INTERACTION CENTRIC REQUIREMENTS TRACEABILITY

Nitesh Narayan, Yang Li, Jonas Helming and Maximilian Koegel
Institut für Informatik, Technische Universität München, Boltzmannstrasse 3, 87548 Garching, Germany

Keywords: Requirements traceability, Interaction, Artifacts, UNICASE, Model.

Abstract: Requirement Traceability provides the ability to follow the life-cycle of a requirement from its evolution till subsequent refinement and use. A key issue that restricts the adaptation of approaches to create and maintain these relationships is the lack of tool support that employs a centralized repository for heterogeneous artifacts. Different artifacts are stored in different repositories and thus traceability links are expensive to maintain. Centralized repository can facilitate capturing the stakeholders interaction, which result in creation and modification of the artifacts and their relationship. These interactions hold the rationale behind changes. In this paper we propose a novel model-based CASE tool UNICASE, which aids in maintaining requirements traceability by incorporating disparate artifacts. Further, the tool facilitates capturing the evolution of requirements invoked by the informal communication in the form of discussion and comments.

1 INTRODUCTION

There had been several definitions of requirements traceability in literature. The most prominent one is “Requirements traceability is the ability to describe and follow the life of a requirement, in both a forward and backward direction. That is from its origin, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases” (Gotel and Finkelstein, 1994).

Requirements traceability benefits the facilitation in program comprehension, maintenance, impact analysis, software reuse and prevention of misunderstandings (Egyed and Grunbacher, 2005) by utilizing the relationship information available in software development artifacts (De Lucia et al., 2004). Artifacts are created and evolve through out the software development process. Various repositories are maintained in software projects to handle different kinds of artifacts (Čubranić and Murphy, 2003). These repositories include bug tracking systems, task repositories and source code management repositories.

An issue that restricts achieving fine-grained requirements traceability is the low interoperability between different repositories, where development artifacts are spread across several repositories with their own underlying traceability management mechanisms. This is especially true in frequently evolving projects where changes in artifacts stored in various repositories are not propagated to static requirements document (Raymond, 1999), which is stored at a different location.

Further in software projects, various stakeholders interact with each other along with the artifacts, which results in frequent change of artifacts and their relationships. These interactions can either invoke creation/modification of an artifact representing certain aspect of the system under development or a read-only interaction to gain insight of the development process. Therefore, we consider the need to capture the dynamic state of artifacts along with the interactions initiating the changes, to capture the rationale behind changes in requirements.

In this paper we present UNICASE (UNICASE, 2011), a model-based case tool which provides a centralized place for heterogeneous artifacts. A common platform to capture and modify different artifacts created through out the software development process at various life-cycle stages. The interaction between the stakeholders and the artifacts is also recorded. Combined together which provides fine-grained traceability between the artifacts at the same level of abstraction along with the artifacts at different level of abstraction (Antoniol et al., 2001). Further, the tool facilitates capturing the evolution of requirements invoked from the interaction of stakeholders over informal artifacts. Informal artifacts include comments and discussions.

In section 2 we give an overview of relevant exist-

Narayan N., Li Y., Helming J. and Koegel M..
INTERACTION CENTRIC REQUIREMENTS TRACEABILITY.
DOI: 10.5220/0003463502320238
In *Proceedings of the 6th International Conference on Evaluation of Novel Approaches to Software Engineering* (ENASE-2011), pages 232-238
ISBN: 978-989-8425-57-7

ing work in requirements traceability mainly with the tool support focus. Section 3 describes the core variables that provide the feasibility to capture and maintain traceability in UNICASE. Further in section 4 we briefly introduce various features of the tool, which combined together, provide the stakeholders means to interact among each other, and with the artifacts. Section 5 in detail explains the process of requirements traceability, as and how it is captured and maintained in UNICASE. Subsequent section presents future work and sums up with the conclusion.

2 RELATED WORK

Several research and commercial tools provides means to achieve traceability between artifacts, including some with specific focus on requirements traceability (IBM, 2011a) (IBM, 2011b) (Pinheiro and Goguen, 1996) (Boldyreff et al., 2002). In spite of having a major research focus, there is no tool support addressing every aspect of attaining requirements traceability. Artifact type being supported in each of them is based on the underlying design decision and extensibility to support new artifact type is limited.

Hipikat (Čubranić and Murphy, 2003) for example proposed a mechanism to integrate various independent artifact management repositories and on the top means to identify relationship between them for recommending related artifacts. As Hipikat work on the top of existing repositories, thus it completely depends upon the knowledge being stored in various repository and the methodologies to mine them.

Jazz (Frost, 2007) is an integrated development environment providing a centralized repository for tools supporting the development process such as planning, work item tracking, report and build system. Although Jazz provide means to capture various artifacts originating during the development process, its ability to maintain requirement traceability is limited and restricted by the commercial goals. Thus the underlying model itself cannot be extended or modified to suit project and domain-specific needs.

Lee et. al. (Lee et al., 2003) in their work presented an agile approach to capturing requirements and traceability. The proposed approach provides means to capture requirements in an informal manner and then transform it to formal requirements, to suit formal requirements-specification. This work is promising and can be very effective in capturing and transforming informal knowledge into formal one. Though the extensibility of this approach to consider formal artifacts remains a matter of investigation.

Hong et al. (Hong et al., 2010) presents the need for a tool to support requirements management with evolving traceability for heterogeneous artifacts in the entire life-cycle. They put forward the deficiency of existing change management system in handling traceability link evolution. Finally they conclude with their vision of providing a tool support to fulfill the need of requirement traceability management.

3 TWO DIMENSIONS OF TRACEABILITY

Traceability relationships reference artifacts of the same or a different level of abstraction, in both forward and backward direction (Winkler and von Pilgrim, 2010). Stakeholders interact with artifacts and their links through operations like creation, modification or deletion. Thus, in an incremental software development process, these interactions lead to the evolution of artifacts and the relationship between them.

We identify two dimensions necessary for creating and maintaining traceability, namely *artifact* and *time*. The *artifact* dimension is concerned with **what** is linked by traceability, in other words, the entities of concern. The *time* dimension represents the evolution of artifacts and their traceability information. Over the artifacts and throughout the time dimension stakeholders interact, which represents the rationale behind the changes. The interaction of stakeholders over artifacts comprise of informations as **who** performed the changes and **how**. **How** is represented in the terms of interaction sequence leading to the final state of artifacts. **Who** (role of the stakeholder) and **how** (see Figure 1) along with the initial and final state of artifacts can represent the rationale behind **why** the changes occurred.

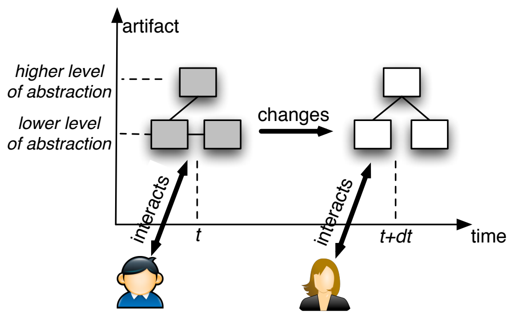

Figure 1: Two Dimensions of Traceability.

In the proposed approach of maintaining requirements traceability, artifact types and the relationship is defined in a unified extensible model, which depicts the *artifact* dimension. The *time* axis is realized in an operation-based version control system to record every interaction invoking changes over the artifacts and version the changes. In the next subsections we describe each of these two dimensions in detail.

3.1 Artifact Dimension: Unified Extensible Model

To address the low interoperability between geographically separated artifacts, we propose and employ a unified extensible model, which contains heterogeneous artifacts produced in software development process. The model comprises of four submodels (see Figure 2) from different level of abstraction (Bruegge et al., 2008): (1) The *requirements model* contains model elements like Scenarios or Use Cases and describes the system under construction in the application domain. (2) The *system model* describes the design using artifacts like UML class diagram. (3) The *collaboration model* contains artifacts like work items, sprints and meetings, while the (4) *organization model* defines an organization in terms of groups and members.

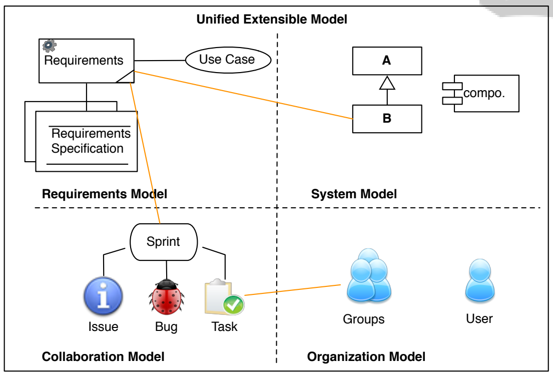

Figure 2: Unified Extensible Model.

Every artifact is modeled as `ModelElements` with its own unique identifier in the proposed model. Types of artifact is defined by extending `ModelElement`. In the unified model, links are defined to reference related model elements. Alternatively links can also be modeled as model element. Explicit links can exist between different types of artifacts. The model is easily extensible by adding new elements. Research approaches often require new model elements or attributes to be added, which is even possible during project run-time.

3.2 Time Dimension: Operation-based Versioning

Artifacts and the traceability links among them are constantly in evolution (Jiang et al., 2007) in an iterative and incremental software development paradigm. All the information of the artifacts and the links, as well as the interactions with stakeholders should be stored and versioned to provide the possibility of "time travel". We capture the interaction information in terms of operations and than persist it in an operation-based version control system called EMFStore. EMFStore facilitate versioning and collaboration. It provides change tracking, conflict detection, merging and versioning of models. Also it includes repository-mining facilities for data extraction. It employs the well-known checkout-update-commit workspace interaction schema.

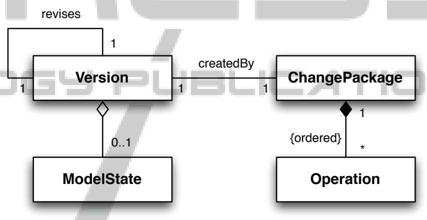

Figure 3: Version Model.

Figure 3 shows the version model of EMFStore. It is a tree of versions with revision and variant links. Every version contains a change package and can contain a full version state representation. A change package contains all the operations that transformed the previous version into this version along with the administrative information such as the user, a time stamp and the log message. An Operation is performed while stakeholders interact with involved model elements (artifacts) and their links. Such an operation can be applied to a project state, thereby executing the change that was recorded in the operation. In contrast to other versioning and change tracking approaches, the operation-based versioning preserves the original time-order in which the changes occurred. Therefore it can help reason the evolution of the artifacts and their traceability links.

Figure 4 shows the simplified taxonomy of operations (Herrmannsdoerfer and Koegel, 2010) considered in our approach to capture interactions resulting in change of model element. All operations refer to one ModelElement that is being changed by the operation. A `ModelElement` has values for a number of attributes. An `AttributeOperation` changes the value of an attribute of a model element. A `ReferenceOperation` creates or removes one or several traceability links between model elements. A `CreateDeleteOperation` creates or deletes a model element. A `CompositeOperation` allows to group several related operations to represent a refactoring, for example.

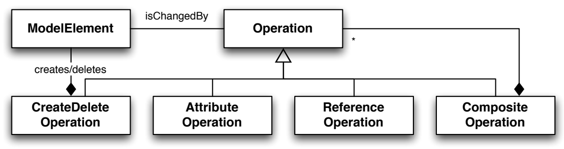

Figure 4: Operation Meta Model.

4 UNICASE

The two aspects central to achieving requirements traceability (see Section 3) is realized in a CASE tool, UNICASE. This tool is based on the Eclipse platform including Eclipse Modeling Framework (EMF) and Graphical Modeling Framework (GMF).

UNICASE consists of generic views and editors: (1) A tabular view, showing a filtered and sortable list of model elements, (2) A tree-based view, showing the containment structure of the model (3) A form-based editor to visualize and modify the content of model element (artifact) and its references (4) A diagram view, showing graphical representations such as Unified Modeling Language (UML) diagrams.

5 REQUIREMENTS TRACEABILITY WITH UNICASE

Existing software repositories including the one supporting heterogeneous artifacts versions and stores them. This does not sufficiently imply that they capture the interaction information underlying the requirements evolution. This interaction information along with the state of artifacts can derive the rational behind changes in the requirements or any related artifact, in an iterative and incremental development paradigm. Further having a centralized artifact repository facilitates identifying and propagating changes, which triggers major changes in the project such as creation of a new requirement or a discontinued functionality.

In the next subsection we describe how the traceability relationship between artifacts at different level of abstraction aids in maintaining requirements traceability in both forward and backward direction. Subsection 5.2 explains how the interactions in collaborative environment leads to evolution of traceability link between requirements and associated artifacts. From the technical perspective this feasibility is provided by underlying operation-based versioning 3.2. Following subsection 5.3 describes how the inception of new requirements or modification of existing one invoked from an informal artifact is captured in UNICASE.

5.1 Forward and Backward Traceability

It is evident that any of the artifacts are subject to change during the development process. Traceability gives assistance to traverse from a requirement to its implementation and vice-versa (Ramesh and Jarke, 2001), also termed as forward and backward traceability in literature.

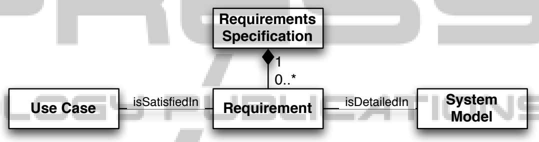

Figure 5: Excerpt from unified model with transition from requirements model to system model.

Software projects tend to start with loosely defined requirements especially projects with longer life span, representing stakeholders view. In subsequent process these requirements are analyzed and refined. Further these requirements provide a basis for the future refinement as system, subsystem and component requirements. Requirements in UNICASE can be linked to the artifact resulting from the interaction of stakeholders in the process of requirements engineering (see Figure 5). Some of the artifacts resulting at this level of abstractions are requirements specification and use case. Due to the complexities involved in requirements analysis, it is performed in close cooperation and collaboration with various stakeholders like users and customers. This interaction of various stakeholders plays a crucial role in requirements identification and refinement, and is considered part of the requirement traceability. An example of artifact produced within this interaction cycle is mail exchange between customer and analysts, negotiating requirements.

System model consists of the artifacts produced in an effort to map problem to the solution domain, during design phase. Artifacts resulting at this level of abstraction include various UML diagram types, representing static and dynamic states of the system under development. UNICASE provides inbuilt support for UML diagrams. Unified model can be extended to incorporate new diagram types. Eventually these diagrams can be linked to the requirements they represent at higher level of abstraction (see Figure 5).

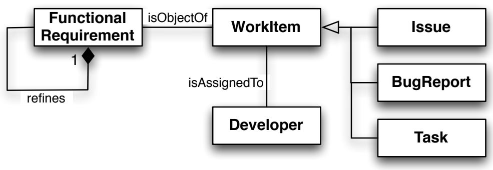

Figure 6: Requirements to realization.

Requirement traceability can play an important role in project management and planning. This can substantially enhance the efficiency of project managers by providing information on sprint status, due date and open high priority tasks in scheduling and resource allocation decisions. Collaboration model in UNICASE provides all the necessary artifacts type to achieve this. Model elements such as sprint provide possibility to manage tasks and bug reports. WorkItems are linked to requirement model by the association isObjectOf (see Figure 6). This expresses that the activities conducted during the execution of work item is related to the requirement or to its representation in the implementation. The type of relationship is derived based on the activity of work item. For example an implementation task linked to a requirement, means that this work item needs to be done in order to realize the requirement.

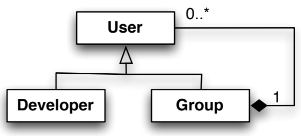

Figure 7: Developer organization.

Organization model in UNICASE consists of elements to represent various human entities. User is an abstract super class for every stakeholder, while the concrete implementation encapsulates the role centric properties. Every stakeholder is modeled as a model element in the project. For example a new developer can be added in the repository and then assigned a task by the association isAssignedTo (see Figure 6). Developer can belong to a group (see Figure 7), which in turn can compose several sub-groups. Further the organization model defines the different level of access rights based on the role of stakeholder.

As every artifact in UNICASE is derived from Model Element class, they inherit the property of having Annotations (see Figure 8).Thus every artifact can be linked to a source code attachment based on the relationship type. For example a bug report can be linked to an attachment consisting of a source code patch to fix it. Similarly a development task can be linked to the source code produced during the execution of the task. Hyper-graph (see Figure 9) shows the traceability relationship between different artifacts. At the center of the graph is the project itself as the root element. Sequence of interactions, recorded as operation combinedly represents the state of artifacts in certain project state.

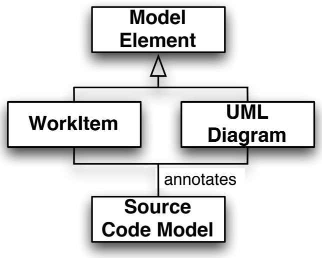

Figure 8: Traceability to source code model.

5.2 Evolution of Traceability

Collaborative software development process is central to the evolution of traceability links between artifacts in UNICASE. Sequence of interactions over artifacts is recorded as change package on the client side. This change package contains the interaction information and changes per interaction. After performing the necessary changes on the artifacts a stakeholder can propagate his changes to EMFStore, the underlying operation-based version control system for EMF models (see 3.2). Stakeholders can also update the state of their artifacts to reflect the changes made by other stakeholders. Further, a stakeholder can walk through every interaction over an artifact or the whole project itself, to follow its evolution process.

5.3 From Informal Artifacts to Formal Requirements

A centralized artifact repository like UNICASE eases capturing the interaction between stakeholders over artifacts created by informal communication. Informal communications stored in mailing list and forums as artifacts capture various aspects of the system under development. It can be identified in these discussions, various issues occurring at different stage of software project life cycle along with the collaborative approach to resolve them. These informal communications are also used to propose new feature request or changes in requirement. For example when

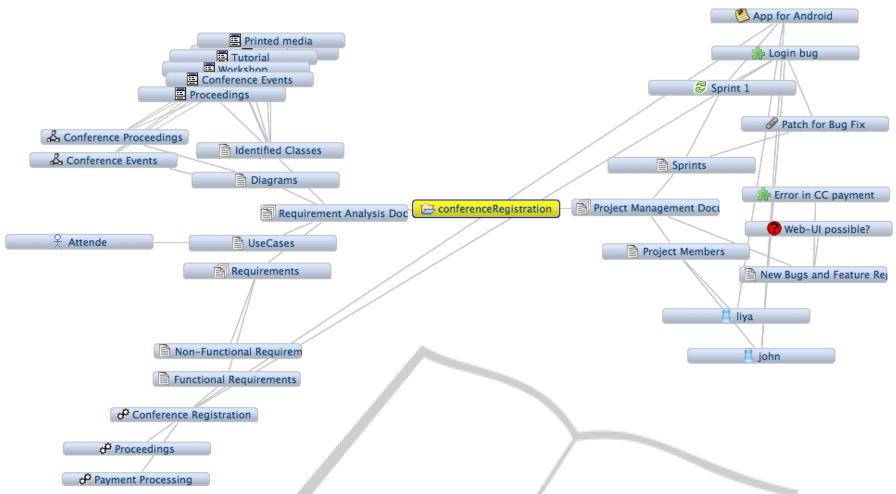

Figure 9: Hyper-graph showing traceability between requirements and other artifacts.

a new bug report or recently created issue, raises the need of having a new requirement.

In a previous study we showed how the system model elements and requirements evolve after a sequence of discussions and posts in informal communication (Helming et al., 2010). Externalizing this informal knowledge as formal requirements become even more important in case of frequently evolving projects where throughout the development original set of requirements gets blurred, outdated or lost (El-Ramly et al., 2002). Which eventually raises critical issues when the need arises for having a precise and up-to-date oracle for testing and other purposes. This is also true for maintenance of software where the need might arise to migrate a legacy project to support newer platform and execution environment (Lo and Khoo, 2008) (Jalote, 1997).

6 CONCLUSIONS

In this paper we presented a novel model-based case tool for capturing and maintaining requirements traceability. The tool considers interaction between stakeholders and with the artifacts of major importance to derive the rational behind change.

In the first part (see Section 3) we described two main variables central to the requirements traceability in our tool. Underlying unified extensible model provides a centralized place for heterogeneous artifacts. Through the interaction of various stakeholders new artifacts are created and evolved. This interaction is captured and versioned in a repository for model elements, thus providing the possibility to iterate over time. Change delta containing sequence of interactions, along with the initial and final state of artifact can be used to investigate the possibility of extracting rationale for change. Though an in-depth study is matter of future investigation.

In the second part (see Section 4) we gave a brief introduction to various views and editor provided by UNICASE to ease the interaction of stakeholders with artifacts. We also discussed the importance of change notification in requirements traceability along with the feasibility provided by the tool. Finally in the last part (see Section 5) we discussed how requirements traceability is captured and maintained with in UNICASE, in both forward and backward direction. Interoperability between artifacts at different and same level of abstraction eases capturing traceability links. Furthermore, the tool incorporates informal communication in a development project as artifacts. Thus providing the feasibility to capture traceability between requirements and the informal communications, which invoked the evolution of this particular requirement.

7 FUTURE WORK

This paper is a part of a research work with focus to identify, evaluate and present novel approaches for traceability between development artifacts. In several previous studies we have presented heuristic, which support the benefits of having a unified model to achieve better traceability between development artifacts.

Currently the proposed tool rely on API's provided by subversion, which is a widely used source code management system, to provide traceability from higher level model elements to source code. In the future we will focus on investigating approaches to capture and maintain fine-grained traceability between higher-level artifacts to source code artifact. We are further planning to investigate methodologies, which can help in identifying missing/broken traceability links between artifacts automatically or semi-automatically.

REFERENCES

Antoniol, G., Canfora, G., Casazza, G., and De Lucia, A. (2001). Maintaining traceability links during object-oriented software evolution. *Software: Practice and Experience*, 31(4):331–355.

Boldyreff, C., Nutter, D., and Rank, S. (2002). Active artefact management for distributed software engineering. In *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International*, pages 1081–1086. IEEE.

Bruegge, B., Creighton, O., Helming, J., and Kogel, M. (2008). Unicase-an ecosystem for unified software engineering research tools. In *Third IEEE International Conference on Global Software Engineering*.

De Lucia, A., Fasano, F., Oliveto, R., and Tortora, G. (2004). Enhancing an artefact management system with traceability recovery features. In *Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on*, pages 306 – 315.

Egyed, A. and Grunbacher, P. (2005). Supporting software understanding with automated requirements traceability. *International Journal of Software Engineering and Knowledge Engineering*, 0(0).

El-Ramly, M., Stroulia, E., and Sorenson, P. (2002). Recovering software requirements from system-user interaction traces. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, page 454. ACM.

Frost, R. (2007). Jazz and the eclipse way of collaboration. *IEEE Software*, 24:114–117.

Gotel, O. and Finkelstein, C. (1994). An analysis of the requirements traceability problem. In *Requirements Engineering, 1994., Proceedings of the First International Conference on*, pages 94 –101.

Helming, J., Narayan, N., Arndt, H., Koegel, M., and Maalej, W. (2010). From Informal Project Management Artifacts to Formal System Models. *System*.

Herrmannsdoerfer, M. and Koegel, M. (2010). Towards a generic operation recorder for model evolution. *Proceedings of the 1st International Workshop on Model Comparison in Practice - IWMCP '10*, page 76.

Hong, Y., Kim, M., and Lee, S.-W. (2010). Requirements management tool with evolving traceability for heterogeneous artifacts in the entire life cycle. In *Software Engineering Research, Management and Applications (SERA), 2010 Eighth ACIS International Conference on*, pages 248 –255.

IBM (2011a). IBM - rational DOORS - software. http://www-01.ibm.com/software/awdtools/doors/.

IBM (2011b). IBM - rational RequisitePro - software. http://www-01.ibm.com/software/awdtools/reqpro/.

Jalote, P. (1997). *An integrated approach to software engineering*. Springer Verlag.

Jiang, H.-y., Nguyen, T. N., Chang, C. K., and Dong, F. (2007). Traceability Link Evolution Management with Incremental Latent Semantic Indexing. In *Proceedings of the 31st Annual International Computer Software and Applications Conference*.

Lee, C., Guadagno, L., and Jia, X. (2003). An Agile Approach to Capturing Requirements and Traceability. In *Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE 2003)*. Citeseer.

Lo, D. and Khoo, S. (2008). Mining patterns and rules for software specification discovery. *Proceedings of the VLDB Endowment*, 1(2):1609–1616.

Pinheiro, F. and Goguen, J. (1996). An object-oriented tool for tracing requirements. *IEEE SOFTWARE*, pages 52–64.

Ramesh, B. and Jarke, M. (2001). Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering*, 27(1):58–93.

Raymond, E. (1999). The cathedral and the bazaar. *Knowledge, Technology and Policy*, 12:23–49. 10.1007/s12130-999-1026-0.

UNICASE (2011). Unicase. http://www.unicase.org.

Čubranić, D. and Murphy, G. C. (2003). Hipikat: recommending pertinent software development artifacts. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 408–418, Washington, DC, USA. IEEE Computer Society.

Winkler, S. and von Pilgrim, J. (2010). A survey of traceability in requirements engineering and model-driven development. *Software and Systems Modeling*, 9:529–565. 10.1007/s10270-009-0145-0.