

Automatic Change Support based on a Trace Model¹

Antje von Knethen
Fraunhofer Institute Experimental Software Engineering
Sauerwiesen 6
D-67661 Kaiserslautern, Germany
Phone: +49 6301 707 185 Fax: -200
vknethen@iese.fhg.de

Abstract

Software systems have to be changed under various circumstances during development and after delivery. Software development tools support the construction of different system artifacts (e.g., UML diagrams) but lack support for impact analysis of desired changes and for consistent changing of artifacts. Therefore, it is expensive and error prone to plan and implement changes. This paper presents our experience with analysis guidelines derived from a conceptual trace model. The guidelines support the effectiveness of change activities but, partly, they are difficult to apply. This experience motivated us to develop tools that implement the guidelines. This paper shows strengths and limitations of extending an existing software development tool and developing an own tool solution. We experienced in small case studies that both tools help to improve efficiency of change implementation.

1. Introduction

Software systems have to be changed under various circumstances during development and after delivery. Modifications during development occur, for example, if the system is developed incrementally, or if the requirements on the system themselves evolve during development. Modifications of the system after delivery occur, for instance, if faults have to be corrected, if performance or other attributes have to be improved, or if the system has to be adapted to a changing environment.

Current software development tools (e.g., Rational Rose) support the construction of different system artifacts (e.g.,

various types of UML diagrams) but lack support for impact analysis and change implementation. Therefore, it is expensive and error prone to plan and implement changes.

We developed guidelines for analyzing relationships based on a conceptual trace model for a specific product model in the domain of embedded systems. A product model defines different artifacts that have to be developed at different abstraction levels (e.g., system requirements, software requirements, software design, code). The product model used adapts the model of the Unified Process [1] for embedded systems [2].

In a controlled experiment, we experienced that applying our analysis guidelines result in more effective and efficient impact analyses. However, the guidelines were difficult to apply for maintainers because of their large number and complexity. This experience motivated us to develop a tool that implements our analysis guidelines. This paper shows strengths and limitations of extending a commercial development tool (i.e., StP/UML by Aonix) and developing a prototype to support impact analysis on a fine-grained level. In small case studies, we experienced that both tools improve the efficiency of a change implementation in comparison with applying our analysis guidelines and traditional tool support.

This paper is structured as follows. Chapter 2 gives an overview on existing tools for modeling and changing system artifacts consistently. Chapter 3 describes our conceptual trace model and the guidelines derived from it to support impact analysis and change implementation. Chapter 4 points out experience with applying our guidelines and motivates tool support. Chapter 5 shows an extension of an existing commercial development tool and a prototype that implements our guidelines. In addition, the chapter shows strengths and limitations of both tools.

¹ This work was supported by BMBF under grant VFG0004A ("QUASAR"), „Integrated quality assurance and requirements analysis for the software development in automotive systems”

Finally, in Chapter 6, we summarize our contribution and discuss future work.

2. Existing Tools Supporting Change

There are a large variety of UML software development tools available, such as Rational Rose, StP/UML, or Rhapsody by I-Logic. All development tools allow the developer to model different types of UML diagrams but, typically, change support is limited. There are some rudimental facilities to support changes, such as

- Navigation from one diagram entity to another in the same or a different diagram (e.g. from a state to its sub states)
- Renaming of diagram entities system wide (e.g., a class is renamed in all diagrams)
- Automatic generation of diagrams (e.g., automatic generation of collaboration from sequence diagrams)
- Syntactic and semantic checks. StP/UML, for instance, provides some consistency checks within a diagram (e.g., “each actor must have a name”) and between diagrams (e.g., “each actor described in a use case diagram must also be described in one or more sequence diagrams”).

Other facilities, such as an automatic impact analysis or guidance on how to implement a change are not provided.

There are other tools that support change activities. Dependency analysis tools, for instance, support a fine-grained impact analysis. They are based on program slicing and/or analyzing program dependency graphs. However, they cannot be applied to investigate relationships between artifacts at different abstraction levels.

In contrast, tracing tools provide support for capturing traces between entities of any system artifact. In addition, tracing tools manage and represent captured traces. There are commercial traceability environments (e.g., DOORS) [3] and research environments (e.g., TOOR [4], PRO-ART [5], or STAR-Track [6]). TOOR, for example, uses relations instead of simple links. A user is able to define relations that are meaningful for the kind of connection being made. In addition, the tool allows multiple, flexible ways to trace requirements, including both browsing and regular-expression search. PRO-ART allows different relationships of the defined three dimensions of requirements engineering to be managed. PRO-ART enables requirements pre-traceability by recording process execution, product evolution and their relations according to the traceability structure defined. The goal of STAR-Track is to reduce the cost of using a commercial requirements engineering tool by using Internet tools, and to support heterogeneous computing environments. Tracing tools still require human analysis to interpret the nature of the impact and assess its significance [7]. In addition, typically, their underlying trace model is somewhat too coarse-grained to support a precise impact analysis [8], the

trace model has to be defined by the user (see, e.g., [4]), or the model does not consider all development artifacts (see, e.g., [5]).

There are impact analysis tools that compute a set change impacts (i.e., system entities affected by a desired change) by analyzing relationships (e.g., IAS [9], and [10]). The IAS, for example, is a tool that determines the impact of a change by analyzing a graph of entities and links. Graphs are stored in a general-purpose repository and are constructed with the help of a Traceability Toolset that extract dependencies from existing sources (development and maintenance documentation, source code) and exploit them. Changes are modeled as events that are propagated through the graphs according to propagation rules predefined by the user of the system. Impacts are themselves considered new modifications, which can then be recursively propagated to obtain a "complete" set of impacts.

Some tools support the propagation of changes (e.g., [11]). [11] is a generic software engineering environment that can be specialized to a specific organizational context. The environment provides direct support for change propagation by allowing the user to define change propagation rules.

Most impact analysis and change propagation tools extract dependencies between artifacts by analyzing documents and code or by analyzing explicit links defined by the user. Typically, these tools do not determine what types of links have to be documented in advance to support a precise impact analysis later on and they do not investigate how the effort for documenting links can be reduced. In addition, such tools do not support the identification of a starting impact set (interpretation).

There are tools supporting inconsistency management [12] in development environments: Grundy et al. [13] describe an architecture for software development tools that helps to detect and manage inconsistencies that occur during software development (e.g., structural, semantic, interpersonal, and interprocess inconsistencies). They point out techniques for presenting inconsistencies to the developer (e.g., highlighting system entities to indicate the presence of inconsistencies, inconsistency descriptions that inform the user of inconsistencies, providing change histories). In addition, they give an example of how such a development environment may look.

3. Conceptual Trace Model and Guidelines

A conceptual trace model (also called reference model [14]) determines types of documentation entities and their relationships that have to be traced to fulfill a certain purpose. We developed a conceptual trace model for a fine-grained impact analysis.

In contrast to existing trace models (see, e.g., model defined by [14]), we explicitly separate logical entities from

documentation entities and we suggest tracing logical entities to support a fine-grained impact analysis. A logical entity describes the semantics of a documentation entity. Each documentation entity represents one or more logical entities (e.g., a textual system requirement (documentation entity) represents a system task “close window” (logical entity)). Figure 1 shows documentation and logical entities and their relationships.

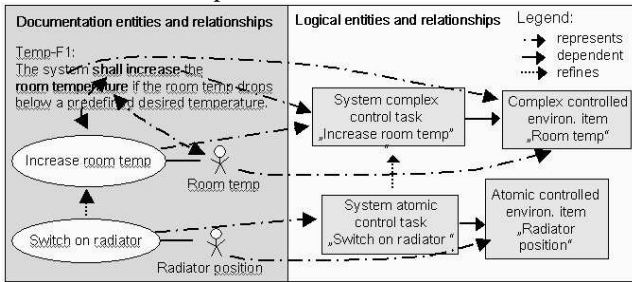


Figure 1. Documentation and logical entities

We argue for separation of logical and documentation entities because, on the one hand, investigating logical entities helps the maintainer to identify a starting impact set. A change request may use documentation entities for describing change impacts that differ from the documentation entities described in the system artefacts. Therefore, it is easier to identify logical entity types described in the change request (e.g., a system control task should be removed), then, determine how the types are represented by documentation entities in the system artefacts (e.g., system control tasks are represented by use cases), and finally, search for documentation entities within the system artefacts affected (e.g., use case “close window” should be removed).

On the other hand, it is easier to identify different types of relationships that have to be documented. We distinguish between three general types of relationships that have to be traced:

- *Dependency relationships* are relationships between documentation entities that represent logical entities at the same level of abstraction. A system complex control task “increase room temp” has, for example, an influence relationship to a controlled environmental item “room temp”.
- *Refinement relationships* are relationships between documentation entities that represent logical entities at different levels of abstraction. A system complex control task “increase room temp” has, for instance, a refinement relationship to an system atomic control task “switch on radiator”.
- *Representation relationships* are relationships between two documentation entities that represent the same logical entity. A part of a textual requirement and an actor describe, for instance, a controlled environmental item “room temp”, therefore, there is a representation relationship between the documentation entities.

Dependency and refinement relationships between documentation entities can be derived from relationships between logical entities. Logical entities of a certain domain (e.g., embedded control systems) require domain dependent relationships that can be identified more easily, if logical entities and their relationships are investigated independently from their representation. Based on the required relationships, constraints can be defined (e.g., each system control task must have an influence relationship to exactly one controlled environmental item). Representation relationships result from investigating the relationships between logical and documentation entities.

According to our distinction between logical and documentation entities, our conceptual trace model consists of a conceptual system model and a conceptual documentation model. The conceptual system model defines types logical entities, relationships between them, and constraints on the relationships. The conceptual documentation model defines types of documentation entities, relationships between them, and constraints on the relationships.

We developed a certain trace model for a specific product model in the domain of embedded systems. Our conceptual system model defines types of logical entities of an embedded system that occur at different abstraction levels (e.g., system tasks, software tasks, environmental items, hardware items, software items). We experienced that this model can easily be adapted to variants of embedded systems (e.g., system artifacts describing electronic control units [19]). Our conceptual documentation model extends the documentation entities determined by the product model and the description techniques used by the Do-it-process [2]. If the Do-it-process is used in an organization, our conceptual documentation model and our guidelines for analyzing relationships can be used directly. Using another process requires an adaptation of this model and of the guidelines (an adaptation of our documentation model to completely textual requirements documents is described in [19]).

3.1 Conceptual System Model

Our concrete conceptual system model is based on the Four Variable Model for embedded systems developed by Parnas et al. [15], [16]. Figure 2 gives a subset of our conceptual system model.

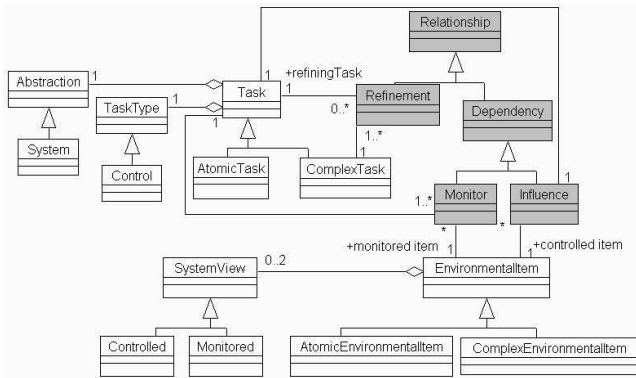


Figure 2. Subset of conceptual system model

Our model distinguishes mainly between types of:

- Items at different abstraction levels (e.g., controlled or monitored environmental items),
- Tasks at different abstraction levels (e.g., system complex and system atomic control tasks),
- Dependency relationships (e.g., monitored environmental items must have monitor relationships to a system task), and
- Refinement relationships (e.g., a system complex task must have a refinement relationship to a system atomic task; a system task must have a refinement relationship to a software task and at least two hardware tasks).

Our conceptual system model is a UML class diagram. Constraints on relationships are described in natural language and in OCL (Object Constraint Language) [17]. A more detailed description of the conceptual system model is given in [18].

3.2 Conceptual Documentation Model

Our conceptual documentation model extends the documentation entity types determined by the Do-it-process [2]. It refines these documentation entity types to allow clear identification of logical entity types. Using the Do-it-process, two logical entity types “system control task” and “software control task”, for instance, are represented by the same documentation entity type “use case”. We introduced two documentation entity types: a “system control use case” that represents a “system control task” and a “software control use case” that represents a “software control task”. This distinction allows the logical entity types to be identified in software documents. Constraints on logical entity types can now be transferred. A conceptual system model constraint, for example, “a system control task must have a refinement relationship to a software control task” can be transferred to a documentation model constraint “a system use case must have a refinement relationship to a software use case”.

Figure 3 gives a subset of our conceptual documentation model.

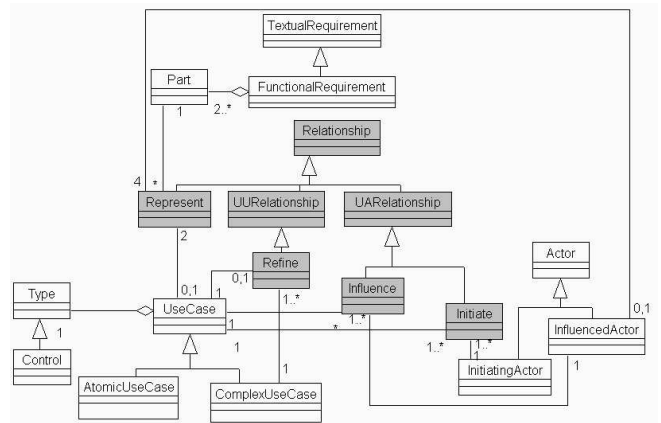


Figure 3. Subset of conceptual documentation model

Our model distinguishes mainly between types of:

- Documentation entities (e.g., system complex control use cases, system atomic control use cases, or initiating actors),
- Dependency relationships (e.g., each system control use case must have an influence relationship to an influenced actor). We derived these relationships from the conceptual system model. Each relationship described for a logical entity type must be true for a documentation entity type that represents the logical entity type.
- Refinement relationships (e.g., each system complex control use case must have a refinement relationship to a system control use case; each system control use case must have a refinement relationship to a software control use case). Again, we derived the refinement relationships from the conceptual system model.
- Representation relationships (e.g., a use case must have a representation relationship to a part of a textual functional requirement and to a use case description because all three represent a system task).

We used UML extension mechanisms (i.e., the mechanism of stereotypes) to extend UML description elements. For natural language documents, we annotate documentation entities with the logical entity type described. Figure 4 shows a subset of a system use case document. Different types of use cases, relationships, and actors are signed with stereotypes.

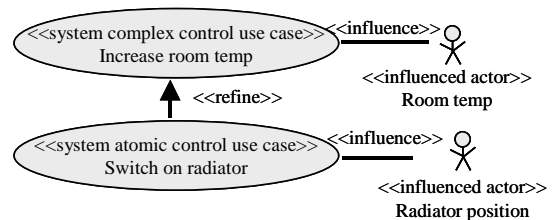


Figure 4. Extended documentation entities

Similar to the conceptual system model, our conceptual documentation model is a UML class diagram. Constraints

on relationships are described in natural language and in OCL. A more detailed description of the conceptual documentation model is given in [18].

3.3 Analysis Guidelines

We derived our analysis guidelines from the constraints defined on the relationships of the conceptual documentation model. We developed three sets of guidelines: for each investigated document one (i.e., system customer and software customer requirements document, software developer requirements document, and software design document). Each set of guidelines includes:

- A description of the artefacts included in the document (e.g., use cases, or class diagrams),
- An explanation of the logical entities described in the document (e.g., environmental items and system tasks in the system requirements document),
- A description of how logical entities are represented in the document and what relationships have to exist. This description is structured according to the different logical entity types. The following paragraph gives a subset of such a description taken from the guidelines for the system and software customer requirements (i.e., use case diagrams and descriptions):

Complex control task

- *A complex control task is represented by a “complex use case” in the “system use case diagram” and in the “software use case diagram”.*
- *A complex control task is described by a “system “complex use case” description” and a “software “complex use case” description”*
- *A „complex use case“ must have one or more refinement-relationships to “use cases”.*
- *A „complex use case“ must have one or more influence relationships to an “influenced actor”.*

4. Experience with Analysis Guidelines

We evaluated our analysis guidelines in a controlled experiment with 25 upper graduate students. The experiment focused on the viewpoint of the maintainer and investigated the effects of applying two approaches (extended guidelines and traditional guidelines) on: (1) understanding: a maintainer has to understand the required change and the system to be changed, (2) impact analysis: a maintainer has to analyse the impact of a required change on the system, and (3) implementation of changes: a maintainer has to consistently implement the required changes. The term “traditional guidelines” refers to development guidelines provided by the Do-it process. “Extended guidelines” refers to our analysis guidelines. The experimental hypotheses, design, variables, subjects, tasks, procedure and threats to validity are described in detail in [18].

The results show that the extended guidelines had

- A significantly beneficial influence on the correctness and completeness of the maintainer’s understanding of a system and on a set of change impacts predicted.
- A beneficial (but not significantly) influence on the efficiency of the understanding and impact analysis process.
- A beneficial influence on the efficiency and effectiveness of change implementation. We could not perform a statistical evaluation concerning this aspect because of too few data points.

The experimental results show that our analysis guidelines support change activities, especially impact analysis, effectively. However, we do not get significant benefits according to the efficiency of the understanding and impact analysis process. In addition, experimental subjects claim that the number of guidelines, especially, for the software developer requirements document, was too large and single guidelines were too complex and difficult to apply. All three sets of guidelines comprise about 22 pages in natural language. Based on this experience, we decided to implement our analysis guidelines in a tool. Our goals were to reduce the number and complexity of the guidelines to be checked by the maintainer.

5. Tools that Realize Trace Model

We had several requirements for a supporting tool:

1. The tool shall implement our analysis guidelines
2. The tool shall reduce the effort for linking documentation entities.
3. The tool shall be applicable for modeling and changing large systems.
4. The tool shall change documentation entities automatically if possible.
5. The tool shall primarily support the maintainer (i.e., an incremental impact analysis and change implementation shall be performed).
6. The tool shall produce two sets of change impacts: primary change impacts have to be changed to implement a desired change and secondary change impacts have to be changed potentially to implement the required change. Primary change impacts are documentation entities that are related by representation or refinement relationships to the initially changed entity. Secondary impacts are entities related by dependency relationships.
7. The tool shall be easy adaptable if new guidelines are defined or existing guidelines are changed.

5.1 Extension of commercial development tool

We investigated the possibilities of extending a commercial development tool because of requirement 3. We chose the commercial software development tool StP/UML for several reasons:

- It supports diagrams that are part of the investigated product model,
- It provides various extension mechanisms (UML-based and its own), and
- We had experience with its application.

StP/UML mainly provides two extension mechanisms: semantic extensions of editors, which allow the meaning of single diagram entities to be changed or new diagram entities to be constructed, and functional extensions of editors, which allow new operations on diagrams (e.g., syntax check function) to be introduced.

We extended StP/UML by two types of impact analysis functions and four types of consistency checks. An impact analysis is supported as follows: The user marks an item in an artifact. Then, s/he chooses an impact analysis function from the menu. The tool performs an impact analysis and provides the result in the form of a list of change impacts. Figure 5 gives an example of how change impacts and inconsistencies are described.

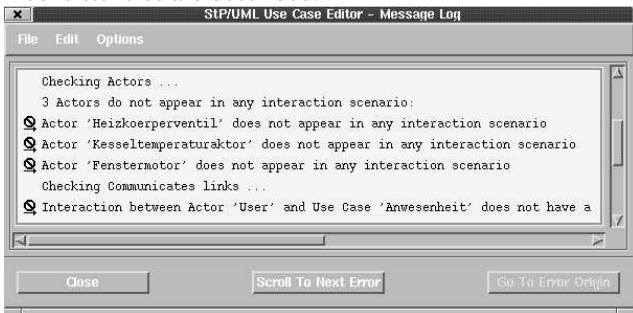


Figure 5. Message log to display impacts/inconsistencies

Extending a commercial development tool, such as StP/UML has following benefits and limitations:

- + The tool could easily be extended. The extension mechanisms are provided to each user of the tool. Therefore, an expert in an organization can easily integrate such impact analysis functionality as we do. An expert could also do adaptation of the implemented guidelines easily.
- + The provided extension is applicable for the modeling and changing of large systems because a commercial development tool is suited for the modeling and changing of large systems. We did not find possibilities to prepare the set of change impacts (e.g., to distinguish two sets of change impacts or two highlight system entities to indicate the presence of inconsistencies as described in [13]. This may limit the applicability of the impact analysis functions in a large project.
- It was difficult to identify which consistency functions are predefined within the tool. Each tool should explain the consistency functions provided.
- Links that could be analyzed during impact analysis had to be set manually. The tool does not support the

definition of links between entities of different artifacts, for example, a refinement relationship between a use case and a set of methods. We established links, for example, by annotating each use case by the refining methods. Establishing such links and maintaining them is effort intensive.

- We did not find possibilities to automate changing. We could only use the message log to represent change impacts and inconsistencies.
- The tool does not support all artifacts determined by the Do-it process. Use case descriptions, for instance, are not supported. The implemented impact analysis does not investigate relationships to artifacts that are not supported by the tool.

5.2 Development of own prototype

Based on our experience with extending a commercial development tool, we decided to develop an own tool solution that focuses on (1) reducing the effort for establishing links, (2) supporting automatic changing, and (3) supporting artifacts that are typically not included in a commercial development tool.

The implemented prototype allows system customer requirements (e.g., use case diagrams and descriptions) and software customer requirements (e.g., use case diagrams and descriptions) of an embedded system to be constructed and changed. The tool provides the functionality to add, delete, and rename entities of the system customer requirements and to propagate changes consistently within the system and to the software customer requirements.

The interfaces provided by the tool to change different documentation entities consider the constraints on relationships.

Figure 6 shows the interface that allows a new use case to be added.

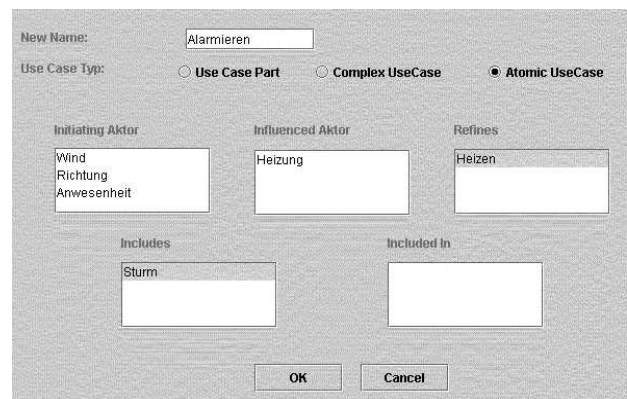


Figure 6: Interface to add use case

Relationships between documentation entities are established with the help of the user or automatically, if new entities are added.

Changing (adding, deleting, changing) of a documentation entity leads to an automatic impact analysis. The tool shows the result of the impact analysis in form of two lists: (1) a primary change impacts list and (2) a secondary impacts list. These lists look like the message log provided by the commercial case tool. If a new documentation entity is introduced, the tool computes a set of documentation entity types as primary and secondary change impacts. If a documentation entity is deleted or changed, the tool computes a set of concrete documentation entities. If all necessary information is available (i.e., this is especially the case for representation relationships), the tool automatically changes primary impacts.

Developing an own tool solution has following benefits and limitations:

- + The prototype supports setting of links. This reduces the effort for linking.
- + Some changes can be performed automatically. This reduces the effort for changing.
- The facilities of the prototype are limited. The major problem with developing an own tool is the effort that has to be spent for implementation. Our prototype provides only limited functionality for modeling and changing and cannot be applied in larger projects. Much more effort is necessary to develop a tool that is more relevant for an industrial application.
- The tool implements strict observance of the analysis guidelines and guides the maintainer through the change process. This seems to be too much restriction for experienced maintainers. In addition, the tool cannot easily be adapted to specific processes in an organization.

6. Conclusion

We developed a conceptual trace model to support impact analysis and change implementation. In comparison to existing conceptual trace models, our model is more fine-grained because we suggest tracing logical entities. In addition, it defines constraints on relationships, and it is tailored to a specific product model and description techniques used in the domain of embedded systems. Based on our conceptual trace model, we derived guidelines for analyzing relationships.

Experimental evaluation shows that applying our analysis guidelines results in more effective change activities; however, the guidelines are too complex and, partly, difficult to apply. Therefore, we decided to implement a tool that helps the scale of the guidelines to be reduced. In comparison to other impact analysis and change propagation tools, our tool does not intend to be a general tool that supports impact analysis independent of artifact types and concrete propagation rules. Our intention was to implement our specific analysis guidelines.

We experienced that commercial development tools, such as StP/UML can be extended easily by an expert of the development organization. Such an extension could but need not be provided by a tool supplier. Extending a commercial development tool has the benefit that existing features of the tool can still be used. Therefore, we assume that the tool can be applied in large projects. However, extending an existing tool is limited to the extension mechanisms provided by the tool. We did not find possibilities to automate changing and concepts to visualize change impacts are limited. In addition, we experienced that dependency and representation links can relatively easy be established and maintained. It is more difficult to establish and maintain refinement links because, typically, development tools do not provide mechanisms to describe entities at different abstraction levels and to link these entities. In opposite to the tool extension, the implemented prototype automates certain types of changes and reduces the effort for establishing links. However, so far the prototype is not applicable for modeling and changing large systems.

More research is required to develop tool environments that supports change activities across the boundaries of tools (e.g., RM tool, development tool, test suites).

The incremental impact analysis and change implementation suggested by both tools is suitable to support the maintainer. In two small case studies (i.e., each case study was performed by one person), we experienced that the effort of implementing a required change is reduced by both tools in comparison with using our analysis guidelines and traditional tool support. More research is required to support cost estimation of changes. The impact of a desired change has to be predicted for all system artefacts and cost models have to be integrated.

7. Acknowledgement

We would like to thank Wolfgang Wagenbichler and Michael Graf who had a major stake in developing the tools. In addition, thanks go to Barbara Paech and the anonymous reviewers how helped to improve this paper.

References

- [1] I. Jacobson, G. Booch, and J. Rumbaugh, "The Unified Software Development Process", *Addison-Wesley Object Technology Series*, 1999.
- [2] A. von Knethen, and J. Münch, "Entwicklung eingebetteter Software mit UML: Der Do-it-Prozess V 1.0", *SFB-Report No. 05/2000*, Sonderforschungsbereich 501, Dept. of Computer Science, University of Kaiserslautern, 2000, in german.
- [3] INCOSE Requirements working group, "Requirements Management, Requirements Analysis, and Requirements Engineering methodology", <http://www.incose.org/rwg/>, 1993 - 1999.

- [4] F.A.C. Pinheiro and J.A. Goguen, "An Object-Oriented Tool for Tracing Requirements", *IEEE Software*, Vol. 13, No. 2, 1996, pp. 52-64.
- [5] K. Pohl, "PRO-ART: A Process Centered Requirements Engineering Environment", In: M. Jarke, C. Rolland, A. Sutcliffe, R. Dömges (ed.), "The NATURE of Requirements Engineering", *Shaker Verlag*, 1999, pp. 255-278.
- [6] X. Song, B. Hasling, G. Mangla, B. Sherman, "Lessons Learned From Building A Web-Based Requirements Tracing System", In: *Proc. of International Conference on Requirements Engineering*, 1998, pp. 41-50.
- [7] W. Lam, V. Shankaraman, and G. Saward, "Requirements Change: A Dissection of Management Issues", *Proc. of the 5th. International Workshop on Requirements Engineering: Foundations of Software Quality REFSQ'99*. Heidelberg, Germany, 14-15 June, 1999.
- [8] S. A. Bohner and R. S. Arnold, "Software Change Impact Analysis", *IEEE Computer Society Press*. Los Alamitos, California 1996.
- [9] S. Barros, Th. Bodhuin, A. Escudié, J.P. Queille, and J.F. Voidrot, "Supporting Impact Analysis: a Semi-Automated Technique and Associated Tool", *Proc. Conference on Software Maintenance* 1995, pp. 42-51.
- [10] J. Li, P. H. Feiler, "Impact Analysis in Real-time Control Systems", *International Conference on Software Maintenance*, 1999, pp. 443-452.
- [11] J. Han, "Supporting Impact Analysis and Change Propagation in Software Engineering Environments", *Technical Report 96-09*, Peninsula School of Computing & Information Technology, Monash University, Australia, 1996.
- [12] G. Spanoudakis, and A. Zisman, "Inconsistency Management in Software Engineering: Survey and Open Research Issues", In: *Handbook of Software Engineering and Knowledge Engineering*, Vol. 1 (Ed.) Chang S. K., World Scientific Publishing Co, 2001.
- [13] J. Grundy, J. Hosking, and W. Mugridge, "Inconsistency Management for Multiple-View Software Development Environments", *IEEE Transactions on Software Engineering*, Vol. 24, No. 11, 1998, pp. 960-981.
- [14] B. Ramesh and M. Jarke, "Towards Reference Models for Requirements Traceability", *IEEE Transactions on Software Engineering*, Vol. 27, No. 1, 2001.
- [15] D. Parnas, and J. Madey, "Functional Documentation for Computer Systems Engineering", *CRL Report 237*, McMaster University, Hamilton, Ontario, Canada 1991.
- [16] R. Bharadwaj and C. Heitmeyer, "Hardware/Software Co-Design and Co-Validation: Using the SCR Method", *HLDVT'99*, Nov. 1999.
- [17] J. Warmer, and A. Kleppe, "The Object Constraint Language. Precise Modeling with UML", *Addison-Wesley Object Technology Series*, 1999.
- [18] A. von Knethen, "Change-Oriented Requirements Traceability. Support for Evolution of Embedded Systems", *PhD Theses in Experimental Software Engineering*, Vol. 9, Fraunhofer IRB Verlag 2001.
- [19] A. von Knethen, B. Paech, F. Kiedaisch, and F. Houdek, "Systematic Requirements Recycling through Abstraction and Traceability", *Joint Int. Requirements Engineering Conference*, 2002.