Modeling Traceability in Software Development: A Metamodel and a Reference Model for Traceability

Bruno Azevedo and Mario Jino
School of Electrical and Computer Engineering, University of Campinas, Brazil

Keywords: Traceability, Traceability Model, Software Quality, Requirements Engineering, Software Development.

Abstract: Many traceability models lack well-defined traceability link types, provide incomplete coverage of situations, do not provide mechanisms to ensure consistency of traceability, and consider only requirements traceability ignoring other activities of development. We propose a set of basic concepts for traceability, a reference model, and a comprehensive metamodel for traceability created using this reference model. The reference model defines: basic elements for traceability, basic actions to be done on artifacts, basic properties that sets of link types and artifact types should have, basic categories that should be realized regarding these sets, and basic set of processes for traceability. The metamodel is composed of a visual model defining how its elements interact, the definition and semantic description of link types and artifact types which realize the categories of the reference model, and a set of detailed processes describing the steps to maintain traceability and system consistency. Our proposal aims to reduce the problems identified; the reference model provides directions to help the creation, or evaluation, of a traceability model; the metamodel provides semantically described traceability link types, coverage of the most common situations, mechanisms to ensure consistency of traceability, and covers the most common activities in software development.

1 INTRODUCTION

Traceability is the ability to keep track of elements of a system throughout its life cycle (Gotel and Finkelstein, 1994); this involves the knowledge of the origins of each element and the rationale for its existence. Elements relate to other elements in many ways; to add traceability information is to expose such relations.

Traceability is advantageous for both the business and the development sides of software development: it facilitates the alignment of the customers needs with the product, enables the correct assessment of the impact of changes, satisfies compliance for regulatory standards by delivering audit-ready products, avoids loss of design decisions by keeping development history, avoids dependency issues by preventing the removal of necessary elements, enables the identification of the actors involved in actions performed in the system (accountability), among many other benefits. Succintly, traceability allows the generation of a better quality product.

Traceability in software development has been a focus of interest for at least forty years. Several authors have contributed to a better understanding of traceability: Ramesh and Edwards (Ramesh and Edwards, 1993) highlight several important issues in the subject; Gotel and Finkelstein (Gotel and Finkelstein, 1994) analyze the problem, identifying the need for pre-requirements traceability; Pfleeger and Bohner (Pfleeger and Bohner, 1990) propose traceability graphs and metrics for vertical and horizontal traceability in the context of impact analysis; Goknil et al. (Goknil et al., 2011) provides a formal approach to the definition of traceability links; Mäder et al. (Mäder et al., 2009) highlight a few basic concepts which traceability models should take into account; among many others.

Still, the adoption of traceability by the industry is hindered by the lack of consensus on how and what should be traced. Traceability models aim to solve this issue by providing a standard to be used as a guide in software development projects.

1.1 Motivation

Based on an extensive literature review we have perceived that traceability models for software development: (i) lack well-defined traceability link types, i.e., a traceability link type is shown, or named, but never described semantically, making difficult or impossible to use; (ii) provide incomplete coverage of situations, i.e., link types do not model the most common

Azevedo, B. and Jino, M.
Modeling Traceability in Software Development: A Metamodel and a Reference Model for Traceability.
DOI: 10.5220/0007715103220329
In *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2019)*, pages 322-329
ISBN: 978-989-758-375-9

relations such as dependency, accountability, or authorization; (iii) do not provide mechanisms to ensure consistency of traceability, i.e., processes are not provided to keep traceability and system consistency after system changes; (iv) and consider only requirements traceability, ignoring other activities of the development process such as design or implementation, i.e., artifact types do not model the most common artifacts. A comprehensive model for traceability is needed to address these issues.

1.2 Contribution

Given the deficiencies discussed in the previous section, we propose: (i) a reference model for traceability, and (ii) a metamodel for traceability created using (i). The first part aims to provide a better understanding of traceability and allows the creation and evaluation of traceability models; the second part provides a semantically described metamodel for traceability in software development projects.

A set of basic concepts for traceability is proposed and a reference model is built on top of this conceptual framework; our reference model defines the basic elements for traceability, the basic actions to be done on artifacts, the basic properties that sets of link types and artifact types should have, the basic categories that should be realized regarding these sets, and the basic set of processes for traceability.

Using the reference model as a basis, we created a traceability metamodel composed of: a visual model defining how the elements interact, the definition and semantic description of link types and artifact types which realize the categories of the reference model, and a set of processes detailing the steps to maintain traceability and system consistency.

This contribution aims to solve the problems we identified in our literature review, mentioned previously. Our reference model provides directions to help the creation of a traceability model without these problems and our metamodel provides semantically described traceability link types, provides coverage of the most common situations, provides mechanisms to ensure consistency of traceability, and takes into account the most common activities in software development.

1.3 Structure

The paper is structured as follows. In Section 2, we summarize the most closely related papers we found in the literature. In Section 3, we propose a conceptual framework and present our reference model for traceability built using these concepts. In Section 4, we provide our metamodel for traceability built using the reference model shown previously. In Section 5, we conclude the paper and discuss ideas for future work.

2 RELATED WORK

A literature review was performed to find out the current state of research in modelling traceability for software development. We looked for papers proposing traceability models, traceability link types, artifact types, and processes for traceability, plus conceptual papers. We found a total of 223 papers of direct interest; we performed a deeper inspection on these papers to study the concepts and elements for traceability found in the current state of the art. These were used as the starting point to create the reference model. From this selection, the following 23 papers were considered the most relevant to the work discussed in this paper; these propose traceability concepts, models, links, and applications to specific domains.

Concepts and important issues to be taken into account when developing a model for traceability are highlighted in (Ramesh and Edwards, 1993) and (Mäder et al., 2009).

Many works in this selection contain models for traceability: a model derived from a real case scenario is presented in (Ramesh et al., 1995b); empirical approaches were used in (Ramesh et al., 1995a) and (Ramesh and Jarke, 2001) to create models; a case study was used to develop a model integrating traceability and Software Configuration Management in (Mohan et al., 2008); a model integrating different modeling techniques is described in (Berenbach and Wolf, 2007); in (Dermeval et al., 2013), a model relating the requirements model to the architecture model is proposed; a model and an approach for automatic generation of link types is described in (Jirapanthong and Zisman, 2005); a traceability management method for modelling traceability with Multi perspectives View is proposed in (ghazi, 2008); a feature-oriented traceability model for software product line development is presented in (Shen et al., 2009); a four step traceability management process having four steps and a model for traceability are proposed in (Haidrar et al., 2016); in (Dubois et al., 2010), a model for requirement traceability linking the requirement model, the solution model, and the V&V model is described; a model which integrates textual specifications with UML specifications is proposed in (Letelier et al., 2005); a model-driven approach for supporting the evolution of design decisions is described in (Malavolta et al., 2011); a link model for the Unified Process and a set of link verification rules are proposed in (Maeder et al., 2007). In (Goknil et al., 2008), a metamodel having a set of

formalized link types is proposed and is used in subsequent work to support different goals: consistency checking (Goknil et al., 2011), reasoning about requirements (Goknil et al., 2013), validation of traces between requirements and architecture (Goknil et al., 2014a), and change impact analysis (Goknil et al., 2014b).

A couple of works propose frameworks, given specific contexts: a framework containing mechanisms for model slicing and a model for traceability, and a view-based model-driven framework supporting semi-automatically eliciting and semi-formalizing trace dependencies among process development artifacts are presented in (Nejati et al., 2012) and (Tran et al., 2011), respectively.

3 A REFERENCE MODEL FOR TRACEABILITY

In this section we propose a conceptual basis for traceability and a reference model built on top of this basis. We identify the basic elements of traceability; propose the set of basic actions that should be considered when tracing actions on artifacts; propose three basic properties that sets of traceability link types and artifact types should have; propose sets of categories for link types and artifact types, respectively, that should be satisfied, and propose the set of basic processes needed to mantain traceability and system consistency.

3.1 Basic Elements in a Traceability Model

There are three basic elements to be considered when tracing software; these are: artifacts, links and processes. The model depicted in Figure 1 show how these elements relate to each other.

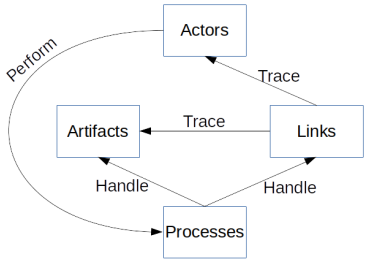

Figure 1: Conceptual model.

Artifacts are the workproducts of the software development process (Tekinerdoğan et al., 2007). Actors are representations of agents interacting with the software system. Links represent relationships between artifacts and actors; links can be classified into different types, each one having a specific semantics concerning the relationship between the elements it connects. Processes aim to maintain consistency of traceability information in the software system; whenever an action happens, a process provides the set of steps needed to maintain consistency.

3.2 Basic Actions Regarding Artifacts

We define six basic actions that should be taken into account when tracing actions on artifacts. Creation is the action through which an artifact comes into existence. Modification is the action of changing an artifact. Removal is the action of removing an artifact from the working project. Homologation is the action of adding an artifact to the working project. Decomposition is the action of decomposing an artifact into two, or more, artifacts. Application is the action of executing a procedure, defined in an artifact, on another artifact.

3.3 Properties of Link Types and Artifact Types

We propose three basic properties which the set of link types and the set of artifact types in a traceability model should have; these are Comprehensiveness, Specificity, and Coverage. Specifically, the set of link types should have the properties of Comprehensiveness, Specificity, and Coverage. The set of artifact types should have the properties of Comprehensiveness and Specificity.

3.3.1 Comprehensiveness

Comprehensiveness is the property of taking into account a wide range of situations. For instance, a set of link types having Comprehensiveness is able to model the most common relationships between elements in a project.

3.3.2 Specificity

Specificity is the property of being capable of expressing particular situations. For instance, consider a set of link types having only one link type named 'Trace' which will represent all relationships in a project. This set has Comprehensiveness but fails to characterize each specific situation it is modelling. It is able to model both a dependency and an accountability relationship, but we are unable to identify which one is being modelled; therefore there is loss of traceability information.

3.3.3 Coverage

Coverage is the property in which the set of link types cover all the artifact types they should cover. For instance, a set of link types having a link expressing authorship of artifacts should cover all artifact types in the model; there should not be an artifact type which it is not possible to identify its author.

3.4 Basic Categories for Link Types

We propose seven basic categories in which link types may be classified into: Accountability, Constraint, Permission, Characterize action, Evolution, Action outcome, and Composition. These categories cover the link types found in the literature which we identified as essential.

A model should satisfy these categories, i.e., there should be link types embodying the roles described by each category.

A brief summary of the categories is shown next.

3.4.1 Accountability

Link types within this category are used to assign authorship to actions. The subset of link types in this category should have link types that assign authorship to at least the six basic actions previously defined.

3.4.2 Constraint

Link types within this category are used to convey that one artifact constrains another artifact. The subset of link types in this category should have link types that convey at least the following roles: that one artifact is required by another artifact and that one artifact is in conflict with another artifact.

3.4.3 Permission

Link types within this category are used to convey information on authorization to perform actions concerning artifacts. The subset of link types in this category should have link types that convey information on authorization to perform at least the six basic actions.

3.4.4 Characterize Action

Link types within this category are used to convey the rationale and/or the description (who, what, and how) concerning an action. The subset of link types in this category should at least have link types – for each of the six basic actions – that: (i) justify and describe an action, (ii) solely justify an action, and (iii) solely describe an action.

3.4.5 Evolution

Link types within this category are used to convey that information is changed and propagated from one artifact to another. The subset of link types in this category should at least have link types that: (i) convey that an artifact was refined into other artifacts, and (ii) convey historical information on a given artifact, i.e., to enable version control of artifacts.

3.4.6 Action Outcome

Link types within this category are used to convey the outcome of an action, e.g., a set of test cases applied on a program produces a test log containing the results of the test.

3.4.7 Composition

Link types within this category are used to convey the information that an artifact was, or is, part of another artifact. The subset of link types in this category should at least have link types that: (i) convey that an artifact was decomposed into two or more artifacts, and (ii) convey that an artifact is part of another artifact.

3.5 Basic Categories for Artifact Types

We considered the most common activities in software development (IEEE Computer Society et al., 2014) to define the basic categories for artifact types: Pre-requirements, Requirements, Design, Implementation, Verification & Validation and Testing, and Rationale. There are development approaches which do not use all these activities, such as agile approaches; in this case, a user may ignore the categories which model the unused activities.

The activities considered are well known with the exception of Rationale.

3.5.1 Rationale

Artifact types within this category are created to justify and/or describe actions concerning artifacts. The subset of artifact types in this category should have artifact types that provide rationale for at least the six basic actions.

3.6 Basic Processes

The purpose of a traceability process is to maintain traceability and system consistency. Consistency may be broken by actions performed throughout the development process. For instance, suppose an artifact α

is dependent on another artifact β. If β is removed, the system becomes inconsistent because α has an unsolved dependency; the traceability information also becomes inconsistent because α has a traceability link to a nonexistent artifact. Hence, processes are necessary to correct inconsistencies caused by actions.

Given the set of basic actions which may affect a system, a model should at least have a process for each of the six basic actions defined in Section 3.2.

3.7 Why a Model for Traceability Should Have these Characteristics?

We argue that a model for traceability lacking these characteristics is incomplete and, we believe, this is self-evident in most cases. For instance, a model lacking Accountability link types can not record who modified or created an artifact; a model having specificity but lacking comprehensiveness will fail to represent relations between elements in a project; a model without artifacts to represent requirements can not trace such artifacts; etc.

4 A METAMODEL FOR TRACEABILITY

In this section we present our metamodel for traceability constructed using the reference model defined in Section 3. It is composed of a visual model defining how the elements interact, 57 link types semantically described, 12 artifact types, and 7 detailed processes to maintain traceability and system consistency during changes. Each of these sets satisfy the categories and properties of the reference model.

4.1 Traceability Space

We organize the elements of our metamodel in a Traceability Space composed of four subspaces. The Actors Space contains abstractions of the agents that interact with the project. The Rules Space contains the rules defining the actions which can be performed by the actors. The System Space is the collection of products of the development process. The Processes Space contains the set of processes used to maintain consistency. Links and artifacts are also part of the Traceability Space; these are instances of link types and instances of artifact types, respectively. Link types connect: (i) artifacts to other artifacts within the System Space, and (ii) artifacts in the System Space to actors in the Actors Space.

An example Traceability Space is depicted in Figure 2. It shows an actor who performed the Modification Process on artifact α. Permission to do so was assigned by the Rules Space. The resulting Accountability links connect the actor and the artifact. Constraint links connect artifacts α and β.

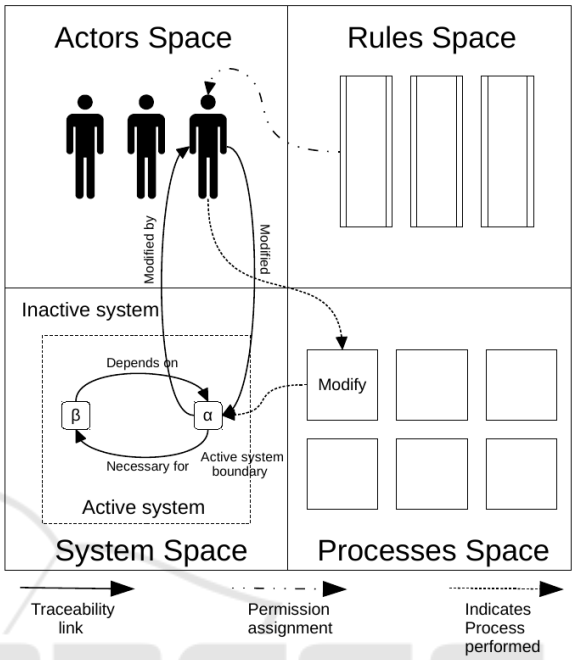

Figure 2: A Traceability Space.

4.2 Link Types

We have created 57 link types satisfying the roles of the seven categories for link types of the reference model. Each link type was semantically described to: (i) enable its mapping to real life projects and (ii) avoid ambiguities during its use. Table 1 shows the complete set of link types arranged by category.

Two link types are shown next to give a general idea of how each link type in the table was described.

4.2.1 Link Type ReifedFrom

The link type ReifiedFrom, an Evolution link type, has the following description:

Definition 4.1. *An artifact α is reified from an artifact β if part of, or all of, α was constructed in a structured way from all of, or part of, β. Implicit information from β may be made explicit in α and/or new information may be added.*

Let α and β be two artifacts and let *e* be a link starting in α and having β as its destination. The link *e* is a *ReifiedFrom* link type if it conveys the information that α was reified from β.

For instance, let α_1 and α_2 be requirements and let β be a design artifact. Let all of α_1 and part of α_2 be

Table 1: Categories × Link types.

Accountability	HomologatedBy, Homologated, RejectedBy, Rejected, CreatedBy, Created, ModifiedBy, Modified, AppliedBy, Applied, DecomposedBy, Decomposed, RemovedBy, and Removed.
Constraint	DependsOn, NecessaryFor and ConflictsWith.
Permission	MayHomologate, MayBeHomologatedBy, MayModify, MayBeModifiedBy, MayRemove, MayBeRemovedBy, MayDecompose, MayBeDecomposedBy, MayApply, MayBeAppliedBy, MayActivate, and MayBeActivatedBy.
Characterize Action	DefinesModificationOf, ModificationDefinedBy, DefinesDecompositionOf, DecompositionDefinedBy, DefinesActivationOf, ActivationDefinedBy, DefinesRemovalOf, RemovalDefinedBy, JustifiesHomologationOf, HomologationJustifiedBy, JustifiesRejectionOf, RejectionJustifiedBy, DefinesCreationOf, CreationDefinedBy, DefinesApplicationOf, ApplicationDefinedBy, SubjectToApplicationOf, and AppliesTo.
Evolution	ReifiedFrom, ReifiedInto, ModifiedFrom, and ModifiedInto.
Action Outcome	ApplicationProduces, and ProducedByApplicationOf.
Composition	DecomposedFrom, DecomposedInto, PartOf, and ComprisedOf.

used in the construction of β, i.e., information written in natural language from α_1 and α_2 was transformed into information written in a design representation language in β. Two *ReifiedFrom* link types starting in β and having α_1 and α_2 as its destination models this relation.

4.2.2 Link Type DependsOn

The link type DependsOn, a Constraint link type, has the following description:

Definition 4.2. *An artifact* β *satisfies an artifact* α *if a condition required by* α *is fulfilled by* β.

Let α and β be two artifacts and let *e* be a link starting in α and having β as its destination. The link *e* is a *DependsOn* link type if it conveys the information that β satisfies α.

There are several types of dependencies, such as: the application of an artifact may depend on the previous application of other artifacts (dependency of previous actions) or an artifact may depend on characteristics of another artifact to run correctly (existencial dependency).

4.3 Artifact Types

We have defined 12 artifact types satisfying the six categories for artifact types of the reference model; seven artifact types for the Rationale category to allow the justification and/or description of the six basic actions plus an extra action (Activation), and one generic artifact for each remaining category: Pre-Requirements Engineering Artifact (PREA), Requirements Engineering Artifact (REA), Design Artifact (DA), Implementation Artifact (IA), Verification & Validation and Testing Artifact (VVTA), Rationale for Creation (RC), Rationale for Modification (RM), Rationale for Removal (RR), Rationale for Decomposition (RD), Rationale for Activation (RAc), Rationale for Application (RAp), and Rationale for Homologation or Rejection (RHR).

4.3.1 On Rationale Types

A rationale type provides rationale and/or the description concerning an action. A rationale is a justification (why) for an action; a description defines who, what, and how, concerning an action. An explanation on each rationale type is given below.

RCs provide rationale for the creation of an artifact but it can also provide more information about the artifact being created; for instance, an RC for an IA could explain its utility and describe how it can be used. RMs provide rationale for modification of an artifact and describes the modifications to be made. RRs provide rationale for removal of an artifact; i.e., given an artifact, an RR argues for its removal from the Active System. RDs provide rationale for decomposition of an artifact but can also define how the decomposition should be done. RAcs provide rationale for activation [1] of an artifact but it can also define modifications that should be done before it is activated. RAps provide rationale for application of an artifact but can also define how the application should be done. RHRs provide rationale for homologation or rejection of an artifact; for instance, after a failed attempt to remove an artifact, an RHR would be created explaining the reasons for the rejection of the proposed removal.

4.4 Processes

We have created 7 processes satisfying the six basic processes required by the proposed reference model plus an extra process (Activation). A very brief explanation of the Homologation Process is shown next. Permissions are assumed to be checked for actions in the process.

[1]The process of trying to activate an artifact previously rejected or removed.

4.4.1 Homologation Process

Every newly created artifact starts off in the Inactive System; it must go through the Homologation Process to become part of the Active System. Let α be an artifact which goes through the Homologation Process and $\mathcal{A}$ be the group of actors who created it. Let $\mathcal{B}$ be a group of actors disjoint from $\mathcal{A}$ which performs the homologation process on α. The group $\mathcal{B}$ evaluates the artifact α to determine whether it is added to the Active System. There are two possible outcomes: α is homologated becoming active, or it is rejected remaining inactive.

Why is the Homologation Process Useful? Homologation avoids the introduction of inconsistencies in the Active System; e.g., if the evaluation by the Homologation Process is not performed, a necessary artifact could be removed, leaving the system inconsistent. Furthermore, it allows the evaluation of the impact of an action before it is done. Therefore, the decision whether to proceed with an action may be guided through the assessment of its impact; e.g., during the homologation process it may be found that the cost of the removal of an artifact outweighs its benefits.

Issues when Homologating a Rationale for Modification or Removal. The decision to homologate a Rationale for Modification (RM) or a Rationale for Removal (RR) is contingent upon the impact in the Active System; it may be necessary to perform a sequence of actions to deal with the problems arisen from the actions of modification or removal. Hence, the homologation may not occur if the impact is undesirable.

To exemplify the role of the Homologation Process, consider the situation where an RM α is up for homologation, β is the target of α, and β has *NecessaryFor* link types starting from it; the implementation of the modification described in α could cause existing dependencies to become unfulfilled. Therefore, each *NecessaryFor* link type should be taken into account and solutions should be provided for each possible problem. The same is true for *DependsOn* link types starting from β; the modification described in α could eliminate the need for these link types, or even create new dependencies. The Homologation Process provides the steps to check for problems and its possible solutions.

4.5 Customizing the Metamodel

The metamodel may be customized according to the specific needs of a project. Some projects may need a leaner approach, choosing not to use certain link types, artifact types, and processes. Such customizations may be implemented at the expense of loss of traceability information.

5 CONCLUSIONS

We identified a series of issues with the current traceability models; to address these issues we have proposed a reference model and a metamodel for traceability. The reference model, defined by the conceptual basis, may be used in the creation of traceability approaches; it also allows the evaluation of approaches, checking whether an approach fulfills basic properties and categories of link types, artifact types, and processes. The reference model was used to develop a metamodel composed of: a Traceability Space, showing how the elements interact with each other; a set of link types and artifact types fulfilling all the categories; a set of processes detailing the steps to maintain traceability and system consistency.

The models may be customized according to the specific needs of a project; some projects may need a leaner approach, choosing not to use certain link types, artifact types, and processes.

Our proposal may be extended in several ways, as follows.

Investigate new concepts and mechanisms to enrich the conceptual basis, aimed at reducing subjectivity in traceability-related tasks.

Extend the set of link types, artifact types, and processes to cover additional domains, such as security-focused projects, or to to increase the comprehensiveness and specificity of the current sets.

Investigate additional properties of the link types to improve our proposal expressiveness. For example, the addition of different levels of dependency strength would provide more information when performing modifications on the System Space; it would also be useful for change impact analysis.

The link types and processes may be formalized using an existing language or by describing each element using first-order logic.

The development of tools to support the use of the reference model and metamodel. The reference model tool will support the evaluation of approaches and facilitate the creation of approaches. The metamodel supporting tool will provide semi-automated use of the processes. It will also be useful for impact analysis.

We are currently using the metamodel to trace the elements in a software development project.

REFERENCES

Berenbach, B. and Wolf, T. (2007). A unified requirements model; integrating features, use cases, requirements, requirements analysis and hazard analysis. In *ICGSE 2007*, pages 197–203.

Dermeval, D., Castro, J., Silva, C., Pimentel, J. a., Bittencourt, I. I., Brito, P., Elias, E., Tenório, T., and Pedro, A. (2013). On the use of metamodeling for relating requirements and architectural design decisions. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 1278–1283. ACM.

Dubois, H., Peraldi-Frati, M. A., and Lakhal, F. (2010). A model for requirements traceability in a heterogeneous model-based design process: Application to automotive embedded systems. In *2010 15th IEEE International Conference on Engineering of Complex Computer Systems*, pages 233–242.

ghazi, H. E. (2008). Mv - tmm: A multi view traceability management method. In *2008 32nd Annual IEEE International Computer Software and Applications Conference*, pages 247–254.

Goknil, A., Kurtev, I., Berg, K., and Veldhuis, J.-W. (2011). Semantics of trace relations in requirements models for consistency checking and inferencing. *SoSyM*, 10(1):31–54.

Goknil, A., Kurtev, I., and Millo, J.-V. (2013). A metamodeling approach for reasoning on multiple requirements models. In *EDOC '13*, pages 159–166, Washington, DC, USA. IEEE Comp. Soc.

Goknil, A., Kurtev, I., and van den Berg, K. (2008). *A Metamodeling Approach for Reasoning about Requirements*, pages 310–325. Springer Berlin Heidelberg, Berlin, Heidelberg.

Goknil, A., Kurtev, I., and Van Den Berg, K. (2014a). Generation and validation of traces between requirements and architecture based on formal trace semantics. *Journal of Sys. and Soft.*, 88:112–137.

Goknil, A., Kurtev, I., van den Berg, K., and Spijkerman, W. (2014b). Change impact analysis for requirements: A metamodeling approach. *Inf. & Soft. Tech.*, 56(8):950–972.

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

Haidrar, S., Anwar, A., and Roudies, O. (2016). Towards a generic framework for requirements traceability management for sysml language. In *CiSt 2016*, pages 210–215.

IEEE Computer Society, Bourque, P., and Fairley, R. E. (2014). *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. IEEE Computer Society Press, 3rd edition.

Jirapanthong, W. and Zisman, A. (2005). Supporting product line development through traceability. In *APSEC'05*, pages 9 pp.–.

Letelier, P., Navarro, E., and Anaya, V. (2005). Customizing traceability in a software development process. In *Information Systems Development*, pages 137–148. Springer US.

Mäder, P., Gotel, O., and Philippow, I. (2009). Getting back to basics: Promoting the use of a traceability information model in practice. In *TEFSE '09*, pages 21–25.

Maeder, P., Philippow, I., and Riebisch, M. (2007). A traceability link model for the unified process. In *SNPD*, volume 3, pages 700–705.

Malavolta, I., Muccini, H., and Smrithi Rekha, V. (2011). Supporting architectural design decisions evolution through model driven engineering. In Troubitsyna, E. A., editor, *Software Engineering for Resilient Systems*, pages 63–77, Berlin, Heidelberg. Springer Berlin Heidelberg.

Mohan, K., Xu, P., Cao, L., and Ramesh, B. (2008). Improving change management in software development: Integrating traceability and software configuration management. *Decision Support Systems*, 45(4):922 – 936.

Nejati, S., Sabetzadeh, M., Falessi, D., Briand, L., and Coq, T. (2012). A sysml-based approach to traceability management and design slicing in support of safety certification: Framework, tool support, and case studies. *Inf. & Soft. Tech.*, 54(6):569 – 590.

Pfleeger, S. L. and Bohner, S. A. (1990). A framework for software maintenance metrics. In *Proceedings. Conference on Software Maintenance*, pages 320–327.

Ramesh, B., Dwiggins, D., DeVries, G., and Edwards, M. (1995a). Towards requirements traceability models. In *Systems Engineering of Computer Based Systems, Proceedings of the 1995 International Symposium and Workshop on*, pages 229–232.

Ramesh, B. and Edwards, M. (1993). Issues in the development of a requirements traceability model. In *Requirements Engineering, Proceedings of IEEE International Symposium on*, pages 256–259.

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

Ramesh, B., Powers, T., Stubbs, C., and Edwards, M. (1995b). Implementing requirements traceability: a case study. In *Requirements Engineering, Proceedings of the Second IEEE International Symposium on*, pages 89–95.

Shen, L., Peng, X., and Zhao, W. (2009). A comprehensive feature-oriented traceability model for software product line development. In *Australian Software Engineering Conference*, pages 210–219.

Tekinerdoğan, B., Hofmann, C., Akşit, M., and Bakker, J. (2007). Metamodel for tracing concerns across the life cycle. In *Early Aspects: Current Challenges and Future Directions*, pages 175–194. Springer Berlin Heidelberg.

Tran, H., Zdun, U., and Dustdar, S. (2011). Vbtrace: using view-based and model-driven development to support traceability in process-driven soas. *SoSyM*, 10(1):5–29.