View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by AIS Electronic Library (AISeL)

Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2005 Proceedings

Americas Conference on Information Systems (AMCIS)

2005

Provenance in Software Engineering - A Configuration Management View

Peng Xu
University of Massachusetts Boston, peng.xu@umb.edu

Arijit Sengupta
Wright State University, arijit.sengupta@wright.edu

Follow this and additional works at: http://aisel.aisnet.org/amcis2005

Recommended Citation
Xu, Peng and Sengupta, Arijit, "Provenance in Software Engineering - A Configuration Management View" (2005). *AMCIS 2005 Proceedings*. 515.
http://aisel.aisnet.org/amcis2005/515

This material is brought to you by the Americas Conference on Information Systems (AMCIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in AMCIS 2005 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.

Provenance in Software Engineering – A Configuration Management View

Peng Xu
Management Science and Information Systems
University of Massachusetts, Boston
peng.xu@umb.edu

Arijit Sengupta
Information Systems and Operations Management
Wright State University
arijit.sengupta@wright.edu

ABSTRACT

Information provenance is a mechanism for tracing and verifying sources of information. In software development, provenance can be seen in two dimensions: (a) traceability among different versions of the same artifact and (b) traceability among various artifacts across system lifecycle. Maintaining the provenance, including the history of changes and the rationale of changes, are critical in assessing change requests, identifying of appropriate products/builds, and ensuring configuration integrity. Although some Configuration Management (CM) tools support a form of provenance by keeping logs of changes, such logs are proprietary and cannot be migrated to other systems if needed. In this research, we demonstrate how provenance can be achieved in configuration management by binding an artifact to its traceability and evolution information and storing such information in XML-based metadata, so that the information can be moved along with the artifact from one CM tool to another.

Keywords: provenance, traceability, software configuration management.

INTRODUCTION

Information provenance is a mechanism for tracing and verifying sources of information. Provenance refers to the process of adding annotation to an information artifact for the purpose of tracing its source and modification history (Buneman, et al., 2001). Provenance is the process of answering questions such as where a piece of data came from and what operations have been performed on the data. It is critical to maintain the accuracy and currency of data.

In software engineering, information provenance is a key factor in Configuration Management (CM). The Capability Maturity Model (CMM) defines CM's purpose as to "establish and maintain the integrity of the products of the software project throughout the project's software lifecycle" (Paulk, et al., 1993). In order to achieve its goals, CM needs to manage the traceability among artifacts to maintain integrity. Such traceability can be seen in two dimensions: (a) traceability among different versions of the same artifact and (b) traceability among various artifacts across system lifecycle. While the first dimension focuses on tracing evolvement history and multiple variants of a particular artifact (e.g., source code), the second dimension focuses on capturing the dependency among different artifacts across software development lifecycle (e.g., the dependency among requirement, design, and code). Maintaining the provenance in CM, including the history of changes and the rationale of changes, is critical in assessing change requests, identifying appropriate products/builds, ensuring configuration integrity, and hence, ensuring provenance in software engineering[1]. Such information would allow designers and developers the capability of querying the CM system for advanced information regarding the history of not only versions, but also the interactions between versions of different documents across the lifecycle. A designer, for example, may determine how a specific requirement is handled in the design, and implement in the code, and thus, could predict how the change in a requirement will affect the design and implementation. Such queries would enable designers make intelligent decisions regarding potential alternative designs, saving production time and costs.

Different tools are available to support either the first or the second dimension of information provenance in CM. However, current tools share one weakness. Although some CM tools support a form of provenance by keeping logs of changes, such logs are proprietary and cannot be migrated to other systems if needed. This weakness hinders the ability of sharing artifacts outside one CM environment without loosing valuable information such as how and why they evolve. For example, in open

[1] RUP® Framework, IBM®.

source development, developers can read, redistribute, modify, and publish the source code for a piece of software. However, the published software loses provenance information that explains how and why a piece of software was developed and modified, which make software adoption and maintenance difficult.

In this research, we demonstrate how provenance can be achieved in configuration management by binding an artifact to its traceability and evolution information and storing such information in XML-based metadata, so that the information can be moved along with the artifact from one CM tool to another. The paper is organized as the following. Section 2 reviews relevant literature on CM, traceability, and information provenance. Section 3 describes the research model. The implementation and the architecture of the system are depicted in Section 4. Section 5 concludes the paper and discusses the future research.

LITERATURE REVIEW

Research on Software Configuration Management

Software configuration management (SCM) is the discipline of managing the evolution of software systems. The main goals of SCM include identifying product components and their versions, establishing procedures that should be followed when performing changes, maintaining the status of components and change requests, and ensuring consistency of products (IEEE/ANSI Standard 1042, 1987). CM tools have been developed to support SCM (e.g., +1CM®, CM Synergy®, Clearcase®, SourceSafe®, etc*)*.

In managing information provenance, most CM tools focus on providing version control mechanisms to keep track the variants of the same artifact. Facilities such as identifying and organizing versions and objects, retrieving existing versions, and constructing new versions are provided (Conradi and Westfechtel, 1998).

Research on Traceability

Current CM tools mainly focus on one dimension of provenance, i.e., the management on versions and variants of the same artifact. Intensive research in software engineering has also been conducted to investigate the second dimension of provenance-- traceability across software development lifecycle.

Traceability is defined as "the ability to describe and follow the life of a requirement, in both a forward and backward direction" (Gotel and Finkelstein, 1994). Traceability can facilitate software development by maintaining the relationships that exist among requirements and other artifacts, ensuring consistency among artifacts, helping ascertain how and why system development satisfy stakeholder requirements, and allowing to trace changes throughout the system (Palmer, 1997).

Traceability includes not only the dependency among artifacts but also the design rationale behind each design. It can facilitate negotiation among developers by clarifying purposes and criteria and make knowledge of how the system was designed explicit, which can improve system maintainability and artifact reusability (Dutoit and Paech, 2000).

Different approaches have been adopted to maintain traceability and DR in software development, such as using matrix (Davis, 1990), hypertext links (Kaindl, 1993), and graphic manner (Ramesh and Dhar, 1992). Commercial tools such as RequisitePro® and Doors® have been built as well to support traceability in software development.

Information Provenance

However, the common weakness of current version control tools and traceability tools have is that the dependency among artifacts and development history are not portable as artifacts themselves. They are stored in the CM environment and can be easily lost when artifacts are moved from one development environment to another. This issue is more severe when open source code and outsourcing become the trends of the system development where multiple parties are involved in software development and maintenance.

Information Provenance, a term introduced by Buneman (Buneman, et al., 2002; Buneman, et al., 2001), is a new notion of traceability. Applicable in virtually any software development domain, provenance refers to the process of adding annotation to an artifact for the purpose of tracing its source and modification history. Since annotation is bound with artifacts, it can be moved along with artifacts. This enables the user of an artifact to quickly and efficiently trace source and modification history of the information to discover and reason inconsistencies. The annotation information, if properly managed and harnessed, can be a major source of scientific discovery (Myers, et al., 2003).

In software engineering there is a vast amount of information that should be recorded and harnessed for the purpose of efficient software development, and for the purpose of improving traceability. Provenance can solve the weakness of current CM tool (i.e., losing traceability information when leaving the CM environment). Our model, presented next will address such annotation methods, with a process of provenance of such annotation.

SOFTWARE CONFIGURATION PROVENANCE (SCP) MODEL

We present a two-dimensional model for the purpose of provenance of software configuration, that we term SCP (Software Configuration Provenance). Figure 1 shows the SCP model. We posit that every software development process goes through a series of milestones, regardless of which software development life cycle (SDLC) model is used. Artifacts for each development cycle (such as task analyses, requirement specifications, design documents, actual code, as well as user and system documentations) go through several revisions during and after production. Not only the artifacts are related along the versions (vertical axis), but also along the development cycle (horizontal axis). In our model, arrows connecting along the vertical axis represent the traditional version control information, and arrows between different vertical columns represent traceability annotation.

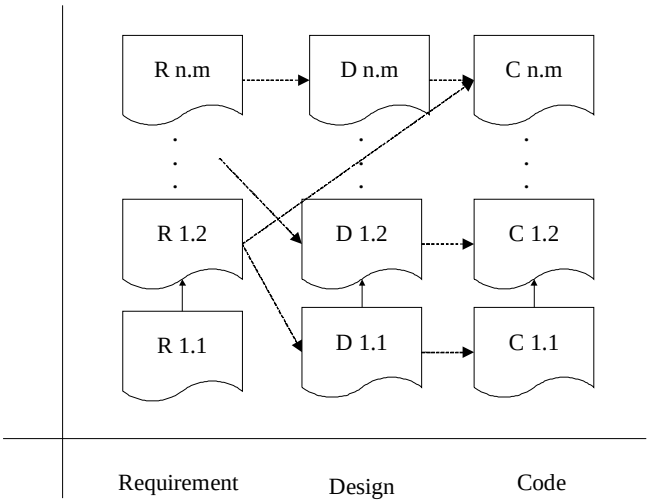

Figure 1. The SCP model

The provenance information is managed in a Fully Traceable System (FTS) where every connection is appropriately annotated. The annotation information in each artifact may include – version ID, name, type, summary of modification, rationale of the modification, relationships with other artifacts, and author names. Summary of modification can explain *what* operations were performed and *how* they were implemented in this artifact. Making the rationale of the modification explicit can help explain *why* certain operations were performed. The relationships with other artifacts explicitly document the dependency, which ensures system consistency when changes are made.

Conceptually, the provenance information provides a layer on top of a standard CM and version control system. In a most basic implementation, such information can be implemented in a set of read-only XML documents along with the other content of the project. However, a proper realization of SCP would require a more tight integration of the provenance information, described in the next section. Here we briefly describe a few scenarios where such information would be needed for proper project development and management:

1. *Open Source Development*: Open source development has heavily influenced the software industry. However, being involved in open source development requires strong understanding of the design process and the incorporation of the design in the codebase, often resulting in very high learning curve. SCP will enable queries for designers and developers of the system, even those who join the development phase late, a quick mechanism for finding necessary information for assisting in development.
2. *Outsourcing and Offshoring*: Offshoring has become a common business practice. To ensure proper deployment and control of offshored components of code, often major amount of resources are used in proper integration of

offshored code. SCP will allow a tighter integration of offshored code into the primary development cycle, and will allow quick discovery of potential inconsistencies between the design and the offshored implementation.

PROTOTYPE ARCHITECTURE

Our prototype, Fully Traceable System (FTS), is designed to enhance current CM systems by binding provenance information with the artifact.

Figure 2 shows the architecture of our system. FTS is integrated with a CM tool. "Provenance generator component" provides an environment where provenance information including traceability and design rationale for artifacts are defined. When users check in an artifact through the "version control component" of the CM tool, "provenance generator" in FTS is invoked to elicit information from users to construct the XML annotation. The information elicited from users includes version id, file type and name, brief summary of the artifact, whether this file is related to any other files in the CM tool, descriptions of the relationships, and information on changes. Provenance information is represented in the XML format (shown in Section 3) and saved separately from the artifacts that are stored in the CM tool. An XML tag is inserted into the artifact to be used to link the provenance with the artifact.

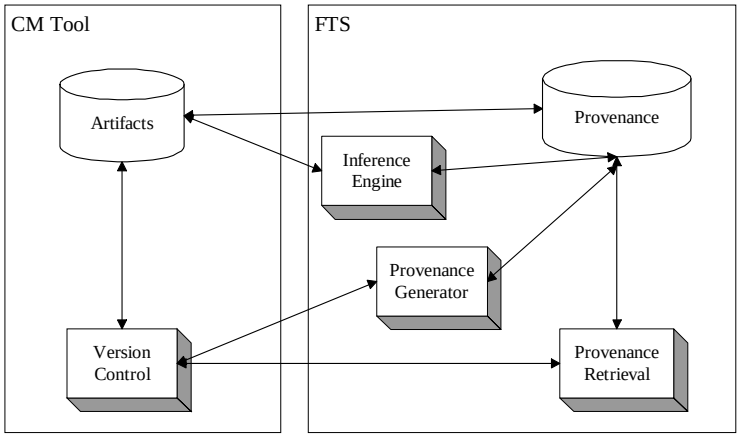

Figure 2. Architecture of the FTS

When checking out an artifact from the CM tool, "provenance retrieval component" examines the XML tag in the artifact and retrieves the relevant XML file. The provenance information is be checked out and saved as an attached folder along with the artifact. When checking in the artifact, "provenance generator" elicits provenance information on new changes and update provenance information. "Inference engine" will trace the dependency information in the XML file and suggest the impacted artifacts.

Users interacting with the system will not see the provenance information. For most part, the interaction between the user and the system will be similar to how a user interacts with a version control system. When checking out a document, the system will note the time and date of checkout, and the ID of the user. When checking in, the user would be asked not only for the changes made to the document, but also whether the changes made are in response to some changes in any other chain of the artifacts. For example, a change in the requirements may cause a change in the design, and eventually a change in the code. Such information can be subsequently used by the provenance retrieval component for traceability, version control, as well as provenance.

CONCLUSION AND FUTURE WORK

The SCP model presented in this paper provides a new method to incorporate versioning, traceability, and provenance in software design. Such information is needed for many different applications, especially where software is developed in teams, where some teams may not have control over how other teams operate. A close-knit model such as SCP will ensure that all requirements are fulfilled, and designs are properly implemented. Real world applications outside software development can also be felt in the process of outsourcing, as well as open source development. After the system is completely implanted and tested, a case study will be conducted to validate the model and the approach.

REFERENCES:

1. Buneman, P., Khanna, S., Tajima, K., and Tan, W. C. (2002) Archiving scientific data, *Proceedings of rhe ACM SIGMOD International Conference on Management of Data*.
2. Buneman, P., Khanna, S., and Tan, W. C. (2001) Why and where: A characterization of data provenance, *Proceedings of the International Conference on Database Theory*, London, UK.
3. Conradi, R., and Westfechtel, B. (1998) Version models for software configuration management, *ACM Computing Surveys*, 30, 2.
4. Davis, A.M. (1990) Software requirements: analysis and specification, Prentice-Hall.
5. Dutoit, A.H., and Paech, B. (2000) Rationale Management In Software Engineering, Handbook of Software Engineering and Knowledge Engineering, S. K. Chang (Ed.) World Scientific Publishing Company.
6. Gotel, O.C.Z., and Finkelstein, A.C.W. (1994) An analysis of requirements traceability problem, *Proceedings of the IEEE International Conference on Requirements Engineering*, Colorado.
7. IEEE/ANSI Standard 1042 (1987) IEEE Guide to Software Configuration Management.
8. Kaindl, H. (1993) The missing link in requirements engineering, *ACM SIGSOFT Software Engineering Note*, 18, 2.
9. Myers, J.D., Chappell, A., Elder, M., Geist, A., and J. Schwidder (2003) Reintegrating the research record, *Computing in Science and Engineering*, 2003.
10. Palmer, J.D. (1997) Traceability, in R. H. Thayer And M. Dorfman (Eds.), *Software Requirements Engineering*, IEEE Computer Society Press.
11. Paulk, M., Weber, C., Garcia, S., Chrissis, M.B., and Bush, M. (1993) Key Practices of The Capability Maturity Model Version 1.1, CMM.
12. Ramesh, B., and Dhar, V. (1992) Supporting systems development by capturing deliberations during requirements engineering, *IEEE Transactions on Software Engineering*, 18,6.