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

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.

© 2008 Elsevier B.V. All rights reserved.

Keywords: Software configuration management Traceability Change management Process knowledge

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
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 [9,20,29]. 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 difficult 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.
Knowledge elements that are needed for shaping crucial design decisions exist as chunks scattered in various development environments. Integration of such distributed knowledge elements is considered as a key to successful software development [35]. In distributed software development, which is very commonplace today, knowledge integration becomes especially challenging when the team members are involved in the co-construction of ‘collective work products’ [15,22]. Our research is based on the premise that while SCM assists in the management, control, and execution of change and evolution of systems, augmenting them with traceability which helps maintain the dependencies among related artifacts across and within different phases of development lifecycle (e.g., the dependency among requirements, design elements, source code, etc) will help achieve more effective knowledge integration.

In this section, we present past research on the roles of traceability and SCM in supporting knowledge integration in software development.

2.1. Traceability

Traceability assists in understanding the relationships that exist within and across software artifacts like requirements, design, and source code modules. These relationships help designers ascertain whether and how the design and implementation satisfy the requirements. Also, traceability helps understand the rationale behind the design decisions made during system development. Traceability has been considered as a quality attribute and many standards governing systems development require the creation of traceability documents [29]. The need for maintaining traces among artifacts to support change management in software development is well documented in the literature [29]. Prior literature also describes the adverse impact of poor traceability practices on project cost and schedule [10]. Decrease in system quality, increase in the number of changes, loss of knowledge due to turnover, erroneous decisions, misunderstanding, and miscommunication are some of the common problems that arise due to lack of or insufficient traceability knowledge [10].

A traceability system helps in maintaining a semantic network in which nodes represent objects among which traceability is established through links of different types and strengths. The simplest traceability tools are purely relational (i.e. in the form of relational databases or spreadsheets) and do not systematically distinguish different node and link types. They are suited only to support simple traceability practices and provide limited support for dependency analysis. A few tools allow the user to specialize link types but it is hard to attach semantics to them. Others offer a fixed high-end set of link types with hardcoded semantics but tend to force the user to actually supply this detailed information in all cases, even if a simpler model would suffice. Prior research [29] establishes the need to develop situation-specific traceability schemes that suit the unique goals and requirements. In our research, this entails the development of a traceability scheme that helps integrate process and product knowledge that is fragmented across different artifacts managed by SCM and traceability environments.

2.2. Knowledge integration across SCM and traceability

Prior research has highlighted the relationship between the integration of existing knowledge to provide a holistic picture and the quality of problem solving and decision making [1]. For example, several studies in software comprehension have focused on assessing the impact of developer’s knowledge on understanding a system [5,26]. Links between the nature of knowledge required to understand the system and the performance of tasks by programmers have been well established [4]. Prior research on knowledge integration focuses on several contexts, viz., organizational knowledge integration, knowledge integration for new product development, knowledge integration in learning, etc. Across these contexts, knowledge integration has been defined as the synthesis of individuals’ specialized knowledge into situation-specific systemic knowledge [1]. It is also referred to as the pooling and recombination of individuals’ tacit knowledge to create group level knowledge [17]. Here, in the context of change management practice, such individual, specialized knowledge is fragmented primarily across SCM and traceability environments (as product and process knowledge). The importance of these two dimensions of knowledge—product and process—has been well established [14]. Also, while past research has examined the different contributors of quality in software development [16], knowledge integration has gained little attention in this context. Hence, drawing from the literature on knowledge integration, we argue that the integration of product and process knowledge will improve developers’ understanding of the system and thereby improve the change management process. As summarized in the research framework in Fig. 1, we argue that such a synthesis of specialized knowledge will have a positive impact on the change management process. The development of our approach to traceability-based augmentation of SCM is theoretically guided by this research framework.

3. Research methodology

Our research was conducted in three phases:

- Phase 1: Development of a traceability model that guides the integration of traceability and SCM practices.
- Phase 2: Development of a traceability tool that supports the use of the traceability model and is integrated with an SCM tool.

![Fig. 1. Research framework.](image-url)
Table 1 summarizes the steps and sources for each of the three phases of this research.

The development of the traceability model (phase 1) involves two steps: (1) Use of a meta-model from the literature on traceability, and (2) development of a detailed model that specializes the meta-model based on the findings from a case study. The development of the traceability tool (phase 2) is based on requirements drawn from the case study. Scenarios from the case study are used to illustrate the usefulness of our approach and a qualitative evaluation of our approach was conducted to examine the feasibility and usefulness of our approach as well as to identify areas in need of further study (phase 3). In the following sections, we discuss the case study and the three phases of this research.

We conducted a case study in an organization (referred to as HospCom) that develops telecommunication and television management systems for hospitals (the system is referred to as HospSys). The objective of the case study was to understand the challenges faced by stakeholders involved in the development process and thereby inform the development of the traceability model and the integrated SCM and traceability system.

### 3.1. Case description

HospSys is a billing system for patients who use a telephone and a television in their rooms. The patient usually operates the television by dialing preset codes from his/her telephone. These requests are routed to a HospSys server through a Private Automatic Branch Exchange (PABX). Services running on the HospSys server evaluate the patients’ requests by checking patient privileges and account balances. Authorized requests are sent to the PABX or the TV controller connected to the television sets. Patients who have an adequate balance in their accounts can watch pay channels and make long distance calls. HospSys controls the connection of calls and the operation of the televisions. The system was designed to work with one specific type of telephone system. When different hospital clients started using different types of telephones in patient rooms, the telephone management system had to be changed to accommodate the new types of phones. Due to the changes in the nature of messages received from the telephones, the television management system which was accessed by patients through telephones also required changes. The system was being developed to fulfill requirements of a client located offshore. The HospSys team had continuous interactions with the offshore coordinator from the client site. Rational Unified Process (RUP)® was followed in the development of HospSys. At the outset, the project manager selected MS Visual SourceSafe® as the version control tool. HospCom was a CMM level 3 certified organization and followed well documented procedures. Project management plan, SCM plan, verification and validation plan, etc., were created from the templates provided by the quality assurance department. During the execution of the project, two major challenges were faced by the team engaged in change management. These correspond to the two challenges presented in Section 1, viz., lack of process knowledge and lack of product and process knowledge at fine-grained level. The development of our traceability model and the traceability tool are informed by the challenges faced and the knowledge needs identified through this case study. The traceability model that is developed in this study presents an abstraction of best practices in integrating SCM and traceability. Analogous to the Ramesh and Jarke’s [29] argument that ‘reference models are condensed from case studies’ and are subject to continuous adaptation, refinement, and evaluation, we submit that the traceability reference model presented in this paper which helps integrate SCM and traceability is not categorically, ‘provably correct’, but it derives its relevance from the slice of practice it represents. Since the model is developed by studying the knowledge needs of an organization that follows mature development practices, we argue that the model represents ‘best practices’. Also, it should be noted that this model may not be applicable in every context/situation/project. It needs to be continuously refined and tailored according to project-specific needs. Such customization of traceability models has been advocated by past research on traceability [10].

### 3.2. Data collection and analysis

Our study was conducted over a period of about 2 years, during which structured and semi-structured interviews were conducted multiple times with five developers and two project managers. These participants were drawn from the team that was responsible for the development of HospSys. The traceability and change management practices of the development team were also observed as it went through several phases of the development lifecycle. The interviews were recorded and transcribed when feasible or detailed short-hand notes taken during the interviews were expanded in full text immediately after the interviews. Observations were also documented as notes by one of the authors.

Transcribed and documented data was analyzed using open, axial, and selective coding techniques that are commonly used in qualitative research [32]. In open coding, we focused on breaking down, examining, comparing, conceptualizing and categorizing data on change management practice [32]. It was aimed at revealing essential ideas about change management found in the data. Labeling and discovering categories are the two tasks involved in open coding. In labeling, discrete events and ideas receive a label. Categories were discovered by finding related phenomena or common concepts or themes in the data. These themes were

<table>
<thead>
<tr>
<th>Phase</th>
<th>Steps</th>
<th>Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Development of a traceability model</td>
<td>Meta-model</td>
<td>Traceability literature</td>
</tr>
<tr>
<td>2. Development of an integrated tool</td>
<td>Detailed model</td>
<td>Case study</td>
</tr>
<tr>
<td>3. Preliminary evaluation of our approach</td>
<td>Illustration</td>
<td>Requirements for the tool were drawn from our case study</td>
</tr>
<tr>
<td></td>
<td>Qualitative evaluation</td>
<td>Through examples drawn from our case study</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Qualitative data from practicing software developers</td>
</tr>
</tbody>
</table>
grouped into common headings thereby leading to categories and sub-categories. Coding was done by making comparisons of events, quotes, and instances. Categories and sub-categories corresponding to challenges faced in SCM practices and knowledge needs for effective change management were identified. Axial coding focused on further developing the categories and their properties, and establishing relationships among the different knowledge elements. Data was put back together in new ways by making connections between codes to form categories. Selective coding was used to integrate the categories to form a “theoretical framework”—i.e., the traceability model presented in Section 4. Data collection and coding were done till ‘theoretical saturation’ was reached, when additional data does not add to the concepts and categories developed [11].

During our data collection, project managers were involved in communication with coordinators from an offshore location to understand the requirements for the HospSys system. As the development process continued, changes in requirements were carefully evaluated by the relevant stakeholders, including the offshore coordinator. As the analysis and design phases proceeded, each deliverable developed by the HospSys team was shared with the client team for approval. Interactions between the HospSys team members and the offshore coordinator were observed and used in our analyses. While the change management process facilitated the documentation of changes requested from the offshore team, the coarse level of granularity at which these changes were documented resulted in several problems. Team meetings, during which decisions related to changes were discussed, revisited the decisions repeatedly because specific details were not documented adequately. While the HospSys team did maintain a traceability matrix that helped it link requirements to design documents and code modules, these links were again at a coarse level of granularity. Thus, when processing a change request, developers had difficulty in locating specific design decisions that had been made prior to the current change, and in locating specific versions of design elements that were affected by the change. By examining and analyzing the practices followed at HospSys, we developed a traceability model that represents the process knowledge required in change management. The traceability model was developed by specializing a traceability meta-model developed by past research [29]. Specifically, we specialized the primitives from this meta-model (described in Section 4),—viz., objects, sources, stakeholders, and links among them—to include specific elements that will facilitate the integration of SCM and traceability. In the following sections, we present details on the three phases of our research: (1) traceability model development, (2) tool development, and (3) illustration of our approach using a scenario and a qualitative evaluation.

4. Phase 1: development of a traceability model

A traceability model identifies physical and conceptual elements and artifacts, and the links among them. Prior research suggests that to be most useful, traceability models should be tailored to suit to specific project environments [10]. Though much work has been done in developing environment-specific traceability models, there has been a paucity of research on tailoring traceability approaches to facilitate integration with software configuration management in spite of their synergistic functions in change management.

Prior research suggests that a traceability repository will comprise at least three layers [3,8,27]:

- A meta-model defining the language in which traceability models can be defined.
- A set of reference traceability models which can be customized within the scope defined by the meta-model.
- A (possibly distributed) database of actual traces, recorded under the chosen models.

In this research, we adopt a traceability meta-model that was developed based on an extensive empirical study of software development projects [29]. We, then, develop a reference traceability model that is customized for integration of traceability and SCM. The primitives in this reference model were developed on the basis of our case study presented in Section 3. In Section 6, we present an example of how actual traces may be recorded using this reference model to illustrate how fine-grained knowledge integration can be achieved by augmenting SCM with traceability.

4.1. Traceability meta-model

The traceability meta-model is presented in Fig. 2. This model provides the basic language with primitives for categorizing and describing traceability models in more detail.

Each entity and link in the meta-model can be specialized and instantiated to create organization or project specific traceability models. OBJECTS (such as requirements, designs, system components, rationale, etc.) represent the inputs and outputs in system development. STAKEHOLDERS are the agents such as project managers, systems analysts, designer, etc. who act in different roles in the development and use of OBJECTS and traceability links. OBJECTS are documented by SOURCES, which may be physical media, such as documents or intangible things, such as references to stakeholders’ tacit knowledge. Fig. 3 shows the meta-model (shown at the top) along with a detailed model (shown below the meta-model)
depicting how traceability augments SCM. The detailed model represents different spaces involved in change management, viz., product, version, and rationale spaces. Product objects are expanded in the product space of the detailed model. Different product objects are documented in various types of sources. For instance, requirements may be documented in a requirement specification document. Different types of objects that are represented in the product space include requirement specifications, design models, system components, change requests, etc. Typically version control tools maintain only software objects such as code files. However, to ensure consistency of changes, it is also important to manage the dependencies among different software objects such as those represented in the product space. Configuration management focuses on managing the evolution of various documents, while traceability focuses on managing links or dependencies among product, process objects, and stakeholders. The figure also shows how the focus of traceability
and SCM are mapped to the different spaces in the model—SCM is mapped to version space and traceability is mapped to product and rationale space. The primitives in these spaces and the links among them emerged through the analysis of the challenges faced and the practices followed by developers at HospCom. Though HospCom followed mature software development practices and had developed procedures for documenting change requests, the participants identified two sets of challenges in their practice which correspond to those identified in Section 1.

4.2. Traceability reference model

4.2.1. Process knowledge support

When developers need to process change requests, they typically need to understand the ramifications of the request in terms of the number of different artifacts that would be affected and details about past design decisions that may be relevant for the change request. In essence, developers need process knowledge support to effectively manage changes during the software development process. Change management practices at HospCom did not provide the necessary structure that guided the effective documentation and use of process knowledge. Identifying specific primitives and links among them that represent process knowledge is the first step in guiding developers toward documenting and using process knowledge. Here, we identify specific knowledge elements in the traceability model that provide such guidance to developers.

We observed from our case study that developers were typically unable to link details about change requests to specific versions of artifacts that were related to the change request. For example, the development team faced challenges while processing the request to support different types of telephone systems. Though their version control system managed the different versions of the subsystem modules that handle communication with telephone system, the dependencies between these subsystems were not documented in detail. While their version control system managed versions of specific artifacts and documented changes that resulted in evolution of these artifacts, dependencies across different versions of various artifacts were either undocumented or fragmented in different tools and documents. While the changes done to artifacts managed within the version control system were documented, the reasons for design decisions that resulted in such changes were not explicated in a way that would help developers understand the ramifications of newer changes on the evolved artifacts. Change requests, specific changes in software artifacts that result in newer versions of these artifacts, the configurations that include these artifacts, and the rationale behind such changes needed to be documented and linked. This is represented in the traceability model with the following primitives: ‘change’, ‘version’, and ‘configuration’ in the version space, linked to ‘change request’ in the product space, and to appropriate elements in the rationale space. To understand the impact of changes, developers required detailed knowledge of the system and prior design decisions along with the reasons behind these design decisions. This can be achieved by documenting the rationale using a rationale space and linking the rationale space to appropriate artifacts and their evolution.

Linking product, version, and rationale spaces can be valuable in detecting the ripple effects of changes. Changes may affect different aspects of the system including functions, structures, behaviors, and performance. The traceability model can be used for impact analysis, defined as the activity of identifying the artifacts that must be modified to accomplish a change [2].

4.2.2. Granularity

Although the evolution of artifacts was documented and managed through a version control system at HospCom, much of this was done at the file level. Changes and dependencies were not traceable to knowledge elements documented within specific files—for example, to a class or a use case in a UML model. Therefore, the development team spent a lot of effort identifying the specific locations in the telephone subsystem where design/code had to be changed. This can be attributed to the coarse level of granularity of the model that guided their change management practice. Linking elements of specific versions of artifacts to requirements, change requests, and other related artifacts will help developers address this problem. This is facilitated by linking specific primitives in the product and version spaces. Every time a developer had to make changes to a specific part of a subsystem, s/he had to get an approval from a manager who had the higher level view and a tacit understanding of the dependencies across subsystems. However, this process did not work very well, as lower level implementation details needed a thorough understanding of the dependencies. This is addressed by linking low level product objects with requirements and change requests. For instance, while version space in our model specifies versions, changes that are done to move from one version to another, and configurations, each software object in product space is related to a set of versions in version space. Such a software object can be specified at any level of granularity ranging from a class or a use case to a subsystem or a module. Changes performed to fulfill change requests result in different versions of specific software objects. The knowledge about these dependencies is represented at a fine-grained level by linking specific software objects or design elements to the versions of those objects managed in the version control system.

Rationale, which is one type of process object, is also required at a fine-grained level to provide developers a clear understanding of design decisions made. In this space, reasoning behind decisions can be represented using models such as IBIS [6] and ReMap [28,29]. For example, in the reference model for rationale developed in prior research [29], ‘issues’ represent problems that need to be addressed. ‘Alternatives’ suggest different solutions to ‘issues’, each of which can have supporting and objecting ‘arguments’. ‘Assumptions’ are represented explicitly. ‘Decisions’ are made by evaluating ‘alternatives’. The rationale space represents deliberations related to the creation and modification of software objects, versions, and configurations described in the product and version spaces.

In summary, the lack of adequate process knowledge and the coarse level of granularity of product and process knowledge (i.e. the two challenges identified in Section 1), result in difficulties in performing impact analyses, high costs of implementing changes, and quality issues that slow down
development. We posit that the first step in addressing these two issues involves the integration of product space, version space, and rationale space, thereby providing traceability not only to different software artifacts and process knowledge associated with these artifacts, but also provides such traceability at a more fine-grained level. Our traceability model includes links between product and process knowledge and the different spaces (product, rationale, and version spaces). While product space documents product knowledge, rationale space documents process knowledge. Version space provides a platform for managing product and process knowledge, in that it helps manage versions of the products specified in the product space while facilitating links to finer details about changes, versions, and configurations (for example, links to detailed decisions, issues, etc. that are related to specific changes, versions, and configurations). Our traceability model depicts how product-oriented knowledge may be augmented to represent process-oriented knowledge typically represented in traceability tools, thereby providing a structure for process knowledge support at a fine level of granularity.

5. Phase 2: development of a traceability tool

Based on our case study, we have developed a traceability tool (called Tracer) that supports the acquisition and use of process and product knowledge represented in model shown in Fig. 3. This tool supports the creation of a traceability network representing the association among various knowledge components drawn from different tools used by software developers. Tracer is integrated with MS Visual SourceSafe®, a commonly used version control tool. While the ultimate objective of our research is to integrate change management practices of SCM and traceability, SCM as a set of processes is typically implemented through a set of tools, the most common among which are the version control tools. We illustrate our approach through the integration of traceability with a version control system. However, our approach generalizes to other aspects of change management in SCM.

5.1. Requirements for the tool

When probed about specific requirements for an ideal tool to address the issues faced by the organization in managing product and process knowledge, the stakeholders identified the following three specific requirements:

1. Support for a comprehensive traceability scheme that links product and process knowledge (such as the model developed in phase 1): The tool should facilitate the definition and instantiation of a user-defined traceability model.
2. Integration with an SCM tool: The tool should be able to communicate with an SCM tool so that links can be established across artifacts managed by the tools that implement SCM and the knowledge elements documented in the traceability tool.
3. Linking change requests and versions that satisfy these requests to specific parts of artifacts that are impacted: The tool should facilitate linking process knowledge elements to specific parts of artifacts (such as use cases, classes, etc.) rather than just files.

Table 2 shows the mapping between the two issues presented in Section 1 and the capabilities of our tool that address these issues.

5.2. Architecture of the tool

Fig. 4 shows the architecture of the Tracer. Each layer shown in the architecture is described in the following sections.

5.2.1. User interface layer

Tracer’s user interface layer is used to define a traceability model such as the one shown in Fig. 3. At the beginning of each project, the project manager can use the “Model Editor” to develop a traceability model that suits the needs of the project by specifying the primitives in the model. During the development life cycle, developers use the “Traceability User Interface” to create traceability links that instantiate this model. In this process, they also use the “SCM Communication Interface” to connect to the version control tool (MS Visual SourceSafe®) to link specific versions of artifacts to process knowledge elements such as design decisions and rationale.

5.2.2. Repository layer

The “Traceability Model” that is used as the schema with which knowledge elements are instantiated is stored in the “Model Instance” repository. Tracer communicates with MS

<table>
<thead>
<tr>
<th>Issues in change management</th>
<th>Requirement for traceability tool that augments SCM</th>
<th>Capability of Tracer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lack of process knowledge support</td>
<td>Supporting the traceability model developed in phase 1 Integration with an SCM tool</td>
<td>Support for a traceability model as the one shown in Fig. 3. This model can be tailored to suit different kinds of projects. Managing dependencies among knowledge fragments that reside in various work process tools used by developers. Tracer is integrated with other tools that are used in software development like MS Office Suite, Rationale Rose (a CASE tool), Groove (a collaboration tool), and MS Visual SourceSafe (a version control tool)</td>
</tr>
<tr>
<td>Lack of fine-grained management of process and product knowledge</td>
<td>Linking change requests and versions that satisfy these requests to specific parts of artifacts that are impacted</td>
<td>Tracer facilitates the documentation of changes implemented in specific elements within versions of various software artifacts and facilitates linking such changes to change requests. Links to changes can be done at a finer level of granularity—for example, a specific class or a use case can be linked to a change request to specific versions of this artifact in SourceSafe.</td>
</tr>
</tbody>
</table>
Visual SourceSafe® through the interfaces exposed by it to access the version repository. Software artifacts are organized within projects in SourceSafe®. Process knowledge (e.g., design decisions and dependencies among software artifacts) is stored in the “Model Instance Repository”. Whenever changes are required, specific artifacts that are affected are checked out into the developer’s working folder. During the check out process, typically a textual description of the purpose of the check out is created. After the changes are made, the artifacts are checked in and the changes are documented textually. Using the interfaces exposed by MS Visual SourceSafe®, Tracer can communicate with it and extract data about the change history of specific artifacts.

Specific knowledge elements in Tracer can be linked to specific versions documented in MS Visual SourceSafe.

5.2.3. Tool service layer

The tool services layer supports the following services: model and rules definition, change propagation, model conformance verification, and integration with other work process tools. Each of these services is discussed below.

5.2.3.1. Model and rules definition. As discussed in Section 4, our approach requires the definition of a traceability model and rules that help maintain the semantics of the elements in the model. Model definition involves the definition of the primitives of the traceability model and permissible links among them. Here, we adopt the model shown in Fig. 3.

5.2.3.2. Change propagation. The definition of rules using the format described above helps in change propagation. Any change to a software artifact can be propagated along the dependency paths to suggest possible impacts on dependent artifacts. While simple impact analysis can be facilitated by traceability matrices which represent relationships between requirements and the design elements that satisfy them, complex and detailed analysis of the impact of changes requires the use of more sophisticated inferencing procedures such as the rules defined as part of the traceability model. In Tracer, visual identification of the impact of changes is enabled using a graphical browser. Apart from just identifying elements that are impacted by changes, the nature of the impact is described by the rules that specify the semantics of links. Link semantics explain how a change affects related elements. Rationale that is documented as part of the traceability knowledge also helps explain the nature of the impact of changes.

5.2.3.3. Model conformance verification. While documenting changes, Tracer prevents stakeholders from creating stray knowledge elements and links among them that are not identified by the model. As new knowledge elements and links are created, Tracer verifies the conformance of these instances with the pre-defined model schema (stored in “Traceability Model” repository). In fact, the graphical interface allows the instantiation of only the objects and links that are specified in the traceability model.

5.2.3.4. Integration with SCM and other tools. Tracer has the capability to communicate with a host of work productivity and work process tools. The integration with MS Visual SourceSafe®, a version control tool, is central to the approach discussed here. Using Application Programming Interfaces (API) exposed by MS Visual SourceSafe®, Tracer communicates with this tool to extract knowledge about various versions that form different configurations and the history of changes done to the artifacts documented in it. Any knowledge element in Tracer can be linked to a specific version of a particular artifact by opening MS Visual SourceSafe® from within Tracer and...
selecting the element that is to be linked. This functionality is explained in more detail in the context of our case study in the next section. Tracer is also integrated with other software development tools including Rational Rose®, a popular CASE tool, work productivity tools included in the MS Office Suite®, communication and collaboration tools like NetMeeting and Groove, and MS Project®, a project management tool. Macros and APIs are used to enable such integration. Such an integration of Tracer with other tools used in the software development process enables developers to retrieve relevant knowledge from various tools more easily at a fine-grained level of detail.

6. Phase 3: preliminary evaluation

In this section, we describe the two steps used to provide a preliminary evaluation of our approach. First, we use a scenario drawn from our case study to illustrate the usefulness of our approach. Second, we present a qualitative study involving software developers on the perceived benefits of our approach.

6.1. Illustration through an example from our case study

At the outset of the project, when traceability and change management planning is done, the traceability model (as specified in Fig. 3) is defined by the project manager using the Model editor. Users can define the names of the nodes, the properties of the links including their semantics, and change propagation properties such as the direction of the links and the nature of impacts of changes.

The top panel (toolbar) in Fig. 5 shows all the primitives in our traceability model. The canvas shows the instantiation of the traceability model for a scenario involving changes to the telephone system in HospSys.

This scenario that is documented in the canvas as shown in Fig. 5 relates to a change request for adding a new telephone system. Fig. 5 shows that the design rationale and documented dependency of the telephone system indicates how the telephone and television subsystems are related with respect to messaging that happens between the two subsystems.

Let us consider this scenario from the point of view of the developer who is responsible for implementing the changes to support the new type of telephone system in terms of the two challenges, viz., process knowledge support and granularity.

6.1.1. Process knowledge support

First, the developer needs to understand the change that needs to be incorporated and the existing artifacts that may be related to the change. The traceability network shown in Fig. 5 already documents the details about a change incorporated in the telephone system. This network was created during the initial development of HospSys. The figure shows that the first dependency is the telephone message interpretation feature which is implemented in the telephone message interpretation module. The figure also shows a particular version (v-1-2) as the

![Fig. 5. Instantiating the traceability model.](image-url)
candidate for change. This information is critical, as the latest version of the system may not be the appropriate candidate for change. In fact, later versions of this subsystem had been modified in such a manner that some flexibility in supporting different telephone systems was lost. The developer can further drill down on this knowledge to specifically identify where this version of the module is located. With the integration of Tracer with MS Visual SourceSafe®, the developer is able to view the properties of the version node of the module and access it in the version control tool without exiting Tracer. The link between the specific version of this module in the version control tool and the version node in Tracer was set at the time of initial development.

Fig. 6 shows this integration between Tracer and MS Visual SourceSafe®. Different versions that are involved in the scenario are shown in the dependency network in Fig. 6. These version nodes are linked to the actual source versions managed in SourceSafe. For example, the instance of the ‘version’ node ‘v-1-2’ of the ‘telephone message interpretation module’ (a ‘component’ node) in Fig. 6 is linked to that specific version of that component in MS Visual SourceSafe®. Versioned items here can be linked to specific knowledge elements (for example, rationale elements) in the traceability model, which explain why certain versions were changed. After identifying the implementation details of the telephone message interpretation system, the developer now turns her attention to other dependencies. The television message interpretation module is documented in the traceability network as a dependency of telephone message interpretation module. This dependency is caused by the need for the two systems to communicate with each another (for example, when patients attempt to change a television channel, they use their telephone. These signals have to be sent from the telephone message interpretation module to the television message interpretation module). Therefore, when a new telephone type is introduced, due to the changes in the messaging process, television subsystem should also be changed. The view in Fig. 6 shows the specific versions of the specific parts of the subsystems that need to be changed. Now the developer has a significant amount of knowledge about the change she is about to implement. This will help her make informed decisions about what to change and how to implement the changes. The traceability network also captures the rationale behind past design decisions (not shown in the figure). This knowledge will help the developer avoid repeating any mistakes from the past and make appropriate design decisions.

6.1.2. Granularity

The developer is able to further examine the knowledge required for this change request at the level of specific classes in the UML model for HospSys that are affected by this change.
change. The telephone interpretation module node in the Tracer is linked to the classes that implement it. During initial development, using the integration of Tracer with Rational Rose®, links had been established between this node in Tracer and the classes that implement this module. Now, the developer can use these links and invoke Rational Rose® from within Tracer. Tracer is capable of showing a tree-like view of the elements available within HospSys UML model. All the use cases, classes, components, etc., are listed in this view. The developer can navigate directly to Rational Rose® to the location where the dependent elements are specified (for example, to the logical view containing the classes that need to be changed). These classes can be linked to a requirement stored in text format in a MS Word® document. Therefore, the developer can trace the life of this change from the requirements (in a word document) to source code versions (in a version control tool). This feature improves the level of granularity from the CASE tool) to source code versions (in a version control tool).

The process knowledge that was described above is stored in the “Model Instance and Repository”. This process knowledge helps developers understand the current design and the rationale behind dependencies. While making decisions about “adding new types of telephone systems” to satisfy the new requirement request, this knowledge helps her identify other possible design modules that are affected (i.e., telephone message interpretation and hence television message interpretation). To maintain the integrity and consistency of the system, developers need to update the process knowledge and document current changes and rationale behind these changes that are made to implement the new telephone system.

Without access to this process knowledge, the developer relies on her own expertise and past experience with the system. Since she may not possess adequate knowledge about all the specific dependencies, she may often make inconsistent and incomplete changes. The approach proposed here helps avoid such expensive mistakes.

6.1.3. Summary

The traceability system used in conjunction with the traceability model, addresses two challenges identified in Section 1:

1. It provides process knowledge support. Acting as a repository of process knowledge, it helps developers trace decisions and dependencies among artifacts in change management. The capability to link different types of artifacts developed at various stages is critical since these decisions transcend different artifacts developed during different phases of the software development life cycle.

2. It provides the ability to manage product and process knowledge at a fine-grained level of granularity. It provides the capability to link a particular requirement to a specific class and/or a method and/or a use case so that when a developer implements a change request, he/she can identify specific elements that are relevant for the change.

In summary, our approach to integrate SCM with traceability addresses the challenges currently faced in the change management process. Our implementation illustrates the benefits of this integration.

6.2. Qualitative study

We conducted a qualitative evaluation of our approach by seeking feedback from four experienced practitioners. This qualitative evaluation was done independent of the case study presented in Section 3. The capabilities of our prototype were demonstrated and discussed with the participants in this evaluation. After presenting our approach, semi-structured interviews were conducted with questions that focused on the usefulness and applicability of our approach in managing changes during software development.

Consistent with the traceability and knowledge integration theories we discussed in Section 2, the qualitative evaluation illustrates that the traceability tool used in conjunction with the traceability model provides more appropriate and valuable information required by developers during change management [33]. Specifically, our qualitative evaluation constitutes the first step in identifying the links between knowledge integration facilitated through the management of dependencies and fine-grained process and product knowledge and impact analysis accuracy and efficiency and program comprehension (Fig. 7). Here we developed several propositions that will be tested in future research.

Integrating traceability with SCM leads to better process knowledge support and provides fine-grained process and product knowledge support. Our study participants suggest that this knowledge is useful when a development team is tasked to make changes to the system. The study participants suggest that the repository of process knowledge helps developers in impact analysis during change management. In our qualitative evaluation, the capability to link requirements and design objects (such as a class/method/use case), was highlighted by the participants as helpful in improving the impact analysis process by identifying changes more efficiently and accurately. Therefore, we offer the following propositions:

**P1.** Knowledge integration across traceability and SCM will improve the accuracy of impact analysis.

**P2.** Knowledge integration across traceability and SCM will improve the efficiency of impact analysis.

Our qualitative evaluation also suggests that our approach to integrating SCM and traceability will improve program comprehension [36]. The capability to keep track of product and process knowledge will help developers better understand the various
artifacts such as design fragments and code. The capability for change propagation was found to be useful not only to assess the impact on software artifacts, but also during the process of obtaining approval for changes. In the absence of a model and tool such as ours, developers often rely on their past experience to understand rationale behind past decisions which are documented in the version control tool at a high level of detail. The participants reported that structured rationale documentation with links to appropriate software artifacts improves program comprehension (i.e. understanding of the system components) in two ways—by reducing the time required and by improving accuracy. Hence we propose:

P3. Knowledge integration across traceability and SCM will enhance the accuracy of program comprehension.

P4. Knowledge integration across traceability and SCM will enhance the efficiency of program comprehension.

In summary, the qualitative study identifies four propositions that suggest that our approach to integrating SCM and traceability satisfy the objectives of this research. However, a comprehensive examination of these propositions is a subject of future research (as discussed in Section 7.4).

7. Discussion

In this section, we present the contributions of our research, implications to research and practice, limitations, and future research directions.

7.1. Contributions

This research makes the following contributions by presenting a novel approach to integrating the two critical areas of SCM and traceability in software development. Specifically, this research:

• Presents an empirically derived traceability reference model that integrates knowledge in product, rationale, and version spaces.
• Presents an approach for managing process knowledge support by integrating knowledge about artifacts fragmented across SCM and traceability environments.
• Presents an approach to manage product and process knowledge at a fine-grained level.
• Illustrates integration of version control with traceability that can be generalized to other practices and tools used for change management in SCM. For example, our model and tool can be extended to support integration with a tool used to track change requests by customizing the traceability model.

7.2. Implications to theory

This research has important implications to research on SCM and traceability. Past research on SCM and traceability has been isolated from one another. We propose the synergistic integration of these areas. For example, research on developing or improving change management may use a traceability framework to organize artifacts managed within SCM tools. Traceability model development should include version and configuration management primitives and traceability tool developers may exploit the commonly available SCM infrastructure. Research on process framework development (such as Rational Unified Process) will benefit by defining the SCM processes that are integrated with traceability practices. Such integration will be even more important in software development organizations that develop product families using domain and application engineering due to the complexities involved in managing common and variable aspects. Our research also identifies future research areas on the impact of this integration on impact analysis and program comprehension.

7.3. Implications to practice

This research contributes to practice by suggesting that software developers should consider the synergistic relationship between configuration management and traceability. Though SCM’s definition encompasses traceability, in practice, traceability and SCM have been practiced in isolation. By proposing a model that integrates SCM and traceability, we provide guidelines to project managers and developers for documenting changes and their impacts during software development. Project managers may use our unified model as a reference model. Project managers should consider using version control tools that are tightly integrated with traceability tools to improve the process of change management. CASE tool developers can benefit from this research by understanding the interoperation of version control and traceability tools. They may develop integrated suites that can handle configuration management and fine-grained traceability documentation based on the traceability model.

Specifically, our approach to integrating traceability with SCM will help in the following situations:

1. This knowledge is useful when a development team is tasked to make changes to the system. While the benefits to a team that was not involved in the original development are apparent, it should be noted that even a team that did implement the original system may benefit from this knowledge, especially in large, complex projects. The knowledge will help the team understand past decisions made and the impact of changes on software artifacts.

2. When there is employee turnover, the knowledge documented in the system will help retain at least part of the knowledge that would have been lost otherwise.

3. This knowledge can also be useful in similar design situations. Certain design decisions may have pervasive impact on the architecture of enterprise wide systems. Other teams will find this knowledge useful by generalizing and applying it in different contexts. In essence, this knowledge can lead to the development of general best practices that can be shared across projects.

4. It should be noted that the issues discussed above are not only useful to novice developers, but also to experienced stakeholders. Further, the issues addressed by our approach are common even in organizations that follow considerably mature practices [29].

7.4. Limitations and future research

Generalization of the results from this research must be done with caution in light of its limitations. This research has
investigated the issues in SCM and traceability in only one field setting. Further field studies are necessary to enhance the generalizability of our approach. Several project-related factors such as complexity, size, development environments used, etc. should be considered when applying the results of this research. The effectiveness of our approach should be validated by detailed empirical studies for various types of software development projects. The conceptual framework and the propositions developed need to be validated by future research. Traceability typically involves significant overhead in terms of effort required to document and use traceability knowledge while changing system artifacts. This overhead may be minimized by providing clear guidance on what kind of knowledge will be most valuable during later stages of the development life cycle. The traceability model presented in this research may be used as a reference model for this purpose. Also, we have used MS Visual SourceSafe as the version control tool in the implementation of our approach to integration. Further integration with more comprehensive set of tools focusing on various areas of SCM will be a subject of future research. Development of appropriate incentive schemes to motivate software developers to acquire and use traceability knowledge is a subject of future research. In general, future research can examine the various factors that affect (for instance, cultural factors [21]) developers' propensity to contribute to knowledge integration processes. Future research can also focus on addressing migration issues that may be involved in transitioning to a new approach to change management such as the one presented in this research. We are currently planning a detailed study to empirically evaluate the propositions developed in Section 6.

References


Kannan Mohan is an Assistant Professor of Computer Information Systems at Baruch College. Dr. Mohan received his Ph.D. degree in Computer Information Systems from Georgia State University. His research interests include managing software product family development, providing traceability support for systems development, knowledge integration, and agile development methodologies. His work has appeared in journals such as Communications of the ACM, Decision Support Systems, Information & Management, and Communications of the AIS.
Peng Xu is an assistant professor at the Department of Management Science and Information Systems of the University of Massachusetts Boston. She received her Ph.D. in Computer Information Systems from Georgia State University. Her research areas are software process, software engineering, knowledge management and process agility. Her work has appeared in journals such as Journal of Management Information Systems, Requirements Engineering Journal, Communication of the ACM, and Information and Management.

Lan Cao is an assistant professor of Information Technologies and Decision Sciences at Old Dominion University. She received her PHD on Computer Information Systems from Georgia State University. Her major research interests are agile software development and software process modeling and simulation. Her work has appeared in journals such as Communications of the ACM, Information System Journal, IEEE Software, Communications of the AIS and Journal of the AIS.

Balasubramaniam Ramesh is Board of Advisors Professor of Computer Information Systems at Georgia State University. Dr. Ramesh received his PhD degree in Information Systems from New York University. His research work has appeared in several leading conferences and journals including the MIS Quarterly, Journal of Management Information Systems, Journal of the AIS, IEEE Transactions on Software Engineering, Decision Support Systems, Communications of the ACM, IEEE Expert, IEEE Computer, IEEE Intelligent Systems, IEEE Internet Computing, Annals of Software Engineering, and Annals of Operations Research among others. His research interests include supporting complex organizational processes in areas such as requirements engineering and traceability in systems development, concurrent engineering, NPD, knowledge management, and business process redesign.