Grambow, Oberhauser, & Reichert, 2010a; Grambow, Oberhauser, & Reichert, 2011a), software quality assurance (SQA) is a crucial component for every SE project, and various studies (Brooks, 1987; Glass, 1997; Naur & Randell, 1968; Jones, 2010) show that the quality of software systems has been and remains a significant problem. Automated guidance for combining SQA with SEPM (Software Engineering Process Management) is not yet prevalent. Challenges in software development projects are presented at both the product and process levels based on the nature of software artifacts and manually-driven processes. Product intangibility hinders effective retrieval of timely information about its product quality status. At times, software quality measures come into focus and are applied close to release, or, when the project is behind schedule, they may be jettisoned altogether; however, it is generally acknowledged that their application in earlier development stages saves time as well as money (Abdel-Hamid, 1988; Slaughter, Harter, & Krishnan, 1998). The proper application of quality measures is also 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 (Abdel-Hamid, 1988). Furthermore, quality goals can be conflicting (like e.g., performance and reliability) and the measures must match the goals of the project. If a system or tool aims to provide holistic process support for SE projects, integration of quality assurance must also be taken into account and satisfy the following requirements:

- To be aware of problems in the SE project, the system must have a facility to integrate information on SE process or product problems from various sources (e.g., external tools measuring the state of the source code, bug tracking systems).
- To enable automated integration of quality measures at run-time, the system must be aware of quality opportunities, meaning time points when a user can cope with a quality measure. This requires knowledge about the users' schedule, meaning the abstract activities that have been scheduled and estimated for the user.
- Applied quality measures should be automatically chosen during run-time in alignment with project goals in order to match the defined strategy of the project.

- Quality measures should not only rely on detected problems, but also consider common quality enhancement. Thus, proactive and reactive measures should be available.
- Context-sensitive tailoring of proposed measures is desirable considering different factors of the actual situation, e.g., properties of the applying person and application time point.
- The selection of measures should be aware of their effectiveness to optimally match specific environments or situations in different organizations. Therefore, continuous monitoring of the quality of the source code is essential to detect potential impacts of applied measures on the 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.
- The automated distribution of quality measures should not interfere with standard development process execution. It should be rather seamlessly integrated with other workflows that are part of the process. That way, an enhanced traceability of quality measure application can be fostered.
- The quality measure integration should also not interfere with the users. They should not be disturbed by quality measure proposals.

To counteract these problems and to satisfy these requirements regarding the automatic detection of quality problems and opportunities and the automatic contextual integration of software quality measures in the users' process, we propose the following approach, which has also been implemented in the CoSEEEK framework. The approach features three phases: detection, process, and post-processing, as illustrated in the following figure and explained afterwards:

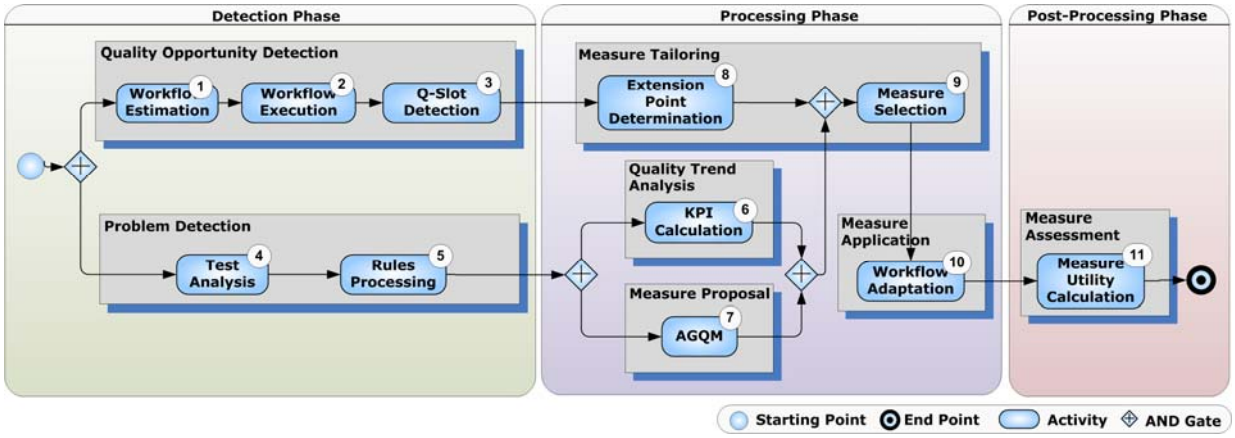

Figure 6. Processing for Automated Quality Assurance

Detection Phase. The *Detection Phase* continuously enables an awareness of the current project situation. For integrating quality measures, two factors are of particular interest: the first is the presence of problems - recognized via the ‘Problem Detection’. This considers primarily problems in the source code like unacceptably high complexity. The second is the availability of opportunities for quality measures (actions) in the users’ schedule - recognized via ‘Quality Opportunity Detection’. The former comprises the following steps:

- Test Analysis: The quality of the source code is measured by static analysis tools that produce reports. These reports are analyzed to gain an awareness of problems and the affected artifacts.

- Rules Processing: In this step automatic rules are applied that specify thresholds for the metrics applied on the code. For all artifacts where a threshold is exceeded, a quality measure is applied automatically.

When a user finishes an activity, the detection for quality opportunities can be started. We call that step 'Q-Slot Detection'. If a Q-Slot is available, an ordered list of proposed measures is generated. This list is used by a tailoring process to select a measure that contextually matches the current situation and user. That measure is then automatically integrated into a workflow. The steps are further described in the following:

- *Workflow estimation*: The different user activities of a project have to be estimated concerning their time consumption.
- *Workflow enactment*: As the workflows are executed by users, the planned activities are completed.
- *Q-Slot detection*: Based on the planned activity times and their real values, it can be determined when a quality measure can be proposed without delaying the process.

Processing Phase. The *Processing Phase* deals with the selection and proposal of the quality measures and involves four steps. Utilizing the Goal-Question-Metric (GQM) technique (Basili, Caldiera, & Rombach, 1994), quality measures are initially proposed in alignment with project goals. To prepare these measures for their automated application, 'Measure Tailoring' incorporates information about the applying persons and the possible points in the users' schedule in which to apply the measure. This leads to a selection of appropriate points (so-called *Extension Points*) and to an automated integration of the quality measures into the concrete workflow of the chosen person.

Post-Processing Phase. Finally, to be able to track the quality of the project continuously, in the Post-Processing Phase, a 'Measure Assessment' is performed utilizing the quality trend analysis. This analysis supports an awareness and automatic assessment of the potential utility of the applied measures, fostering quality. Since each project is unique, the applicability and effectiveness of measures can vary with respect to different projects. Therefore, the system executes an assessment phase to rate the applied measures and to incorporate their impact in the given project.

SQA should be aligned to the SE process being used, and be relevant and applicable at the operation level. The manual combination of SQA with SEPM requires constant vigilance and associated labor in order to avoid missing quality opportunities, to continuously monitor quality goal states, and to adapt measure and measure utility to new quality situations. The application of BPM in SE environments has been sparse due, among other factors, to a lack of contextual adaptability. Automated quality guidance support could assist developers by providing SQA triggering that is based on current and factual data, continuously monitoring quality goal states and trends, and selecting and tailoring measure selection to that being most appropriate in the current situation. This section presented such an automated solution to support the integration of process management and quality assurance in SE, utilizing contextual awareness and dynamic processes to automatically insert suitable quality measures into the SE process.

Knowledge Provisioning

Modernization projects are - as all other SE projects - complex and involve highly knowledge-dependent work, since software development processes are mostly knowledge processes (Kess & Haapasalo, 2002).

Software engineering (SE) and especially software modernization are still relatively new and immature disciplines, and while work has gone into integrating knowledge and process management to support SE, a comprehensive and viable solution is elusive. Currently, process management is typically done in a documentation–centric or agile fashion and lacks automated process support. Knowledge provisioning, in turn, is crucial to enable the distribution of knowledge among different people and to keep and exploit experience gained in various projects. Supporting this with an automated system can be very beneficial (Teigland, Fey, & Birkinshaw, 2000), and important capabilities of such a system are capturing, maintaining, reusing, and transferring knowledge. Wikis are often used for SE knowledge management because of the easy creation and access of information (Schaffert, Bry, Baumeister, & Kiesel, 2008). However, retrieval of contextually relevant information from Wikis remains difficult (Schaffert, Bry, Baumeister, & Kiesel, 2008). Thus, information is captured and stored but its reuse is still problematic. This could be facilitated if knowledge use was connected with process execution (Grambow, 2013; Grambow, Oberhauser, & Reichert, 2011e; Grambow, Oberhauser, & Reichert, 2012c). A framework or system providing holistic process support can achieve this if basic requirements are satisfied:

- To be able to use and disseminate knowledge in a project, the presence of some facility to store knowledge is required. The storage and management of knowledge for users shall not be cumbersome. Additionally, to enable an automated system to access and utilize this information, machine-readable semantic annotation shall be supported.
- Since not all the information users require is stored locally in the system, a facility to integrate external information sources is required. Examples of such information include process documentation or external web pages.
- The knowledge entered by users should be available to an automated system to be able to automatically use it. This includes automatic access to the knowledge store and semantic enhancements to the knowledge structure to enable automatic selection of the suitable knowledge.
- There should be a means to utilize contextual information for knowledge selection. Only that way can it be assured that the automatically provided information matches the current situation of the individual and is therefore more likely to be applicable.
- There shall be a facility to automatically inject the selected knowledge into the standard SE process execution. Users should not be distracted or bothered with additional effort access the knowledge. Seamless integration with everyday work is crucial.
- Certain process information may be specialized and not generally applicable. Furthermore, the amount of information provided must not exceed certain thresholds, so that users are not subjected to information overload and thus overlook important information. Therefore, the provisioning of information should be configurable, e.g., based on the needs or preferences of projects, processes, or users.

In the following we present how knowledge is automatically integrated in a system supporting a project and how it is concretely provided to the users, featuring a concrete example from the software modernization domain.

Knowledge Integration. To provide the user with a consistent knowledge provisioning system (KPS) that realizes holistic information support for the SE process, a set of components have to interact as illustrated by Figure 7. 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 users, artifacts, and various events as well as information from process execution. It receives context information from the Event Management component (1), whose responsibility is the acquisition and aggregation of events from the SE environment (2). This is realized

by a set of sensors integrated in external tools, such as IDEs or source control systems used within a SE project. The Process Management component enables a SE process model implementation as well as operational process support. This is done by means of automated workflows actively governed by that component. The Context and Process Management components work together to enable the usage of context information in the process and to better align process execution with reality.

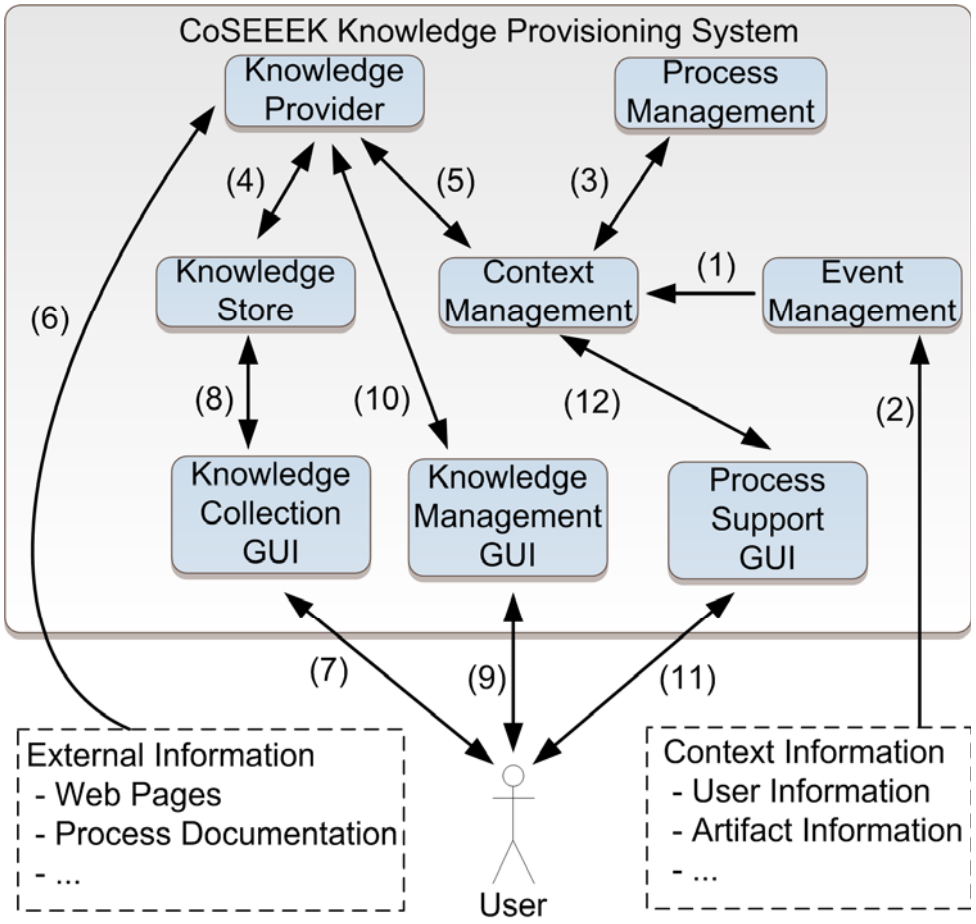

Figure 7. Knowledge provisioning automation

Knowledge provisioning is realized by a *Knowledge Store* and a Knowledge Provider. The former is utilized to store user-relevant SE information and to make it available to the KPS via supplementary machine-readable semantics. The Knowledge Provider, in turn, coordinates that information (4) and provides it to the Context Management component (5) to be injected into the users' workflows (3) in the appropriate context. The Knowledge Provider is also responsible for the abstract definition of user relevant information within the KPS (called Guidance Items), which is referenced by the Context Management component, as well as for the integration of external information resources (6).

User communication is realized as follows: The user can enter relevant project or SE information (e.g., best practices) using the Knowledge Collection GUI (7). That information is stored in the Knowledge Store (8). The Knowledge Management GUI (9) allows users to integrate external information (e.g.,

process documentation at an external web location) or configure the way the information is provided (e.g., ‘This guidance is applicable to this role at that point in the process’). The configuration for information provisioning is stored directly in the Knowledge Provider (10). All support and guidance is then distributed to the user by the Process Support GUI (11). That GUI receives its information from the Context Management component (12), which unites information on the activity and workflow from the Process Management component (3) with additional user-relevant SE information from the Knowledge Provider (5).

Process-centered Knowledge Support. The storage and management of user-related information is realized using the Knowledge Provider and Knowledge Store as well as the two knowledge management GUIs. As already mentioned, information can be collected and stored within the KPS (internal information) and be integrated from other sources (external information). Internal information is collected via the Knowledge Collection GUI that enables users to annotate that information with tags. Examples for tags are ‘junior engineer’, ‘front end development’, or ‘high risk’ that can then be used to automatically and appropriately select the information to support the users in their context. This is accomplished by the Knowledge Provider that also manages the integration of external information sources. For the organization of information in the KPS, the concept of a Guidance Item (GI) is utilized. All guidance the KPS can distribute to users is defined by the GIs created with the Knowledge Management GUI and stored within the Knowledge Provider. GIs are shown to the users as lists (plain lists or checklists). Each GI has a set of tags that are used to describe the information for the KPS as well as for the users entering and managing it. Tags can be any type of identifying property indicating to what the GI applies, like ‘Activity’, ‘Junior Engineer’, or ‘High Risk Artifact’. A GI is an abstract unit of guidance information that may contain an arbitrary number of positions or sub-items. The information defined by it can be static or dynamically compiled by the Knowledge Provider. The latter is only possible for internal information stored in the Knowledge Store. If a GI is internal and dynamic, the Knowledge Provider will use the tags of the GI to query the Knowledge Store for items that are tagged in the same way, creating the GI out of these. Based on its type, a GI will be treated differently by the KPS. For example, a checklist will have a check mark for each sub-item, while plain information will just be shown to the user.

Automatic knowledge support must necessarily be aligned to a person’s context and intent, otherwise it is likely to be irrelevant and the tool rejected. Therefore, the activities performed by the users and governed by the Context Management component are the initiators for GI provisioning. For these activities, four properties decide how the Context Management component presents GIs to the user: These properties are called GI alignment, target obligation, GI usage, and item compilation:

- GI alignment governs when a GI (such as a checklist) is shown to the user in relation to the activity that is the target of the GI. There are three options: ‘Pre’ GIs are shown at the beginning of the activity. ‘Post’ GIs are shown at the end of an activity. ‘Pre/Post’ GIs incorporate both of the aforementioned, allowing, e.g., a checklist to be updated with additional items at the end of the activity.
- Target obligation associates the connection to the target activity. Some GIs like checklists may be directly tied to a target activity. These are called ‘Synchronous’ and their lifecycle depends directly on the target activity. Other GIs may be shown based on certain events (including, e.g., activity termination). These are called ‘Asynchronous’. They are independent of the activities and can have a pre-defined lifetime.
- GI usage provides additional optional information or important information whose incorporation may be required. Therefore, this property distinguishes between ‘Required’ and ‘Optional’ GIs. Using ‘Required’ GIs, the target activity will not be marked complete without also acknowledging the GI.

- Item compilation defines how the items of a GI are compiled: ‘Static’ GIs are pre-defined with a static set of items. ‘Dynamic’ GIs are dynamically built by the system at the time they shall be shown. For these GIs, context properties are incorporated: The GIs can have various tags, like, e.g., ‘Development’. The same applies to information stored in the knowledge store. Consider tags like ‘Development’, ‘Junior Engineer’ and ‘Database’ here. At runtime the system has access to that context information, like 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 coding). So the system can compile, for example, a dynamic database development checklist for junior engineers. As already mentioned, GIs can only be dynamic when they are internal, as the system has no influence on GIs stored in external information sources like web pages.

Utilizing these properties, different types of GIs (e.g., checklists) are possible. In the following, three prominent types are presented for illustration:

- *'Asynchronous static optional assignment pre checklist'*: This type of checklist is intended for a high-level user Assignment like, for example, ‘Develop a new GUI feature’. Being static, the checklist has a pre-configured set of checklist items that is defined in the Knowledge Base. The checklist is configured to be a pre checklist, meaning that is shown to the user when the Assignment is started in order to provide information to the user that is to be considered before he starts working on the Assignment. As this can be relatively general information (as, e.g., to check the requirements specification for the assignment), the checklist is also configured to be asynchronous and optional. This means the information is shown at the beginning, but it is not required to complete the checklist for the completion of the Assignment, and the information does not disappear when the Assignment is completed. The checklists persists for a defined time interval, giving the user 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.
- *'Synchronous dynamic required activity combined checklist'*: This type of checklist relates to a more concrete Assignment Activity that is executed to complete an Assignment, as, e.g., the creation and execution of developer tests. The checklist is dynamic, meaning the system will compile its items dynamically at run time depending on context information, as, e.g., the skill level of the user. As it is a synchronous checklist, its lifecycle is tied to the target Assignment Activity. In this case it is also required and the user can thus not complete the activity without having completed the checklist beforehand. As a combined checklist, it accompanies the activity during its entire processing time: As the activity gets started, initial pre checklist items are shown, helping the user to prepare for the activity. When the user tries to complete the activity, new post checklist items get added, helping the user to not forget important things concerning the activity.
- *'Asynchronous dynamic optional task post checklist'*: This type of checklist is intended for Atomic Tasks that are performed to complete Assignment Activities. Examples for this are ‘Coding’, ‘Debugging’, or ‘Checking in’. In this concrete case, the checklist is asynchronous and optional in order to just give the user the option to consider additional information for the tasks he is performing. Being an asynchronous post checklist, it is shown to the user when he switches tasks, and thus the considered task provides a reminder for things that should be considered at this completion of this task (e.g., at the task ‘Unit test creation’, if all functionality of the class to be tested has been covered). As Atomic Task switches frequently 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, so as not to distract or annoy the user.

We have now presented tree different types of generic automatically distributed checklists. To make the picture of this kind of knowledge provision more vivid, we elucidate a concrete example from the software modernization domain in the following.

Example 5 (Knowledge provisioning): *Figure 8 abstractly illustrates what different actors in the scenario do and how they cooperate to achieve automatic knowledge provisioning support via checklists. (A) During the execution of modernization projects, developers can add knowledge to the knowledge store, e.g., when encountering problems and finding solutions for them. This could be a hint to consider applying the Model-View-Controller (MVC) pattern for an existing GUI component that was not constructed according to the pattern. Developers can tag this GI to support later discovery by humans or by some model-based modernization tool. Other possible tags on a GI could include 'junior' to indicate applicability for junior engineers or 'backend' or 'frontend' to relate them to a specific implementation area. Checklists items specific to the support of modernization can also be included. (B) Project execution is managed and governed automatically by the system by means of different workflows belonging to the development process. (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 the one hand, they can be explicitly and statically pre-defined. On the other hand, all information entered can be tagged to be useable for GIs. (D) The system 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 for such a detected event can be the modification of a source code artifact in an IDE or a check-in to a version control system. 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 system matching information of the current situation as, e.g., the skill level of the user or the time and quality constraints of the project using tags on information in the Knowledge Provider.*

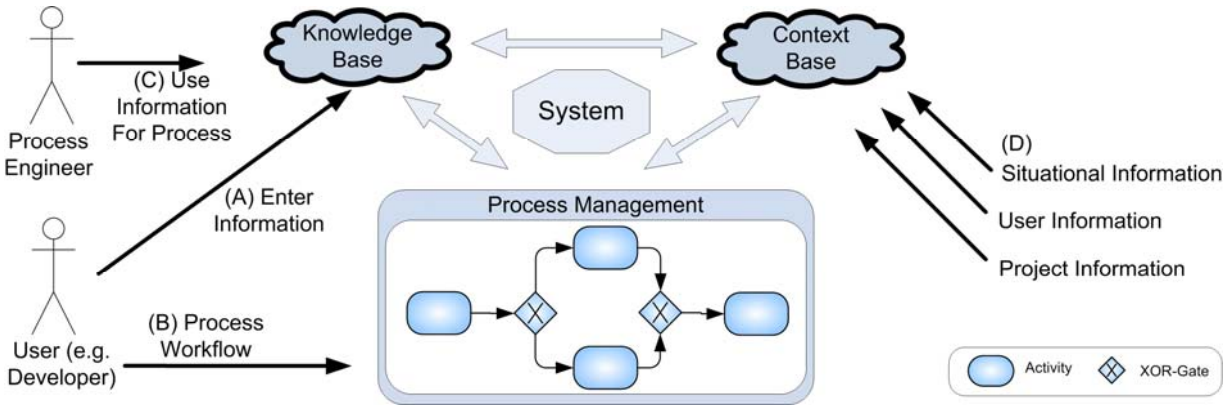

Figure 8. Knowledge provisioning automation

Providing knowledge-based context-aware support for the operational process for the dynamic SE domain remains challenging. In this area, this section contributes an approach for connecting and automating knowledge and process management. Semantic technology is used as link between automatically gathered context information, knowledge resources, and process execution. Thus, it becomes possible to dynamically assemble knowledge relevant to the executing user and to automatically and seamlessly integrate this knowledge with the users' current workflow.

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 user while entering knowledge and enables him to configure the way in which it is later automatically provisioned. Automatic access to the Knowledge Store has been enabled by the system. Having structured knowledge that is enhanced with machine readable semantics allows the system can automatically gather and disseminate that knowledge. The management of the knowledge is done via a separate active component, the Knowledge Provider. The latter also enables integration and management of external knowledge sources. Furthermore, the system can utilize contextual information that is automatically gathered and processed to query knowledge from the Knowledge Store that matches the situation of the user with properties, like the user skill, the project goals, or the implementation area. Finally, the information gathered for the user is provided in a way that is seamlessly integrated with the user's workflow. This supports the use of the knowledge since the information not only fits the user's current needs, but also does not require cumbersome extra work to manually acquire that knowledge.

Process Implementation

This section shows the application of our approach and framework to software modernization projects. Therefore, the modernization process XIRUP was applied to demonstrate these abilities. XIRUP defines four main activities for software modernization. All of these 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*: In this phase, the architecture of the system to be modernized is recovered. Furthermore a preliminary analysis of the system is conducted, which can also influence the modernization decision.
- *Understanding*: In this phase, 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. On this model, an in-depth analysis is conducted. Finally, code generation for prototypes takes place.
- *Building*: In this phase, code is generated for all required components. To be able to test these, 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 will not be accomplished fully automatically without some manual coding involvement. The following figure shows the XIRUP implementation including the four main activities and their sub-processes.

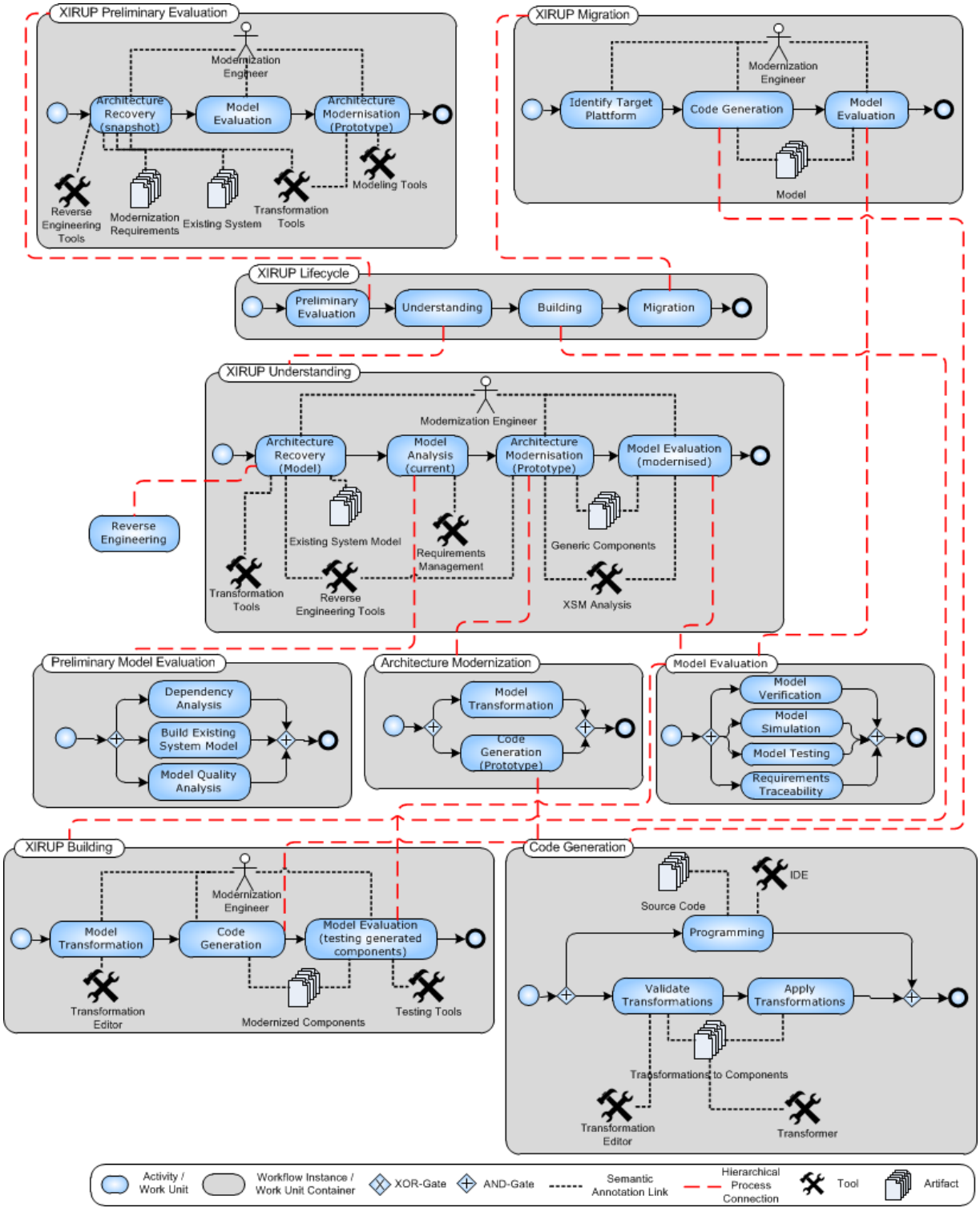

Figure 9. XIRUP process model tailored with manual non-model-based development

Having integrated XIRUP into the CoSEEEK framework, the various features for holistic process and project support can be utilized for the execution of this process. In the following on concrete example from the knowledge management area is shown:

Example 6 (Knowledge provisioning for modernization): *For example, a junior engineer could be provided a GI containing appropriate checklist items as shown in Figure 10 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 might be bothered by some of the "obvious" checklist items, and some items are not applicable to GUIs or databases.*

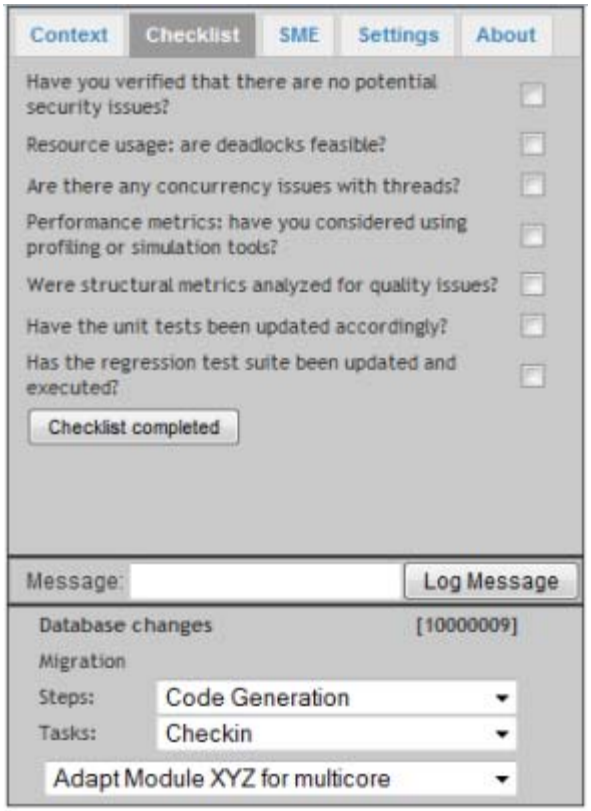

Figure 10. Example contextual checklist provisioned during a software modernization task

FUTURE RESEARCH DIRECTIONS

Towards a more seamless interaction with and automated holistic support of software modernization engineers in their SE environment, including their tool chain, artifacts, knowledge, standards, and processes, much remains to be done. For software modernization projects, the growing size, complexity, and interdependency of software applications and services in addition to the dynamicity of the tools, platforms, languages, processes, and required knowledge will provide continual challenges.

Emergent software archeology tooling is a promising and dynamic area for assisting software engineers with the analysis of legacy systems, including various artifacts and aspects such as business process archeology. As general and project-specific knowledge bases are created (automatically and manually),

this know-how can be automatically and contextually provisioned with the approach described in this chapter.

In the area of software engineering environments, there is an industry trend toward enabling easier integration of heterogeneous tools and data, as evidenced by the Open Services for Lifecycle Collaboration (OSLC) community[viii], which can ease the access to and integration of operational data relevant to the SE process. Also, homogeneous tool chains and Application Lifecycle Management (ALM) and Product Lifecycle Management (PLM) products will play an increasingly important role with a tendency towards an expanded holistic view, including support for software maintenance and modernization efforts. Subsequently, one future research direction is the automated analysis and utilization of the available operational data during a modernization project. Also, the automated discovery and orchestration of tool chains, for instance (Biehl, 2012), will help to reduce the effort associated with configuring and integrating SEEs, enabling greater availability of tooling infrastructures that can be utilized by systems like CoSEEEK.

Another interesting emerging trend is the development of software engineering recommender systems - see (Happel & Maalej, 2008), which could also provide guidance for software engineers during their modernization tasks.

Software modernization has unfortunately received less attention among common software development process models, leaving a disparity. E.g., XIRUP is model-centric and doesn't integrate leftover manual SE tasks, while common SE processes are mostly model- and modernization- oblivious. Future research into software process models should consider the complete long-term lifecycle including modernization and/or provide practical and viable ways for process engineers to consolidate the various needs on modernization projects. This includes a tighter integration of PAIS to support software modernization. Future research opportunities include the automated specification and tailoring of enactable software process models.

CONCLUSION

This chapter described the importance, background, and current issues regarding automated holistic support of software engineers during software modernization projects. With CoSEEEK, a solution approach and implementation was presented that automatically, dynamically, and systematically supports modernization process execution and collaboration.

Various current problem areas in software modernization were outlined, including automated process governance, context integration, process dynamicity, extrinsic process coverage, process exception handling, collaboration and coordination, quality assurance, and knowledge provisioning.

The CoSEEEK solution approach was described, elucidating how it addresses these problem areas to support software modernization holistically. The process implementation mapping from the abstract XIRUP to an operationally enactable workflow was described. This was followed by showing how extended process support for unforeseen extrinsic activities is achieved. Contextual properties were then considered to integrate situational influences into the workflows by applying SME. The section on information gathering described how the user can supply process and product properties that influence the workflow generation. Declarative workflow modeling then described how these properties are used to automatically generate workflows. Task coordination with passive and active collaborative support was discussed. Exception handling then dealt with activity-, process-, and artifact-related exceptions and the automated selection and incorporation of its handling. Quality assurance described the automated support for the selection and management of quality assurance measures. Then knowledge provisioning expounded on the integration of knowledge and process-centered knowledge support.

In conclusion, software modernization remains a high-risk undertaking, especially since the original requirements are being changed dramatically, often the original team members in their original constellation with their original accumulated knowledge and intentions are no longer accessible, and any human knowledge is subject to temporal decay. A holistic approach such as CoSEEEK can mitigate or address a number the larger challenges that software modernization projects currently face.

REFERENCES

Abdel-Hamid, T. K. (1988). The economics of software quality assurance: A simulation-based case study. *MIS Quarterly*, 12(3), 395-411.

Abran, A., & Bourque, P. (2004). *SWEBOK: Guide to the Software Engineering Body of Knowledge.* IEEE Computer Society.

Basili, V. R., Caldiera, V. R., & Rombach, H. D. (1994). The goal question metric approach. *Encyclopedia of Software Engineering*, 2, 528-532.

Bellifemine, F., Poggi, A., & Rimassa, G. (1999). *JADE – A FIPA-compliant agent framework.* Proc 4th Int'l Conf and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents.

Biehl, M., Gu, W., & Loiret, F. (2012). Model-based service discovery and orchestration for OSLC services in tool chains. *Web Engineering*, 283-290.

Brooks, F. P., Jr. (1987). No silver bullet: essence and accidents of software engineering. *Computer*, 20(4), 10-19.

Cánovas Izquierdo, J., & Molina, J. (2009). A domain specific language for extracting models in software modernization. In: R. Paige, A. Hartman & A. Rensink (Eds.), *Model Driven Architecture - Foundations and Applications*, Springer, pp. 82-97.

Comella-Dorda, S., Wallnau, K., Seacord, R., & Robert, J. (2000a). *A survey of black-box modernization approaches for information systems*. Proc Int'l Conf on Software Maintenance.

Comella-Dorda, S., Wallnau, K., Seacord, R., & Robert, J. (2000b). *A survey of legacy system modernization approaches*, Technical Report, No. CMU/SEI-2000-TN-003, Software Engineering Institute, Carnegie Mellon University.

Cook, C., Churcher, N., & Irwin, W. (2004). *Towards synchronous collaborative software engineering*. Proc 11th Asia-Pacific Software Engineering Conf (pp. 230-239).

D31b - Methodology Specification. (2008). MOMOCS (FP6-2005-5) Deliverable d31b. Available at http://www.momocs.org (Accessed 2/28/2012).

Dadam, P., & Reichert, M. (2009). The ADEPT project: a decade of research and development for robust and flexible process support. *Computer Science-Research and Development*, 23(2), 81-97.

Ducasse, S., & Pollet, D. (2009). Software architecture reconstruction: a process-oriented taxonomy. *IEEE Transactions on Software Engineering*, 35(4), 573-591.

Ellis, C., Keddara, K., & Rozenberg, G. (1995). *Dynamic change within workflow systems*. Proc ACM Conf on Organizational Computing Systems.

Gasevic, D., Djuric, D., & Devedzic, V. (2006). *Model driven architecture and ontology development*. Springer-Verlag.

Gelernter, D. (1985). Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1), 80-112.

Gibson, D. L., Goldenson, D. R., & Kost, K. (2006). *Performance results of CMMI-based process improvement*, Technical Report, No. CMU/SEI-2006-TR-004, Software Engineering Institute, Carnegie-Mellon University.

Glass, R. L. (1997). *Software runaways: monumental software disasters.* Prentice Hall.

Grambow, G. (2013), Context-aware process management for the software engineering domain. PhD Thesis, Ulm University, Germany (to appear)

Grambow, G., & Oberhauser, R. (2010). *Towards automated context-aware selection of software quality measures*. Proc 5th Int'l Conf on Software Engineering Advances.

Grambow, G., Oberhauser, R., & Reichert, M. (2010a). *Employing semantically driven adaptation for amalgamating software quality assurance with process management*. Proc 2nd Int'l Conf on Adaptive and Self-adaptive Systems and Applications.

Grambow, G., Oberhauser, R., & Reichert, M. (2010b). *Semantic workflow adaption in support of workflow diversity*. Proc 4th Int'l Conf on Advances in Semantic Processing, Florence, Italy.

Grambow, G., Oberhauser, R., & Reichert, M. (2011a). Contextual injection of quality measures into software engineering processes. *Int'l Journal on Advances in Software*, 4(1 & 2), 76-99.

Grambow, G., Oberhauser, R., & Reichert, M. (2011b). *Event-driven exception handling for software engineering processes*. Proc 5th Int'l Workshop on Event-driven Business Process Management, Springer, LNBIP 99.

Grambow, G., Oberhauser, R., & Reichert, M. (2011c). *Semantically-driven workflow generation using declarative modeling for processes in software engineering*. Proc 4th Int'l Workshop on Evolutionary Business Processes.

Grambow, G., Oberhauser, R., & Reichert, M. (2011d). *Towards automatic process-aware coordination in collaborative software engineering*. Proc 6th Int'l Conf on Software and Data Technologies.

Grambow, G., Oberhauser, R., & Reichert, M. (2011e). *Towards dynamic knowledge support in software engineering processes*. Proc 6th Int'l Workshop on Applications of Semantic Technologies, held in conjunction with INFORMATIK'11, Koellen, LNI 192.

Grambow, G., Oberhauser, R., & Reichert, M. (2012a). Contextual generation of declarative workflows and their application to software engineering processes. *Int'l Journal on Advances in Intelligent Systems*, 4(3 & 4), 158-179.

Grambow, G., Oberhauser, R., & Reichert, M. (2012b). Enabling automatic process-aware collaboration support in software engineering projects. *Communications in Computer and Information Scien*ce (CCIS) 303, 73-89.

Grambow, G., Oberhauser, R., & Reichert, M. (2012c). *Knowledge provisioning: a context-sensitive process-oriented approach applied to software engineering environments*. Proc 7th Int'l Conf on Software and Data Technologies.

Grambow, G., Oberhauser, R., & Reichert, M. (2012d). *User-centric abstraction of workflow logic applied to software engineering processes*. Proc Advanced Information Systems Engineering Workshops, Springer, pp. 307-321.

Happel, H.-J., & Maalej, W. (2008). *Potentials and challenges of recommendation systems for software development*. Proc Int'l Workshop on Recommendation Systems for Software Engineering.

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., & Dean, M. (2004). *SWRL: a semantic web rule language combining OWL and RuleML*. W3C Member submission, 21, 79.

ISO/IEC 14764. (2006). *Software engineering--software life cycle processes--maintenance*. Geneva, (Switzerland): International Organization for Standardization.

Izquierdo, J. L. C., Cuadrado, J. S., & Molina, J. G. (2008). *Gra2MoL: A domain specific transformation language for bridging grammarware to modelware in software modernization.* Proc Workshop on Model-Driven Software Evolution.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The unified software development process*: Addison-Wesley.

Jiang, T., Ying, J., & Wu, M. (2007). CASDE: an environment for collaborative software development. *Computer Supported Cooperative Work in Design III*, 367-376.

Johnson, P. M. (2007). *Requirement and design trade-offs in Hackystat: An in-process software engineering measurement and analysis system*. Proc 1st Int. Symp. on Empirical Software Engineering and Measurement.

Jones, C. (2010). Get Software Quality Right. *Dr Dobb's Journal*.

Kess, P., & Haapasalo, H. (2002). Knowledge creation through a project review process in software production. *Int'l Journal of Production Economics*, 80(1), 49-55.

Koru, A. G., Zhang, D., El Emam, K., & Liu, H. (2009). An investigation into the functional form of the size-defect relationship for software modules. *IEEE Transactions on Software Engineering*, 35(2), 293-304.

Koskinen, J., Ahonen, J. J., Sivula, H., Tilus, T., Lintinen, H., & Kankaanpaa, I. (2005). *Software modernization decision criteria: an empirical study*. Proc 9th European Confe on Software Maintenance and Reengineering.

Koskinen, J., Lintinen, H., Ahonen, J. J., Tilus, T., & Sivula, H. (2005). *Empirical study of industrial decision making for software modernizations*. Proc Int'l Symposium on Empirical Software Engineering.

Lanz, A., Reichert, M., & Dadam, P. (2010). *Making business process implementations flexible and robust: error handling in the AristaFlow BPM Suite*. Proc CAiSE'10 Forum.

Lewandowski, A., & Bourguin, G. (2007). Enhancing support for collaboration in software development environments. *Computer Supported Cooperative Work in Design III*, 160-169.

Lewis, G., Morris, E., & Smith, D. (2005). *Service-oriented migration and reuse technique (smart)*. Proc 13th IEEE Int'l Workshop on Software Technology and Engineering Practice, pp. 222-229.

Luckham, D. C. (2001). *The power of events: an introduction to complex event processing in distributed enterprise systems*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

Luo, Z., Sheth, A., Kochut, K., & Miller, J. (2000). Exception handling in workflow systems. *Applied Intelligence*, 13(2), 125-147.

McBride, B. (2002). Jena: A semantic web toolkit. *IEEE Internet Computing*, 6(6), 55-59.

Meier, W. (2009). eXist: An open source native XML database. *Web, Web-Services, and Database Systems*, Springer, *LNCS*, 2593, 169-183.

Mens, T., Wermelinger, M., Ducasse, S., Demeyer, S., Hirschfeld, R., & Jazayeri, M. (2005). *Challenges in software evolution*. Proc 8th Int'l Workshop on Principles of Software Evolution, pp. 13-22.

Mukerji, J., & Miller, J. (2003). *MDA Guide Version 1.0.1*. Object Management Group.

Naur, P., & Randell, B. (1969). *Software engineering: report on a conference sponsored by the NATO Science Committee*. Scientific Affairs Division, NATO.

Oberhauser, R. (2010). Leveraging semantic web computing for context-aware software engineering environments. *Semantic Web*, Gang Wu (ed.), In-Tech, Vienna, Austria.

O'Brien, P. D., & Nicol, R. C. (1998). FIPA—towards a standard for software agents. *BT Technology Journal*, 16(3), 51-59.

Paton, N. W. (1999). *Active rules in database systems*. Springer Verlag.

Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., & Reijers, H. A. (2011). *Imperative versus declarative process modeling languages: an empirical investigation*. Proc 2nd Int'l Workshop on Empirical Research in Business Process Management.

Pérez-Castillo, R. (2012). *MARBLE: Modernization approach for recovering business processes from legacy information systems*. Proc 28th IEEE Int'l Conf on Software Maintenance.

Pérez-Castillo, R., de Guzman, I. R., & Piattini, M. (2011). Business process archeology using MARBLE. *Information and Software Technology*, 53(10), 1023-1044.

Perez-Castillo, R., de Guzman, I. R., Piattini, M., & Avila-Garcia, O. (2009). *On the use of ADM to contextualize data on legacy source code for software modernization.* Proc 16th Working Conf on Reverse Engineering, pp. 128-132.

Pérez-Castillo, R., de Guzman, I. R., Piattini, M., & Ebert, C. (2011). Reengineering technologies. *IEEE Software*, 28(6), 13-17.

Prud'hommeaux, E., & Seaborne, A. (2006). SPARQL query language for RDF. W3C WD 4.

Putrycz, E., & Kark, A. (2007). Recovering business rules from legacy source code for system modernization. In: A. Paschke & Y. Biletskiy (Eds.), *Advances in Rule Interchange and Applications*, pp. 107-118.

Rausch, A., Bartelt, C., Ternité, T., & Kuhrmann, M. (2005). *The V-model XT applied–model-driven and document-centric development.* Proc 3rd World Congress for Software Quality, Vol. III.

Ralyté, J., Brinkkemper, S., & Henderson-Sellers, B. (2007). *Situational method engineering: Fundamentals and experiences*. Springer.

Reichert, M., & Dadam, P. (1998). $ADEPT_{flex}$—supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems*, 10(2), 93-129.

Reichert, M., & Dadam, P. (2009). Enabling adaptive process-aware information systems with ADEPT2 *Handbook of Research on Business Process Modeling*, New York: Information Science Reference, Hershey, pp. 173-203.

Reichert, M., Dadam, P., Rinderle-Ma, S., Jurisch, M., Kreher, U., & Göser, K. (2009). Architectural principles and components of adaptive process management technology. *PRIMIUM - Process Innovation for Enterprise Software, Koellen, LNI*, P-151, 81-97.

Reichert, M., Rinderle-Ma, S., & Dadam, P. (2009). Flexibility in process-aware information systems. *Transactions on Petri Nets and Other Models of Concurrency II, LNCS*, 5460, 115-135.

Reichert, M., & Weber, B. (2012). *Enabling Flexibility in Process-aware Information Systems – Challenges, Methods, Technologies*. Springer.

Russell, N., ter Hofstede, A. H. M., Edmond, D., & van der Aalst, W. M. P. (2004). *Workflow resource patterns*. Eindhoven Univ. of Technology.

Russell, N., van der Aalst, W. M. P., & ter Hofstede, A. H. M. (2006). *Exception handling patterns in process-aware information systems*. Proc 23rd Int'l Conf. on Advanced Information Systems Engineering.

Schaffert, S., Bry, F., Baumeister, J., & Kiesel, M. (2008). Semantic wikis. *IEEE Software*, 25(4), 8-11.

Schwaber, K., & Beedle, M. (2001). *Agile software development with Scrum* (Vol. 18): Prentice Hall.

Seacord, R. C., Plakosh, D., & Lewis, G. A. (2003). *Modernizing legacy systems: software technologies, engineering processes, and business practices*: Addison-Wesley Professional.

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., & Katz, Y. (2007). Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2), 51-53.

Slaughter, S. A., Harter, D. E., & Krishnan, M. S. (1998). Evaluating the cost of software quality. *Communications of the ACM*, 41(8), 67-73.

Tan, Y., & Mookerjee, V. S. (2005). Comparing uniform and flexible policies for software maintenance and replacement. *IEEE Transactions on Software Engineering*, 31(3), 238-255.

Teigland, R., Fey, C. F., & Birkinshaw, J. (2000). Knowledge dissemination in global R&D operations: an empirical study of multinationals in the high technology electronics industry. *MIR: Management International Review*, 49-77.

van Genuchten, M., & Hatton, L. (2012). Compound annual growth rate for software. *IEEE Software*, 19-21.

Wallmüller, E. (2007). *SPI-Software Process Improvement mit Cmmi und ISO 15504*. Hanser Verlag.

Weber, B., Reichert, M., & Rinderle-Ma, S. (2008). Change patterns and change support features-enhancing flexibility in process-aware information systems. *Data & Knowledge Engineering*, 66(3), 438-466.

Weber, B., Reichert, M., Wild, W., & Rinderle-Ma, S. (2009). Providing integrated life cycle support in process-aware information systems. *Int'l Journal of Cooperative Information Systems*, 18(1), 115-165.

Weber, B., Sadiq, S., & Reichert, M. (2009). Beyond rigidity–dynamic process lifecycle support. *Computer Science-Research and Development*, 23(2), 47-65.

Weiderman, N. H., Bergey, J. K., Smith, D. B., & Tilley, S. R. (1997). *Approaches to legacy system evolution*, TR No. CMU/SEI-97-TR-014, Software Engineering Institute, Carnegie Mellon University.

Weiderman, N., Northrop, L., Smith, D., Tilley, S., & Wallnau, K. (1997*). Implications of distributed object technology for reengineering*, TR No. CMU/SEI-97-TR-005, Software Engineering Institute, Carnegie Mellon University.

World Wide Web Consortium (2004a). *Resource Description Framework (RDF) Concepts and Abstract Syntax*.

World Wide Web Consortium (2004b). *OWL Web Ontology Language Semantics and Abstract Syntax.*

Zugal, S., Pinggera, J., & Weber, B. (2011a). *Creating declarative process models using test driven modeling suite*. Proc CAiSE Forum.

Zugal, S., Pinggera, J., & Weber, B. (2011b). *The impact of testcases on the maintainability of declarative process models*. Proc Int'l Working Conf on Enterprise, Business-Process and Inf Systems Modeling.

[i] http://www.momocs.org/
[ii] http://www.eclipse.org/modeling/emf/
[iii] http://www.hitech-projects.com/euprojects/serious/
[iv] http://adm.omg.org/
[v] http://esper.codehaus.org/
[vi] http://www.jboss.org/drools/
[vii] http://pmd.sourceforge.net/
[viii] http://open-services.net/