Decision Support Systems 45 (2008) 922–936

Contents lists available at ScienceDirect

Decision Support Systems

journal homepage: www.elsevier.com/locate/dss

Improving change management in software development: Integrating traceability and software configuration management

Kannan Mohan [a,*], Peng Xu [b], Lan Cao [c], Balasubramaniam Ramesh [d]

[a] Department of Computer Information Systems, Baruch College, The City University of New York, United States
[b] Department of Management Science and Information Systems, University of Massachusetts Boston, United States
[c] Department of Information Technology and Decision Sciences, Old Dominion University, United States
[d] Department of Computer Information Systems, Georgia State University, United States

ARTICLE INFO

Article history:
Received 20 July 2005
Accepted 20 March 2008
Available online 28 March 2008

Keywords:
Software configuration management
Traceability
Change management
Process knowledge

ABSTRACT

Software configuration management (SCM) and traceability are two prominent practices that support change management in software development. While SCM helps manage the evolution of software artifacts and their documentation, traceability helps manage knowledge about the process of the development of software artifacts. In this paper, we present the integration of traceability and SCM to help change management during the development and evolution of software artifacts. We developed a traceability model using a case study conducted in a software development organization. This model represents knowledge elements that are essential to comprehensively manage changes tracked within the change management function of SCM tools. A tool that supports the integrated practice of SCM and traceability is also presented. We illustrate the usefulness of our model and tool using a change management scenario that was drawn from our case study. We also present a qualitative study towards empirically evaluating the usefulness of our approach.

1. Introduction

Software configuration management (SCM) and traceability are two important practices in software development that support system evolution and change management. Though their overall objectives remain interrelated, their implementation varies widely. SCM is the discipline of controlling the evolution of complex software systems, helping manage changes to artifacts and ensuring correctness and consistency of systems [7,13,18]. It includes both technical and administrative practices to "establish and maintain the integrity of work products using configuration identification, configuration control, configuration status accounting, and configuration audits. [31] (p. 114)" Many SCM tools provide product support (e.g., version control and system components management), tool support (e.g., workspace management, concurrent engineering control and building executable programs) and process support (e.g. defining and controlling change process) [12,13,34].

In our study, we focus on a fundamental functionality of SCM, i.e., supporting change management in system evolution. To implement a change request, SCM helps identify necessary changes, understand the impact of changes, and provides a facility to track the changes [13,19]. In comparison, Traceability is used to describe and follow the life of any conceptual or physical artifact developed during the software development life cycle [29]. It helps developers understand the relationships that exist within and across software artifacts such as requirements, design, and source code modules and analyze the impact of changes made to these artifacts during system development and maintenance [29].

SCM helps in the management, control, and execution of change and evolution of systems, while traceability helps in managing dependencies among related artifacts across and within different phases of the development lifecycle. In vast majority of organizations, these two practices are implemented in isolation. Such a practice results in fragmentation of

* Corresponding author.
E-mail addresses: kannan_mohan@baruch.cuny.edu (K. Mohan), peng.xu@umb.edu (P. Xu), lcao@odu.edu (L. Cao), bramesh@cis.gsu.edu (B. Ramesh).

0167-9236/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.dss.2008.03.003

knowledge across the two environments, and negatively impacts change management.

In general, the objective of different change management practices is to ensure a systematic development process, so that at all times the system is in a well-defined state with accurate specifications and verifiable quality attributes. To reach these goals, knowledge about both the artifacts of the software system and the process of their development and evolution must be managed [9,20]. We characterize these two types of knowledge as product and process knowledge. While product knowledge refers to knowledge about artifacts of the software system (models, specifications, documents, versions of these artifacts, etc.), process knowledge refers to knowledge about the process followed in the development of these artifacts (for example, how a requirement is implemented in the design model or code and why a design decision was made). Components of process knowledge include:

- Design decisions: This includes reasons that explain why system components are designed in a specific way (and in general, rationale behind design decisions).
- Dependencies among artifacts: This refers to how changes in some artifacts may impact other artifacts.

While most SCM tools provide adequate support for managing product knowledge (as they typically focus on version control), they do not provide adequate support for managing process knowledge that is generated and used during the development and evolution of software artifacts. The lack of process knowledge makes it difficult to understand different viewpoints held by various stakeholders involved in design process and estimate the impact of changes, thus hindering change management and adversely affecting the consistency and integrity of systems [29]. Whereas version control tools are commonly used to manage dependencies between versions of source code, dependencies among the variety of other artifacts such as requirements, design models, source code, and test cases are not adequately managed. In fact, it is the presence of these dependencies that makes change management very complex. For example, a decision to change a user requirement may lead to multiple modifications in design models, source code, and test cases. Without the capability to acquire and use process knowledge about how these artifacts are related, it is very difficult to incorporate modifications in the system. Therefore, change management function of SCM should not only help manage changes to products of systems development (product knowledge), but also help trace the effects of the changes on other artifacts (dependencies) and the reasons behind such changes (e.g. rationale) to maintain consistency among the various artifacts. Recognizing the importance of traceability in change management, past research has attempted to link certain aspects of traceability with SCM [9,23–25,37]. For example, De Lucia et al. [9] and Ying et al. [37] propose algorithms to recover traceability links from SCM tools. However, recovery tools still rely on significant amount of manual work to assess and tune the traceability links. It is also difficulty to ensure the accuracy and correctness of these traceability links [23]. Nguyen et al. [24], Nistor et al. [25] and Maletic et al. [23] propose tools that can maintain traceability links between different versions of system architectures and code to ensure that right versions of the architectures map onto the right versions of the code in SCM environments. These studies advocate integrating traceability with change management functions of SCM, but mainly address the traceability between design models and code. They do not address the maintenance of traceability between other artifacts (e.g., requirements, implementation components, test cases, etc.). More importantly, design decisions, an important component of process knowledge, are also often left undocumented. Furthermore, most of the current studies can only manage dependencies between changes and products at the level of documents or files (e.g., architecture description documents to implementation files). They are not capable of representing finer-grained dependencies such as the dependency between a specific requirement and a class in an object oriented design model. Fine-grained management of dependencies is necessary to effectively support change management since it helps developers to quickly identify and implement changes.

In summary, results from prior research [29] highlight the strong link between SCM and traceability practices. We conclude that current change management in SCM practice faces two challenges:

- *Process knowledge support:* Lack of support for managing process knowledge that can trace dependencies among artifacts across different phases of development lifecycle.
- *Granularity:* Lack of support for the management of product and process knowledge at a fine-grained level of granularity.

We argue that these challenges constrain the ability of development personnel to comprehensively understand the existing artifacts and relationships among them, and therefore affect the quality of change management. This motivates our research question: "How can the change management function of SCM be enhanced with process knowledge about artifacts developed in the software development lifecycle?" In this paper, we address this research question by proposing an approach which augments SCM with traceability. SCM comprises of a set of processes that are typically implemented through a set of tools, the most common among which are version control tools. In this research, we illustrate our approach through the integration of traceability with a version control system. However, our approach can be generalized to other practices and tools used for change management in SCM.

The paper is organized as the follows: In Section 2, we discuss traceability and its isolation from SCM, highlighting the knowledge fragmentation across SCM and traceability environments. It also presents the literature on knowledge integration as the theoretical basis for integrating knowledge across SCM and traceability. Section 3 presents our research methodology. Sections 4 and 5 present the two components of our approach (a model and a tool) to augmenting SCM with traceability. We then illustrate the application of our model and tool with scenarios drawn from our case study (Section 6). Section 7 presents the contributions and implications of our research.

2. Theoretical basis for integrating traceability and SCM

The development of large-scale, complex software has been widely recognized as a knowledge intensive activity.