# Traceability for Managing Evolutionary Change

**Patrick Maeder, Matthias Riebisch and Ilka Philippow**
**Technical University of Ilmenau**
**Department of Software Systems/Process Informatics**
**D-98693 Ilmenau, Germany**
{patrick.maeder|matthias.riebisch|ilka.philippow}@tu-ilmenau.de

## Abstract

Traceability links can provide essential support for evolutionary development of software, beyond requirements engineering e.g. for reuse & design decisions, design and code comprehension, effort estimation, checks for completeness and project management. For maximum support, traceability links are required not only for large grained artifacts but for fine grained ones as well. The establishment and the maintenance of these links is crucial, because inconsistent links prevent the aimed positive effects. However, a high effort for traceability links would inhibit the positive effects as well. In this paper, the state of the art approaches of definition and application of traceability links are investigated. They are integrated together with link update operations within development methods. The investigation and the integrated approach was evaluated in various projects in research and industry in the fields of both forward and reverse engineering.

*Keywords: traceability, evolutionary change, development process, software maintenance, comprehension, documentation, roadmap*

## 1 Introduction

Development and maintenance costs of software systems are increasing more and more. Maintenance costs require 67% of the project budgets [33] or more. The requested functional and non-functional requirements are subject of continuous changes. These changes occur from the very beginning of a system's development and throughout its whole life cycle. Iterative software development processes have been introduced to meet and resolve this problem. Usually, the adaptation of software systems to new and changed requirements is called software maintenance. To perform changes in complex systems, it is necessary to establish a permanent connection between changed requirements and their realization in design and maintenance activities. The connection has to follow the different development artifacts and abstraction levels, from analysis documents to source code. Traceability enables such a connection.

Gotel and Finkelstein [11] define requirements traceability as "ability to describe and follow the live of a requirement from its origin, through its development and specification, to its deployment and use, in both forwards and backwards direction". Beyond it, they distinguish pre- and post-requirements traceability for aspects that refer to the requirements life prior to its inclusion in the requirements specification and for those which result from the inclusion.

The connection of two software artifacts (e.g. requirement and architectural element) is usually called Traceability Link. In addition, Letelier [14] has introduced two kinds of links: contribution structures (links between stakeholders and specifications) and rationale associated to specifications, including alternatives, decisions, etc. These link types are aimed to support the following aspects:

- Validation systems functionality versus verification
- Improvement of communication and cooperation among all stakeholders
- Supporting developers to get into software systems and understand them
- Guaranteeing the contribution of all involved stakeholders
- Enabling impact analysis and effort estimation
- Simplifying the examination of legally aspects, e.g. contract obligations, guidelines or constraints
- Improving of change management, avoiding studying considerations already excluded
- Understanding of design decisions and justification of their results

This connection is required in both methods and models. In principle, a connection can be realized in three qualified steps:

1. Using of appropriate and self descriptive terms for names and comments, pointing at connections and enabling the partly or fully automated linking
2. Applying only of unified and clear textual names
3. Repository-based storage of the artifact and establishing unequivocal connections between them in accordance to a uniform scheme (during lifetime of connection)

Each of these steps leads to increased effort and costs for the establishing and maintaining of relations and connections. But simultaneously the stored information grows significantly in terms of usefulness and reliability. Only the results of the third step are considered to be traceability links. Besides the storage of the pure relation, they allow to include additional information like decision aspects or alternative solutions. Traceability links offer a further relevant advantage by enabling a version control of the links themselves, especially if the linked artifacts are managed by different tools.

The objective of this paper is to review and to integrate existing approaches related to traceability for managing evolutionary change. Based on the investigation of these approaches open problems are recovered, that must be discussed and resolved for integrating traceability concepts into project management and software development processes.

A vision demonstrating the impact of powerful traceability is explained in chapter 2. In Chapter 3 the state of the art approaches are evaluated within practical projects. The detected open issues are discussed in chapter 4 considering especially the further development of existing methods and techniques.

## 2 Vision: Comprehensive Support for Round Trip Engineering

The overall goal consists in establishing and maintaining traceability links between artifacts of all types, stored in the same repository as the artifacts themselves. Traceability, provides connections in both directions, according to forward and to reverse engineering activities. In many cases, artifacts have to be managed by different tools and stored in different repositories.

If an artifact is changed in any way, the traceability links connected to this artifact are changed accordingly. Changes are the result of engineering activities. Each engineering activity – forward or backward – updates the traceability links.

Artifacts are described at different degrees of formalization and abstraction. According to both degrees, the activities of changing these artifacts are described more or less formal. For formally described changes, the effects to traceability links are defined formally as well. Updates of the links can be performed automatically, e.g. by tools within an Integrated Development Environment IDE. After each update, the links are left in a consistent state again.

For changes described informally or semi-formally, only support for updating the links is possible. The developer is required to decide about the changes of the links. Tools can support these changes by proposing links and artifacts to be linked. Consistency of the resulting links has to be checked immediately.

The changes to traceability links are embedded into development activities in a way that they can be performed transparently, i.e. so that the developer does not have to care about these changes. This embedding is reached by integrating traceability link changes to all interrelated development methods and their activities.

According to our vision, traceability links support the preparation and the realization of changes at every stage:

- Comprehension: an online documentation of a system enables browsing to improve the comprehension of the developer. Traceability links provide the basis for hyperlinks between documents and artifacts. The developer can easily change his point of view to understand all necessary details of a complex system.
- Planning: Traceability links enable an evaluation of the impact of a change. On this base, an effort estimation is provided for planning and deciding the development activities.
- Design: Traceability links provide relations between design artifacts and patterns and principles, i.e. architectural styles.
- Implementation: Traceability links point to related parts of a system which are influenced by a change.
- Verification & Validation: Traceability links enable checks for completeness and coverage of a change against changed requirements and design artifacts, as well as checks for the consistency of the system artifacts.

## 3 Evaluation of Existing Approaches and Experiences from Practice

The development of traceability links increases effort and costs for software development, as mentioned before. Above that, the establishing of traceability links crossing different abstraction and development levels in not trivial, not even for system specialists and experts. Moreover, the continuous expected change of requirements demands the maintainability of traceability.

In the last decade there have been already some works for establishing traceability links between model artifacts. especially between requirements and for documentation issues. These works are investigated in section 3.1.

There is a need for effort reduction for link maintenance, which is influenced by embedding these links into development activities as well as by providing intermediate artifacts, e.g. models. Approaches an methods concerning these issues are discussed in section 3.2.

Other methods apply traceability links for reverse engineering purposes. These works have successfully provided models, process descriptions as well as generator tools. Section 3.3 is dealing with them.

Another important field where contributions have been developed is the maintenance of traceability links. Methods for the integration of link maintenance into change management procedures are investigated in section 3.4.

### 3.1 Environments for Traceability Representation

Representation concepts for traceability play an essential role. Not only start and ending points of a particular link have to be fixed. The significant additional information, e.g. direction, creation time of link, stakeholders, decisions and alternatives must be determined, too.

On the market there are various commercial and research tools for partly or fully solving this task. But nevertheless it lacks of established standardized methods for case tool provider. Therefore, actually a satisfying tool support for traceability is not yet available. Some promising approaches for traceability representation are investigated now.

#### 3.1.1 Traceability Links Based on UML

Letelier [14] makes use of the widespread UML for software development and their good extension abilities for the development of a so called traceability framework. In this approach the representation of links using UML objects is shown. Unfortunately, the methodical application of the framework is not explained.

Using UML for link representation is a well-founded decision. The acceptance for application of the UML for the description of object oriented systems is growing more and more. The UML offers representation facilities during almost the whole development process (requirement specification, analysis and design modeling, code generation and test).

Figure 1 shows enhancements of the UML metamodel. But, in the referenced paper considerations on bounding various other relevant description means into the framework (requirements engineering tools, CASE tools like Simulink, ASCET SD) and how to integrate the activities into the software development process are missing.

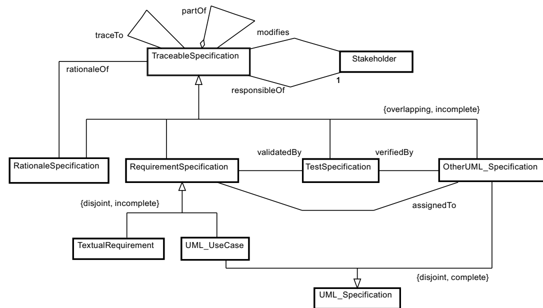


Figure 1: Enhancements to the UML Metamodel [14]

An implementation of this concept has been used and evaluated in the first project phases of a large industrial reengineering project [17]. The CASE tools Rational Rose and Requisite Pro have been connected via a shared repository. Traceability links have been established and maintained. It was very effort consuming to maintain links on a fine grained level, e.g. for style requirements and to style classes. This high effort was observed especially in the cases for links between single requirements and design elements. The effort was much lower for links to more abstract architectural elements. These investigations lead to the discussion of the gap issue, see section 3.2.

### 3.1.2 Traceability Links Based on Hypertext

Potts and Takahashi have developed and described in [21] a so-called conversation framework, the Inquiry Cycle Model. It consists of three parts (see figure 2): the representation of requirements and their mutual relations (shared information), a speech act model and the topology of changes. The first part serves for mapping of requirements and relations (traceability links) on Hypertext. The speech act model contains questions, answers, and reasons. Questioning is considered as the main speech act. The third part deals with requirements evolution and focuses all elementary changes. The Inquiry Cycle starts with a question about the systems specification. Using the speech act model found answer one ore more change requirements are defined and performed. This process leads to a new version and specification.

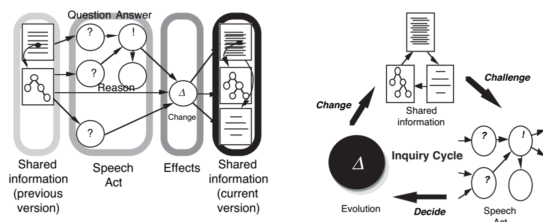


Figure 2: Conversational Model and Inquiry Cycle [21]

Ebner and Kaindl use in [8] a Requirements Engineering Trough Hypertext (RETH) method. They extend this method, but do not describe it in detail.

### 3.1.3 Integration of Hypertext Links with Code Tags and Documentation

Former research of the authors of this paper has been conducted to integrate traceability links into software development processes and repositories. As one of the results, Sametinger et al. [27] integrate traceability links with links to entities for structuring source code e.g. architectural styles, design patterns and aspects. Documentation generator tools – especially javadoc – are used to provide on-line accessible means of documentation for supporting evolutionary development. In this method, javadoc tags are extended to express relations mentioned above by hypertext links. Based on these tags, a documentation generator tool produces an on-line documentation.

### 3.1.4 Comparison of Hypertext Links to Other Types to Traceability Links

Using Hypertext for the construction of traceability links has proved as simple and low cost alternative [21, 8]. Many existing tools are able to manage and to emphasize Hypertext even if their integration and interaction is mostly insufficient. Moreover, the issue of storing additional, semantical information attached to the links is not yet solved consistently.

Originally the hypertext-based methods do not use a central repository. As a consequence, maintaining consistency constitutes an important success factor for integrating additional information. Unfortunately the authors of the related paper do not discuss this issue. The method for integration traceability links remains also an unsolved issue, as well as the version-control of the links.

## 3.2 Reducing Gaps Between Models in Software Development Processes

In software development processes there exists the well-known abstraction gap between requirements analysis and specification and the modeling and design of the systems architecture. Figure 3 illustrates this fact with an example form a Digital Video Recorder project of the authors [30], that aimed at product line development methods. The abstraction level of requirements and design artifacts is very different concerning uncertainty, formalization of artifact descriptions and technical aspects. Even for software development experts it is very difficult to understand comprehensively how all artifacts are linked up. The following two approaches try to reduce this huge abstraction gap by introducing intermediate models.

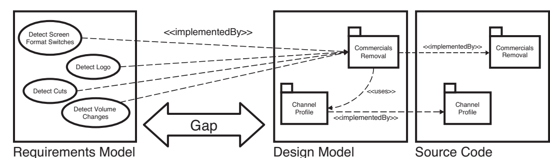


Figure 3: Abstraction Gap Between Requirements and Design

### 3.2.1 Feature Models

Originally feature models were introduced by the FODA methodology [12] for structuring domain properties from the view of costumers. Carnecki et al. [7] have extended feature models by constraints and logical relations for modeling feature dependencies. In [24] these relations are extended with multiplicities. In [26] a summary of current feature model definitions is to be found. A comparison and evaluation of methods for detecting and resolving feature dependencies is given in [5].

Feature modeling focuses on the hierarchical structuring of requirements elicited out of a problem domain. One ore more requirements are related to a feature. Feature models can be considered as starting point for architecture design. This idea is investigated deeper e.g. in [13, 20]. As a newer approach, FArM constitutes a method for mapping features to architectural elements [29]. The method FArM applies feature models for architecture development as intermediate representation between requirements and design. In this method, architectural development of components is performed by stepwise transformations of feature models, each representing a component (see figure 4). This method was especially developed for software product lines. It was applied practically in the mobile phone domain [29]. The experience showed that it is much easier to establish traceability links from a requirement to the so-called initial feature model, further on to the descendant feature models and then to the resulting design model. In the result, feature models are used as bridge between requirements and their solution.

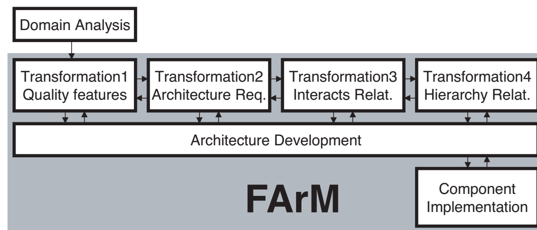


Figure 4: Sequence of Feature Model Transformation in FArM

Using traceability links from requirements to features and from features to architecture and implementation artifact could reduce significantly the above mentioned abstraction gap [23]. The establishment and validation of traceability links requires less effort with than without routing them through a feature model. A feature is then acting as intermediate artifact, as mentioned above. The effort reduction is caused by an easier identification of links between requirements and features and between features and design elements, in comparison to the one between requirements and design elements directly. As an additional observation in large projects, the comprehensive knowledge for both requirements and architectural issues in one person is not necessary to such extend. By splitting the links, the establishment and validation of links on both sides can be done by different people. It simplifies the connection between requirements analysis and documents of the later development phases.

Feature models can be applied for software reverse engineering and maintenance as well, as discussed in section 3.3. Because of their important role as intermediate artifacts, features will be used for traceability links (see chapter 4).

### 3.2.2 CBSP

Egyed et al. describe in [9] their method CBSP (component, connector/bus, system, property) for reducing the gap between requirements and architecture. The CBSP process consists of the following briefly described steps:

1. Identifying all architecturally relevant artifacts and classifying them with respect to their relevance for the different into groups C, B, S, P. In the result artifacts could occur as member of more then one group.

2. Identifying and determining of dependencies between negotiation artifacts.

3. Splitting of complex negotiation artifacts into atomic CBSP artifacts, atomic artifacts should remain member in only one group. This is especially necessary for artifacts that are part of more then one group.

4. Reducing the number of artifacts by eliminating replaced and merging related artifacts

The process leads from a negotiation rationale view to the architecture relevant CBSP view, representing artifacts from different abstraction levels. This view does not present all artifacts and all relations; despite of this the CBSP view captures and relates significant architectural elements and can be considered as bridging level between requirements and architecture. Using the CBSP view leads to an easier understanding of systems for architects and supports the consistency check between requirements and architecture. The author shows the resulting view by an example (see figure 5).

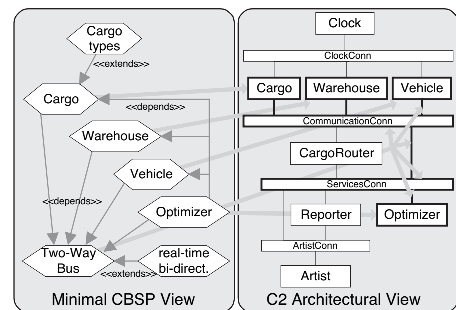


Figure 5: CBSP: Architecture Relevant Artifacts [9]

For supporting evolution the authors define trace requirements. Traceability links have to be established among elements within each abstraction level (requirements, architectural elements, design elements) and between the different abstraction levels using the CSCP model.

This method provides valuable contribution for supporting evolutionary development. Especially the distinction between views according to the stakeholders supports changes. The issues mentioned in the conclusion of [9] have to be elaborated further to be able provide tool support.

Our own observations confirm the advantages of this approach. Contributions to improved tool support are discussed in section 4.

### 3.3 Traceability for Legacy Code Comprehension

#### 3.3.1 Establishing Traceability Links Between Code and Documentation

Especially for reengineering and reverse engineering of legacy systems, the requirements and/or their links to the solution are frequently invalid. Code comprehension is necessary to understand the intentions. Traceability links can be used to store the resulting knowledge.

Antoniol et al. discuss the automated generation of traceability links in several articles.

In [2] the authors describe an approach for the automated linking of two software releases. To achieve this, both code versions are transformed into a kind of class diagram via the abstract object language (AOL). Within the diagram differences between both versions (e.g. added, deleted and modified classes and methods) are graphically displayed.

In [3] Antoniol et al. discuss the problem of subsequent establishing traceability links between documentation and code components. Therefore, they have chosen the approach of information retrieval.

Starting from the assumption that developers use their application-domain knowledge for naming identifier in code artifacts, the authors have chosen two different methods to generate connections between these artifacts and free text documents.

In the first step, both sources will be prepared. The text documents are split in separate words, capital letters are converted to lowercase, stop words are removed and by applying a morphological analysis the singular, infinitive form of the remaining words is chosen. The code-components are treated quite similar. Identifier are extracted and if necessary separated in single words. The rest of the preparation is similar to this for words from free text documents.

The code is processed in a component-based way. From every component a query is generated to all before indexed text documents. The similarity between a component and a document is measured while applying one of two different classifiers. The first classifier is computed by a probabilistic approach from the probability for the similarity between a code-component and a document. As alternative a vector space model (VSM) is used. This model measures the the distance between document and query in vector representation.

The presented method was implemented in a prototype tool. In the prototype object-oriented classes are used as software-components. Only the mnemonics of classes, attributes, methods and parameters are used to build a query. Comments are not used for identification of links so far. The described text preparation is applied semi-automatically, using language tools and thesaurus.

Based on a case study, the method has been applied using both classifier for comparing their single results. For the rating the metrics precision and recall are used. While the values for recall are very good (near 100%) the values for precision are very low, mostly 3 to 15%.

Marcus and Maletic investigate in [15] another approach for automated generation of links between documentation and code. In contrast to the work discussed before, the authors use latent semantic indexing (LSI). LSI is based on the already mentioned vector space model and includes additionally: "aspects of the meanings of words and passages reflective in their usage".

To evaluate their method the authors use the same examples as Antoniol et al. in [3] and give the reader the chance by this to compare both studies. The authors conclude, that their method achieves at least comparable results for recovering traceability links. As additional advantages of their method they mention firstly the lower effort for preprocessing, resulting in less computation time. The second advantage consists in the independence of a special natural language, programming language or paradigm. However, the results in term of precision and recall are similar to the ones mentioned above.

#### 3.3.2 Connecting Requirements, UML Artifacts and Code

Settimi et al. compare in [28] different information retrieval methods for the automated establishing of links between requirements, UML artifacts, test cases and code.

Similar to the previous cited works, a preprocessed extracted text is applied. Stop words are removed and the remaining keywords are reduced to their roots by removing pre- and suffixes.

The authors describe two different methods for retrieving traces: firstly the already mentioned vector space model, and secondly an enhanced version of it, which ranks the relevance of the results while using a pivot normalization weighted score. Additionally, both methods are evaluated using a general thesaurus.

The shown results are quite similar to these of Antoniol et al. [3]. It was possible to reach a recall of more than 90%, but with precision of 10–15% (i.e. nine of ten retrieved links are incorrect). The authors speak also about a higher recall for requirements to UML artifacts than to code artifacts in their study. They suggest to retrieve UML artifacts as intermediate elements between requirements and code.

#### 3.3.3 Traceability Links and Features for Hypothesis-Based Reverse Engineering

Recent research activities of the authors of this paper have made use of traceability links for program comprehension and architecture recovery in industrial projects. Pashov et al. [18, 19] apply features as intermediate artifacts for reducing the gap between legacy code and changed requirements. Features are utilized as media carrying hypotheses for architectural artifacts, with relations between code and features built by traceability links. Identifiers are analyzed using information retrieval techniques, similar to the approaches investigated above. In [17] Pashov describes the application of traceability links in a large industrial refactoring project. To enable tool interaction in heterogeneous environments, the links are stored in database-like cross reference tables to relate requirements, features and legacy code components.

The experience has shown that traceability links can be applied very usefully in reverse engineering activities, if an effective integration between the different types of artifacts can be reached.

#### 3.3.4 Query Based System

Tryggeseth and Nytrø present in [32] a method to dynamically retrieve links. The work of the authors is based on the assumption that static traceability consumes a high amount of work for generation and maintenance and is often still not fine grained enough.

As a consequence, the authors suggest the dynamic establishing of links based on concrete queries which are directed to the system model. Different link types are identified and and query mechanisms to retrieve these links are developed.

### 3.3.5 Code Annotations

For the evolutionary changes of software systems the recovery of former design decisions is an essential issue. There are approaches for establishing code annotations as semi-formal expressions concerning the intentions of a developer. Templates for annotations are proposed to enable and simplify a later recovery. If such annotations could be established consistently, change activities would be much easier. However, the current state of the practice of code documentation by comments discourages such approaches which are demanding much severe discipline of the developers [16, 1].

Within prototype projects our own experience approved the value of such annotations. Among others, they have been used successfully for recovering design patterns within existing source code [31].

## 3.4 Maintenance of Traceability Links

Even the best way for establishing and managing traceability tends to fail to keep pace with the evolution of a system. The result is a gradual erosion of the traceability structure leading to a loss of reliable representation of the state of connections between artifacts. The following work gives an example on how to cope with the effort, necessary to maintain traceability.

### 3.4.1 Event Based System

Cleland-Huang et al. develop in [6] an event based traceability system (EBT) to maintain links between requirements. Requirements evolution is regarded as a series of change events.

If such a change event occurs, a message is published to all dependent objects. Changes of all types are fed back to the primitive steps create, inactivate, modify, merge, refine, decompose and replace. After a change of a requirement, an event trigger automatically generates a message. It includes the type of change, the ID and description of the affected requirement and links to rationale and stakeholder involvement.

Links are established by using the publish-subscribe paradigm. The importance of each link is defined by a so called link strength. According to this strength, every single event is prioritized. Depending on the type of change and the affected objects, a message defines the task that must be performed to update the link.

Furthermore, two mechanisms are explained for resolving changed, indirect dependencies. An indirect link is a connection to an object with a dependency to a changed object. The first mechanism to handle these links is called “lazy notification”. Using this way, dependent artifacts are excluded from notification, if they are connected to intermediate artifacts. The second mechanism is called “pessimistic notification” and requires all indirect dependent artifacts to update their link. EBT allows a forward and backward traceability by use of a recursive query mechanism.

This approach was originally developed for requirements traceability, however it could easily be adopted to all software development artifacts. Especially the priority-based mechanisms proved to be useful for the management of incomplete or uncertain information in models.

The priorities have been applied successfully to control the propagation of a change throughout the network of related development artifacts.

## 4 Strategies for further development

For a comprehensive support for evolutionary development the integration of traceability links constitutes an essential issue. Traceability links have to be established as incorporated part of each model and of each engineering activity. Therefore, three main fields require future research:

- Definition of traceability links within every relevant model, both at syntactical and semantic level, and their introduction into repositories,
- Integration of establishing and updating steps of traceability links to every activity of every engineering method, for forward and reverse engineering,
- Definition of rules and criteria for checking traceability links for correctness, consistency and completeness.

For the definition of traceability links in terms of syntax and semantics there are already various proposals, e.g. [22]. However, to reach an integration into repositories and tools, a consolidation and an agreement is necessary, driven by the needs of evolutionary development. The definition to be established has to be as simple as possible, however with some rigor to enable rule-based checks. The effort for establishing and maintaining traceability links depends on the distance they have to bridge. As a conclusion from industrial refactoring projects [18, 17] the effort depends on the distance concerning the level of abstraction and the developer’s area of competence. Intermediate models and artifacts – e.g. the feature model, see section 3.2 – help to reduce effort significantly. For semi-formal means of description, concepts stored in a glossary or thesaurus can serve as intermediary artifacts and help to shorten the gaps, traceability links have to bridge.

Traceability links have to be integrated into engineering methods in a way, that every activity performs the appropriate changes to the concerned links. The links have to be updated in such a way, that their consistency and correctness is maintained. To perform an integration to *all* engineering methods and their activities, it is necessary to decompose all activities into elementary, atomic steps. These atomic steps are then enhanced by traceability link update activities. Figure 6 illustrates this way of maintaining the consistency of traceability links. By developing traceability changes for an comprehensive set of atomic change steps, all engineering methods can be enhanced easily.

There are approaches for analyzing development activities for elementary, atomic steps, e.g. Baldwins Modular Operators [4] or Potts’ and Takahashi’s Types of Change [21]. The enhanced atomic steps can them be implemented within tools for design and development, assuring consistent traceability links within the tool’s repository. It can already be foreseen that this tool support will reduce efficiently the effort for the traceability link maintenance. This expectation is based on the benefits from refactoring tools supporting elementary steps of code refactoring, because they follow a basic idea very similar to the way the atomic steps mentioned above support link updates. Even if a part of the update decisions have to be performed by a developer, tools can propose alternatives thus reducing the design space.

The propagation of changes within the set of traceability links of a repository represents another important issue. State of the Art methods – e.g. the Event Based Traceability System EBT

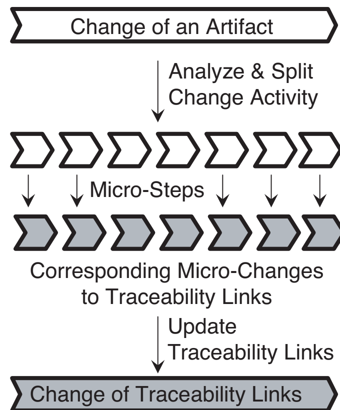


Figure 6: Maintaining Consistency of Traceability Links by Atomic Change Steps

(discussed in section 3.4) – contribute valuable solutions for managing incompleteness, inconsistency and uncertainty of the models. The use of priorities and the propagation mechanisms help to control and to reduce the overhead for updates and validations of the links. The different approaches for connecting artifacts of different types via traceability links as mentioned in sections 3.2 and 3.4 facilitate the maintenance of traceability links between different types of artifacts – e.g. requirements, UML elements, code elements – and for different design methodologies and development activities.

Consistency, correctness and completeness of the traceability links constitute prerequisites for their utilization. Though some of the atomic steps of link updates mentioned above will assure these conditions, a large part of the development artifacts is not yet defined formally enough for automatic updates. Therefore, links have to be checked permanently.

According to the different degrees of rigor and abstraction of the artifacts, checks of different types are required:

- For formally defined artifacts e.g. state transition models or source code, the concerned traceability links can be checked for validity by rules.

- For artifacts with a semi-formal description like most design elements, rules can be applied only partly. Checks for semantic correctness have to be performed by humans, because associations to concepts and terms are necessary. The use of intermediate models as mentioned in section 3.2 will reduce the effort for checks significantly by reducing the complexity of this task.

- For models with a lower degree of formality, e.g. requirements descriptions by use case templates, checks can only be based on the evaluation of natural-language expressions. The use of glossaries, thesauri and linguistic methods can provide support. Checks for this kind of artifacts will result in hints and suspicions, similar to the so-called bad smells for code refactoring [10].

Checks of the last type can make use of important works in the field of requirements traceability [22]. Furthermore, glossaries, thesauri and linguistic methods can be used to increase the degree of formality. Former works of the authors have applied these means successfully for the model-based generation of tests [25].

## 5 Conclusion

Traceability links can provide the necessary support for evolutionary development only if they are available for fine-grained artifacts. However, the establishment and the maintenance of these links has to be performed in a way that assures consistency and correctness. Furthermore, a high effort for traceability links should be prevented. In this paper, the evaluation results of various approaches are presented. This evaluation has shown, that there are already various contributions to the definition of an widely-accepted standard for traceability links. These approaches are integrated to form a roadmap how to integrate the maintenance of traceability links into any development method. Intermediate models and other artifacts reduce the effort by reducing the gaps to be bridged. Furthermore, there are valuable approaches to the application of traceability links to single development activities are investigated in a way to support both forward and reverse engineering. In the discussion of a roadmap for further work three areas are identified – the definition of traceability links, the integration of update techniques into development activities and link validation. These work packages have to be performed at different levels of abstraction, for different types of models and for different domains.

## Acknowledgments

This work was result of a research project partly supported by a grant from the German Research Foundation (Deutsche Forschungsgemeinschaft DFG) under project id Ph49/7-1.

## References

[1] J. Aldrich, V. Kostadinov, and C. Chambers. Alias annotations for program understanding. In C. Norris and J. J. B. Fenwick, editors, *Proceedings of the 17th ACM conference on Object-oriented programming, systems, languages, and applications (OOPSLA-02)*, volume 37, 11 of *ACM SIGPLAN Notices*, pages 311–330, New York, Nov. 4–8 2002. ACM Press.

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

[3] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Trans. Software Eng*, 28(10):970–983, 2002.

[4] C. Y. Baldwin and K. B. Clark. *Design Rules: Volume 1. The Power of Modularity*. The MIT Press, Cambridge, Massachusetts, 2000.

[5] M. Calder, M. Kolberg, E. H. Magill, and S. Reiff-Marganiec. Feature interaction: a critical review and considered forecast. *Computer Networks*, 41(1):115–141, 2003.

[6] J. Cleland-Huang, C. K. Chang, and M. J. Christensen. Event-based traceability for managing evolutionary change. *IEEE Trans. Software Eng*, 29(9):796–810, 2003.

[7] K. Czarnecki and U. W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Boston, 2000.

[8] G. Ebner and H. Kaindl. Tracing all around in reengineering. *IEEE Software*, 19(3):70–77, 2002.

[9] A. Egyed, N. Medvidovic, and P. Grünbacher. Refinement and evolution issues in bridging requirements and architecture - the CBSP approach, May 2001.

[10] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison Wesley, 1999.

[11] O. C. Z. Gotel and A. C. W. Finkelstein. An analysis of the requirements traceability problem. In *First International Conference on Requirements Engineering (ICRE'94)*, pages 94–101, Colorado Springs, Apr. 1994. IEEE Computer Society Press.

[12] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Nov. 1990.

[13] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. FORM: A feature-oriented reuse method with domain-specific reference architectures. *Ann. Software Eng*, 5:143–168, 1998.

[14] P. Letelier. A framework for requirements traceability in UML-based projects. In *Proceedings of 1st International Workshop on Traceability in Emerging Forms of Software Engineering, In conjunction with the 17th IEEE International Conference on Automated Software Engineering*, Edinburgh, UK, Sept. 2002.

[15] A. Marcus and J. I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *Proceedings of the 25th International Conference on Software Engineering (ICSE-03)*, pages 125–137, Piscataway, NJ, May 3–10 2003. IEEE Computer Society.

[16] J. Niere, W. Schäfer, J. P. Wadsack, L. Wendehals, and J. Welsh. Towards pattern-based design recovery. In *Proceedings of the 24th International Conference on Software Engineering (ICSE-02)*, pages 338–348, New York, May 19–25 2002. ACM Press.

[17] I. Pashov. *Feature Based Method for Supporting Architecture Refactoring and Maintenance of Long-Life Software Systems*. PhD thesis, Technical University Ilmenau, Germany, 2004.

[18] I. Pashov and M. Riebisch. Using feature modeling for program comprehension and software architecture recovery. In *Proceedings 11th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS2004)*, pages 406–418, Brno, Czech Republic, May 2004.

[19] I. Pashov, M. Riebisch, and I. Philippow. Supporting architectural restructuring by analyzing feature models. In *CSMR*, pages 25–36, 2004.

[20] I. Philippow, M. Riebisch, and K. Böllert. The Hyper/UML approach for feature based software design. In O. Aldawud, M. Kandé, G. Booch, B. Harrison, D. Stein, J. Gray, S. Clarke, A. Z. Santeon, P. Tarr, and F. Akkawi, editors, *The 4th AOSD Modeling With UML Workshop. Sixth International Conference on the Unified Modeling Language UML2003*, San Francisco, USA, Oct. 2003. published online.

[21] C. Potts and K. Takahashi. An active hypertext model for system requirements. In J. C. Wileden, editor, *Proceedings of the 7th International Workshop on Software Specification and Design*, pages 62–68, Redondo Beach, CA, Dec. 1993. IEEE Computer Society Press.

[22] B. Ramesh and M. Jarke. Toward reference models of requirements traceability. *IEEE Trans. Software Eng*, 27(1):58–93, 2001.

[23] M. Riebisch. Supporting evolutionary development by feature models and traceability links. In *Proceedings 11th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS2004)*, pages 370–377, Brno, Czech Republic, May 2004.

[24] M. Riebisch, K. Böllert, D. Streitferdt, and I. Philippow. Extending feature diagrams with UML multiplicities. In *Proceedings of the Sixth Conference on Integrated Design and Process Technology (IDPT 2002)*, Pasadena, CA, June 2002.

[25] M. Riebisch and M. Hübner. Traceability-driven model refinement for test case generation. In *Proceedings 12th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2005)*, pages 113–120, Greenbelt, Maryland, USA, Apr. 2005. Computer Society.

[26] M. Riebisch, D. Streitferdt, and I. Pashov. Modeling variability for object-oriented product lines. In F. Buschmann, A. P. Buchmann, and M. Cilia, editors, *Object-Oriented Technology. ECOOP 03 Workshop Reader*, LNCS 3013, pages 165–178. Springer, 2004.

[27] J. Sametinger and M. Riebisch. Evolution support by homogeneously documenting patterns, aspects and traces. In *Proceedings 6th European Conference on Software Maintenance and Reengineering, Budapest, Hungary, March 11-13, 2002*, pages 134–140. Computer Society Press, 2002.

[28] R. Settimi, J. Cleland-Huang, O. B. Khadra, J. Mody, W. Lukasik, and C. DePalma. Supporting software evolution through dynamically retrieving traces to UML artifacts. In *Proc of the 7th Int. Workshop on Principles of Software Evolution*, pages 49–54, Washington, DC, USA, 2004. IEEE Computer Society.

[29] P. Sochos, M. Riebisch, and I. Philippow. The feature-architecture mapping (farm) method for feature-oriented development of software product lines. In *Proceedings 13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS06), Potsdam, Germany*. IEEE Computer Society, 2006.

[30] D. Streitferdt. *Family-Oriented Requirements Engineering*. PhD thesis, Technical University of Ilmenau, 2004.

[31] D. Streitferdt, C. Heller, and I. Philippow. Searching design patterns in source code. In *Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC); Edinburgh; July 2005*, pages 33–34, 2005.

[32] E. Tryggeseth and Ø. Nytrø. Dynamic traceability links supported by a system architecture description. In *Proceedings: 1997 International Conference on Software Maintenance*, pages 180–187. IEEE Computer Society Press, 1997.

[33] M. V. Zelkowitz, A. C. Shaw, and J. D. Gannon. *Principles of Software Engineering and Design*. Prentice Hall, 1979.