Software Visualization – A Process Perspective

Juergen Rilling$^1$, Wen Jun Meng$^1$, Fuzhi Chen$^1$, Philippe Charland$^2$

Department of Computer Science$^1$
and Software Engineering
Concordia University, Montreal, Canada
{rilling, w_meng, f_ch}@cse.concordia.ca

System of Systems Section$^2$
Defence R&D Canada
Val-Bélair, Canada
philippe.charland@drdc-rddc.gc.ca

Abstract

Software visualization is one of the enabling techniques to provide support during software maintenance activities. Software maintenance is a multi-dimensional problem domain which involves the integration, abstraction and analysis of different knowledge resources and artifacts. Maintainers are typically left with no guidance on how these existing artifacts, tools and knowledge should be utilized to complete a particular task. In this research, we present a novel visualization approach that integrates these artifacts in the software maintenance process chain.

1. Introduction

Software maintenance creates an ongoing challenge for both the research community and tool developers, due to the variations and interrelationships that exist among software artifacts, knowledge resources, and the maintenance process and tasks [24, 25]. There has been little work in examining how these resources work together for end users [25] and how they can collaboratively support a specific program maintenance task. Maintainers are often left with no guidance on how to complete a particular task in a given context using a set of available resources (e.g., tools, artifacts, etc.).

In [19], we have introduced a unified ontological knowledge representation to model both maintenance process models and the resources relevant to activities related to these maintenance processes. From a user perspective, it is the interaction between users (maintainers), process model (steps to be performed) and context (knowledge and information resources available) of a current task that defines the overall applicability of a model. Ideally, users should become immersed in such a process while a particular task unfolds. In this research, we focus on the visual integration of the process and resources knowledge modeled in the ontological representation of the process chain.

The visual representation allows for the integration of different abstraction views, analysis techniques and visual metaphors to provide maintainers with guidance during specific aspects of maintenance activities.

Fig. 1  Program comprehension – a contextual view

Figure 1 presents a general overview of our approach which involves four major parts. (1) Knowledge resources relevant to a particular maintenance task (e.g., software artifacts, tools, users) are modeled in an ontological knowledge representation (Section 3.1). (2) An existing software maintenance process model (IEEE 14764 [12]) is adopted and formalized (Section 3.2) to provide a framework for integrating these resources. (3) The resulting unified knowledge base (KB) is explored, queried and reasoned upon to enable different levels of context awareness (Section 3.3). (4) A visualization framework that supports a contextual visualization of knowledge resources, tools and artifacts during typical software maintenance activities is presented (Section 4).

Similar to more traditional source code visualization techniques [9, 16, 17], we apply a combination of data abstraction, filtering and analysis to guide maintainers during typical comprehension and maintenance
tasks. Our goal in this research is not to provide a service or data integration among resources. Rather, we focus on how existing resources and techniques can be applied and integrated within a maintenance process to establish contextual views based on the current process activities and relevant resources. The main contributions of our approach are therefore: (1) Development of a visualization environment that supports the visual integration of different knowledge resources (e.g., tools, data) within a software maintenance process context. (2) Establishing traceability between software maintenance related processes activities and the resources supporting these activities, such as software visualization tools and techniques.

The remainder of the article is organized as follows: Relevant research background is introduced in Section 2. Section 3 describes in detail the context-driven process model. Section 4 focuses on implementation issues and demonstrates the use of the visualization approach. Related work is presented in Section 5, followed by conclusions and future work in Section 6.

2. Background

In this section, we introduce background relevant to our research, including software process modeling, ontologies and their application in software engineering, as well as a brief software visualization overview.

2.1 Software Process Modeling

In the past decades, several cognitive models [6, 18, 22] have been developed to explain general strategies on how maintainers form mental models of source code. A mental model describes a maintainer’s current understanding of a software system, while a cognitive model describes both the cognitive processes and information structures needed to create the mental model [24]. Some of the challenges in creating mental models are directly related to the contextual setting of a maintenance task. Such a contextual setting includes aspects like a maintainers’ skills (e.g., programming experience), a software system characteristics (e.g., application domain, size and complexity), the task type (e.g., bug fix, component substitution), as well as available and applicable tools and software artifacts.

Process models in general can be described as networks of activities and relationships among them. Many software maintenance models have been proposed in the literature, including the quick-fix, iterative enhancement, full-reuse model [4], the staged model [5], SEI CMMI model [7] and the IEEE 14764-2006 software maintenance life cycle model [12]. The latest IEEE 14764-2006 maintenance model for example describes an interactive process for managing and executing software maintenance tasks. These activities encode the best practices and expertise in the software maintenance community. However, the standard lacks details on how to implement or perform the activities and tasks included in the process [12]. The standard also does not specify how any available resources (tool, system, user expertise, software artifacts, etc.) should be integrated within the process. Furthermore, there exists no tool support in providing users with guidance in utilizing existing and applicable knowledge resources throughout the process.

2.2 Ontologies and Software Engineering

Research in cognitive science suggests that mental models can take on many forms, but the content normally constitutes an ontology [15]. Ontologies are often used as a formal, explicit way of specifying the concepts and relationships in a domain of discourse [3]. OWL is a standard ontology language for semantic web applications, recommended by the World Wide Web Consortium (W3C). There are three sublanguages of OWL, in ascending level of expressivity: OWL-Lite, OWL-DL, and OWL-Full. In our approach we utilize OWL-DL, which is based on Description Logic (DL) and provides the best tradeoff between expressivity and reasoning power. An OWL ontology contains both TBox axioms and ABox assertions. The TBox specifies the terminology or vocabulary of an application domain and the ABox contains assertions about individuals in accordance with the terminology.

Commonly, OWL ontologies have been used as a data storage medium similar to traditional databases. However, a model constructed with OWL-DL is often more precise and expressive than traditional data semantics [3]. OWL-DL allows establishing information relationships to different data resources and performing sophisticated queries across them. One important feature of OWL-DL is the possibility to exploit reasoning to identify implicit knowledge. Inference services provided by DL reasoners include TBox reasoning (e.g., concept consistency, subsumption, classification, and ontology consistency) and ABox reasoning (e.g., instance checking, instance retrieval, tuple retrieval, and instance realization). For a more detailed coverage of DL and reasoning services, we refer the reader to [3, 10].

Ontologies are also the foundation for the semantic web, an emerging technology that has been studied in

1 http://www.w3.org/TR/owl-features/
many areas within the software engineering domain, such as model-driven software development, [20], maintenance knowledge conceptualization [8,13] components reuse [11] and open source bug tracking [1], etc.

2.3 Software Visualization

Traditionally, software visualization approaches have addressed a wide variety of problems that range from algorithm animation, visual programming, and visualizing recovered structural information form large-scale legacy systems. It is in particular in the area of program comprehension and analysis where software visualization is commonly applied to improve the understanding of inherently invisible and intangible software artifacts and resources. In [23], seven basic visualization principles have been identified that should be supported by typical visualization tools. These principles include among others the ability to zoom, filter, extract, relate and establish contextual views between the activity to be supported and the information to be visualized.

A large number of tools such as SeeSoft [9], Creole [17] and CodeCrawler [16] exist to provide maintainers with some understanding of software systems. However, current research in software visualization tends mainly to focus on developing techniques, metaphors and tools to tackle specific aspects of the comprehension problem [21]. Software visualization tools typically lack guidance on how and when these specialized tools should be applied to support a given task or process. Within current visualization tools, it remains a maintainer’s responsibility to establish a common context that relates maintenance tasks/process to the views created by the visualization tools. Compared with other application domains, such as business process visualization or workflow visualization [2], only limited work exists in providing a visual integration of resources within a software maintenance process context.

3. Software Maintenance Process Model

This section introduces our formal unified ontological model that conceptualizes and integrates activities involved in a maintenance process, as well as their interactions with existing resources and knowledge.

3.1 Knowledge Integration through Ontologies

The unified model is based on OWL-DL to provide an ontological representation of both the software maintenance process and the knowledge resources relevant to perform maintenance activities. Some of the benefits for utilizing an ontological representation are:

**Knowledge acquisition and evolution.** The knowledge acquisition, exploration and management play a dominant role in software maintenance. An ontological representation enables us not only to explore existing knowledge, but also to add newly gained knowledge in the form of concepts and relationships, as well as new instances to an existing KB.

**Reasoning.** Reasoning can be applied to imply knowledge from an explicit modeled KB. OWL-DL ontologies support the use of reasoning services provided by DL reasoners, e.g., the classification of concepts and instances in the KB as well as the inference of knowledge through transitive closure across the sub-ontologies.

![Fig. 2 Software maintenance meta-model](image-url)
3.2 Modeling Software Maintenance

As the foundation for our maintenance model, we have adopted the most recent IEEE software maintenance standard [12]. The IEEE standard describes activities for both managing and executing software maintenance tasks. The starting point for the model is a maintenance request, which is refined within the standard by six major activities: process implementation, problem and modification analysis, modification implementation, maintenance review/acceptance, migration and retirement. Each of these activities is again described by a list of tasks, with each task further refined by a list of task steps. These task steps correspond to the most fine-grained activities specified in the maintenance standard.

Presently, we are limiting our modeling scope on a subset of these activities, namely the problem modification analysis, modification implementation and acceptance phase. These activities are closely related to comprehension aspects involved in maintenance tasks. It also has to be pointed out that the standard document itself lacks both a formal strategy on how existing resources can be integrated within these activities and how to utilize the standard within a specific organizational context.

We address these issues by establishing explicit relationships between the activities described in the standard document and the knowledge resources available, such as user, tool, artifact, task, etc. Having this unified formal model, DL reasoning services are applied to infer relevant knowledge (instance retrieval), classify, categorize and resolve transitive dependencies among the maintenance activities and the KB resources [10].

3.3 Establishing Context Sensitivity

The generality of existing maintenance process models [7, 12] can become, from an organizational viewpoint, a major challenge in adopting and utilizing them. Organizations are often left alone in establishing a context-awareness and customization among the process model, their own internal processes and the resources available. Context-awareness therefore can generally be described as the steps involved in identifying any information that might be relevant to characterize the situation of an entity. An entity is typically a knowledge resource that is considered relevant to the interaction between users and the process. Based on these assumptions, a process can be called context-aware if it establishes a relationship between relevant information resources, users and organizational factors that have a direct/indirect impact on a specific task.

From a user perspective, it is this interaction between users (maintainers), the process model (steps to be performed) and the context (knowledge and information resources available) of the current task that defines the overall applicability of a model. In what follows (Figure 3), we introduce different levels of process context-sensitivity (level 0-3) that support the integration and use of knowledge resources. The process context is established through constraints on both the scope and the knowledge inference used during information retrieval.

From a more pragmatic point of view, these context-sensitivity levels can be applied to reduce the conceptual gap that typically exists between abstract process models and knowledge resources. Predefined queries are provided to support the different context sensitive levels, allowing to retrieve knowledge relevant to the specific maintenance task and environmental constraints. Our approach can therefore provide maintainers with guidance in utilizing and reusing existing knowledge resources while supporting the activities specified in the maintenance process.
4. An Integrated Visualization Approach

Process-centered support systems are based on the premise that in order to provide users with a real and substantial benefit, the right information should be made available at the right time in the right format. What constitutes the right information in the right format and at the right time depends on the available knowledge resources and the given task setting. From a software maintainer’s perspective, it is the need to adjust both information and context in real time to support the current task being performed. Our visual representation can establish traceability between high-level process artifacts, supporting techniques and tools (e.g., software visualization, source code analysis) in a given maintenance context.

4.1 System Overview

The environment that we developed to support our unified model is built on top of a client server architecture with both the server and the rich client being implemented as Eclipse plug-ins (Figure 4).

Ontology, persistent storage management, and reasoning services are provided through the server application. It is implemented in Java and built on top of the Jena Semantic Web Framework\(^2\). Jena provides the backend repository of both concepts (TBox) and instance data (ABox). Leveraging Jena’s persistent storage support, the model storage can be based on files or on a relational database. TBox management is centralized on the server to ensure consistency, quality of the ontology design, and standardization of the KB and its internal dependencies. Reasoning services are provided by Racer [10], which includes highly optimized TBox and ABox reasoning capabilities. Context sensitivity levels are established through constraints applied on the queries and reasoning services.

The rich client provides the foundation for our visual integration, supporting the visual traceability between the maintenance activities and the relevant resources. It also provides a process interface that is build on top of the local process manager. The visual integration includes an ABox management interface. Similar to more traditional database applications, ABox management (population and queries) is performed through the rich client. In addition, the rich client also provides an interface to access a set of pre-defined queries to support the activities described as part of the process model. The contextual process views are established on the client side through our process viewer. The process viewer integrates both the contextual navigation through the KB (pre-defined queries) as well as the query interface for user-defined queries.

4.2 Visual Integration

The goal of our visual process integration is to establish a visual traceability link from the process requirements (activities) to the resources (tools, techniques, and users) available in a given context.

---

\(^{2}\) http://jena.sourceforge.net/
Within our approach, we support context sensitivity through customizable views, abstraction hierarchies, and support for both filtering and extracting details-on-demand. In what follows, we provide an overview of the contextual views to support the integration of process activities and knowledge resources within a unified representation framework.

A software maintenance process typically originates from a modification request (MR). MRs are also the basis of the IEEE maintenance standard to establish the general context of a particular task, its scope and the potential resources and modifications it involves. Similarly, since our approach follows closely the IEEE maintenance process, it starts with a request view (Figure 5). This request view manages all the MR specific information and establishes the current task context through specific resource views (e.g., user information, tools context, etc.). Traceability between the process and the available resources is created through ontological queries to automatically populate the resource views. The task navigation on the left pane of the screen provides a quick process navigation allowing for expanding/collapsing of process activities details.

Figure 6 illustrates some of these contextual views provided by our environment used to establish traceability between the process activities and the KB resources. Once an initial context through an MR have been established (Figure 5), the IEEE process model and its six main phases are displayed (color coded) as a visual anchor for the detailed contextual views (Figure 6 (A)). The process viewer supports different “details-on-demand” contextual views, allowing users to navigate through more specific contextual information. Figure 6(B) provides a hierarchical view that allows the expansion of the process activity into more detailed sub-activities. Each of these activities and its sub-tasks are supported by a set of contextual queries. These contextual queries provide users with suggestions of applicable resources (tools, users, etc.) and artifacts to support the current activity. Depending on the selected context support level (section 3.3), the resource selection and suggestion will be updated correspondingly. Focusing on a specific sub-task will update the on-demand views of the knowledge resources (e.g., user, tools, etc.) to reflect the new current activity context.

We also provide for an additional filtering and visual navigation to explore both the ABox (instances) and TBox (structural dependency) information in the KB (Figure 6D). Users can view both instance information (ABox) and structural information (TBox). Furthermore, we provide also a query interface to allow users to define their own queries to explore, filter, infer and extract knowledge from the KB.

In Figure 6 (C), a contextual view of visualization tools that are available and applicable in the given process context is provided, with the cone indicating the current process context. Colors are used to represent the applicability of resources for a current activity context. For example, the tools applicable in the current context are highlighted in white. The dark
(blue) cells correspond to tools that are available for this particular MR, but are not applicable in the given context. The contextual views for the different resources are available through tabs, which allow for easy switching among views. The right hand side panel provides a query interface allowing for the execution of both pre-defined and user defined queries to further filter and refine these contextual views. A color coded progress bar at the bottom of the window indicates the current process progress.

It has to be noted that one of the main challenges for our approach is the availability and quality of the KB. In this respect, our approach does not differ from any existing knowledge base or data mining approach. We are following currently two main strategies to address these problems. Firstly, we are currently performing several case studies in a controlled environment that restricts both the type of task performed (component substitution) and the resources to be modeled. This limited knowledge corpus is sufficient to provide an initial proof of concept that our ontological representation can capture the relevant information and the presented context levels can be derived. We have conducted one component substitution case study on Debrief3, a medium size open source application for military maritime exercises. The maintenance task was about substituting a non-secure XML data exchange component of Debrief by an open source component performing XML encryption for a new security request. The study was performed in a controlled environment and supported a context level 0. After the study, our ontology was refined and enriched. It now contains 290 concepts, 97 relationships and 660 instances. The collected feedback confirmed our initial hypothesis: at context level 0, the information associated with each maintenance activity did not provide sufficient guidance for the task.

In a second step, the KB, including both tasks and resources being supported, will be extended and generalized. At this point, ontology population becomes a major factor. We are currently exploring different avenues, including the creation of an Internet community portal and/or the support for an organization specific maintenance process KB.

5. Related Work

Very little work has been performed so far in formalizing and visualizing software maintenance processes. Dias et al. [8] applied first order logic to formalize knowledge involved in software maintenance. Ruiz et al. [13] proposed a semi-formal ontology for managing software maintenance projects. Common to these approaches is that their main focus is on knowledge conceptualization. They focus on standardizing terminology and relationships in the maintenance domain and fall short of providing the integration of resources in their processes. Furthermore, they lack tool support and the necessary visual integration of both process and resources.

In [2], Aversano et al. conducted an initial project on introducing business workflow management technologies to support distributed team work in software maintenance processes. Their approach was built on top of Ultimus4 - a commercial business process modeling tool suite. The process workflow prototype and visualization were created using Ultimus. Compared to this research, we utilize a more formal modeling technique to formalize and unify both the maintenance process and its related resources. Furthermore, our approach enables the use of reasoning services to provide a process centered visual integration environment that provides maintainers with a contextual guidance during typical maintenance tasks.

A significant body of research exists in software visualization, focusing on providing different visual representation of the source code to assist with program comprehension and maintenance in general [9, 16, 17]. Compared to these traditional approaches, our research takes a broader perspective. It has to be seen complementary to these existing visualization approaches, since we focus on how existing tools can be integrated within a given maintenance context. Our work is also complementary to existing work on service integration [14], since this integration effort focuses on enhancing the applicability of tools for different task contexts. In addition, we visualize the IEEE standard maintenance process and integrate relevant resources in an environment to provide maintainer’s with active guidance during typical maintenance tasks.

6. Conclusions and Future Work

Common to most software maintenance process models is that they share a certain generality in abstracting and describing activities to be performed and resources to be used as part of their process model. Organizations are therefore often left alone in establishing a context-awareness to support their processes.

The main contribution of our research is the integration from both a modeling and visual perspective of resources (like software visualization tools) within a maintenance process. Our work promotes the use of

3 http://www.debrief.info/index.php
4 http://www.ultimus1.com
formal ontologies and automated reasoning in software maintenance research, by utilizing a DL-based ontology and reasoning services. In addition, we provide a novel approach to integrate resources (like software visualization tools, artifacts, user experience, etc.) within a particular process context. The resulting unified representation enables user guidance at different context sensitive levels by identifying available and relevant resources to be utilized as part of the maintenance process. Moreover, we deliver a visualization environment which visually integrates resources and maintenance process activities in a common visualization framework. The visualization environment allows for the navigation and exploration of the knowledge base while providing contextual views and representation supporting the various maintenance process activities.

As part of our future work, we will conduct additional case studies to further populate our KB and validate our context sensitive views.

Acknowledgement

This research was partially funded by Defence Research and Development Canada Valcartier (contract no. W7701-052936/001/QCL).

7. References