Traceability Types for Mastering Change in Collaborative Software Quality Management

Boban Celebic, Ruth Breu, and Michael Felderer(✉)

Institute of Computer Science, University of Innsbruck, Innsbruck, Austria
{boban.celebic,ruth.breu,michael.felderer}@uibk.ac.at

Abstract. Software is constantly evolving and to successfully comprehend and manage this evolutionary change is a challenging task which requires traceability support. In this paper we propose a novel approach to traceability as a cornerstone for successful impact analysis and change management, in the context of collaborative software quality management. We first motivate the crucial role of traceability within lifecycle management of the new generation of distributed fragmented software services. Based on the model-based collaborative software quality management framework of Living Models, we then categorize software quality management services and identify novel types of traceability. This is followed by an overview and classification of sample software quality management services from literature, enabled by the interrelation with the identified types of traceability. From this classification we derive the need for further research on traceability in collaborative software quality management.

Keywords: Collaborative software quality management · Traceability · Software change management · Software evolution

1 Introduction

In modern IT systems distributed across organizational and system boundaries the grand challenge for software quality management becomes the coordination of people, methods, processes and tools [5]. Current quality management processes and methods are not yet capable to scale up to the novel arising scenarios like cars and aircrafts communicating to each other, health records exchanged on national level or energy traded all over continents, for several reasons.

First, an integrated view of the full service life cycle from business alignment, service design to deployment and runtime monitoring is of utmost importance. This becomes immediately obvious for the quality attribute of security. Security requirements are mostly negative requirements and security vulnerabilities may originate from all parts of the service lifecycle including, for instance, organizational flaws, design failures or deficiencies in runtime configuration. The necessary integration of different processes and data, for example from IT management and software development, is hardly supported in current practice and theory.

Second, the new generation of software services is of inherent evolutionary character. Novel technologies make flexible composition of services technically feasible. However, current quality management methods and tools do not support change and evolution at a level of effectiveness which is required by the complexity of the novel application scenarios.

Finally, new business models require new kinds of quality management processes. As an example, Sneed [44] has recently pointed out the new role of testing for systems incorporating cloud services.

To successfully comprehend and manage continuous software change, stakeholders must first identify what will be affected by any proposed change - forecast and analyze its potential impacts. This implies the necessity to record and maintain the traces among artifacts to guarantee traceability. *Traceability*, as a general term in software engineering, is the ability to describe and follow the life of software artifacts [50]. This is a generalization of the requirements traceability definition of Gotel and Finkelstein [21] to arbitrary artifacts. In a model-based context, the artifacts of interest are models, conforming to one or more meta models. Traceability in this context is predominantly concerned with typed relationships (often called *trace links* or *traces*) between models and model elements - i.e., traceability needs to relate elements in a source model to the corresponding elements in a target model (in a transformation chain) and vice versa (which further implies that trace information ought to be generated as a result of model transformations). For example, trace links help with tracking which part of the code satisfies which requirements, monitoring the (implementation, test) status of requirements, or measuring coverage of artifacts by test cases. Unfortunately, impact analysis and traceability are not easy tasks, due to the complexity and size of software systems nowadays, their ever-changing artifacts with complex interdependencies, short development cycles, and numerous stakeholders involved in the process. Moreover, the information amount increases significantly over time, as the services evolve. As a result, keeping trace links synchronized is a cumbersome, time-consuming, expensive and error-prone task, often resulting in undesired confusion concerning the software product's current status.

In this paper we argue that a novel approach to traceability (as a cornerstone for successful impact analysis and, thus, software change and quality management) is a prerequisite to overcome the aforementioned challenges. This novel approach presupposes identification of novel trace types, particularly those specific to concrete environments (so to achieve the full potential and benefits of traceability use in such environments).

The paper is structured as follows. Section 2 provides an overview of the state-of-the-art. Section 3 sketches Living Models, a conceptual framework for collaborative software quality management, and identifies as well as categorizes relevant types of collaborative software quality management services. Section 4 presents the associated new types of traceability. Section 5 provides examples of beneficial use of traceability in all service categories. Finally, Sect. 6 concludes this paper.

2 State-of-the-Art

Before discussing the existing classifications of traceability, we must first justify the need for such classifications. In short, classification of traceability into types leads to: better understanding of the traceability links, their meaning and semantics [37]; better management and evaluation of trace links [37]; ability to perform automatic operations with traces [33]; and better visual representation of trace links, as a result of the possibility to customize the visualization to a specific type of traceability link.

Classification of software traceability has been addressed earlier on several occasions [45]. These classifications were based on several criteria: the types of the related artifacts, the activities connected to the relation (e.g., evolution, verification, impact analysis), the elements and properties of trace links (different tasks may require access to a specific set of links, based on their properties [29]), and others.

Besides the well-known concepts of *pre-* and *post- requirement specification*, *forward* and *backward*, *horizontal* and *vertical*, *functional* and *non-functional* traceability [50], some other, *model-specific* classifications have been proposed. The simplest taxonomy is the one with *manual* (trace link established manually in the trace model) and *automatic* (created by a tool) trace types [36]. According to *transformation traceability*, traces are produced as a result of (automatic) model transformations and indicate how source elements are related to target elements and vice versa. Galvao et al. [18] classify the traceability approaches in model-based and model-driven engineering into three categories: *requirements-driven approaches* use requirements models as abstractions to guide their traceability methods; *modeling approaches* are interested in how meta models, models and conceptual frameworks are involved in the tracing process; *transformation approaches* make use of model transformation mechanisms for generating trace information. Another taxonomy is on *post-model-specification* (i.e., traceability of models to and from various artifacts produced by their use) and *pre-model-specification* traceability [39]. A similar taxonomy is given in [50]: *pre-model-*, *intra-model-*, and *post-model-* traceability - *"denoting traceability between early artifacts and the first model, traceability between the gradually refined models, and traceability between the final model and the non-model artifacts generated or derived from it, respectively"*.

Further traceability classifications in model-based software engineering have been developed so to emphasize different attributes or characteristics of traceability [37]. More precisely, two directions for classifications can be identified in the literature: classifications that focus on *explicit* trace links (captured directly in models by using a suitable concrete syntax) and *implicit* trace links (trace information is generated as a result of an application of model management operation(s)) [37].

Problems with Existing Traceability Classifications. Unfortunately, all of the proposed classifications are difficult to evaluate and compare because there is no common level of abstraction and usually no unambiguous or formal definition of the different categories, and, additionally, the categories themselves cannot be separated clearly [50]. The classifications vary from more or less abstract, conceptual classifications to concrete ones (i.e., traceability meta models). It can be stated that the definition and the basic classifications of model-related traceability are still not agreed upon.

Furthermore, some of the more obvious open issues in model-based traceability can easily be identified by analyzing the referenced traceability classifications and types: lack of automation to cope with traceability in the early development stages in model-based approaches should be addressed, as well as mechanisms for the evolution of trace links; better trace meta models for enhancing model-based traceability should be developed; more efficient management of fine grained trace links is needed.

3 Collaborative Software Quality Management

In this section, we outline the model-based collaborative software quality management framework Living Models and related quality management services, which together provide the context for identifying novel traceability types. For a more detailed presentation of Living Models, we refer to [2].

3.1 The Living Models Framework

The scope of Living Models is not only software engineering but includes also IT management and systems operation. Accordingly, the stakeholders we envisage range from IT and security managers to system analysts, developers, testers to platform responsibles and network administrators.

In a Living Models environment all kind of information is conceptualized in a model-aware way. We distinguish functional data (like data about business processes, services, components or physical infrastructure elements [30,31]) and non-functional data (like a security requirement, a test case or a bug attached to a component [16]). We assume that each non-functional data instance is attached to a functional data instance. Our minimum requirement to model awareness is that all data instances adhere to a global meta model. The global meta model may vary in the degree of rigor, e.g., comprising both a partial meta model representing the abstract syntax of a programming language and a partial meta model capturing informally described requirements and their interdependencies. In [2] we have presented a reference meta model integrating the enterprise architecture and the software engineering domain (Fig. 1).

Living Models suggests stakeholder-centric model-based work environments. These environments support the tasks of the stakeholders involved and use models to provide concepts and information at an appropriate abstraction level. An example for a stakeholder-centric model-based work environment is a model-based testing tool supporting the tester to design, execute and evaluate tests [17],

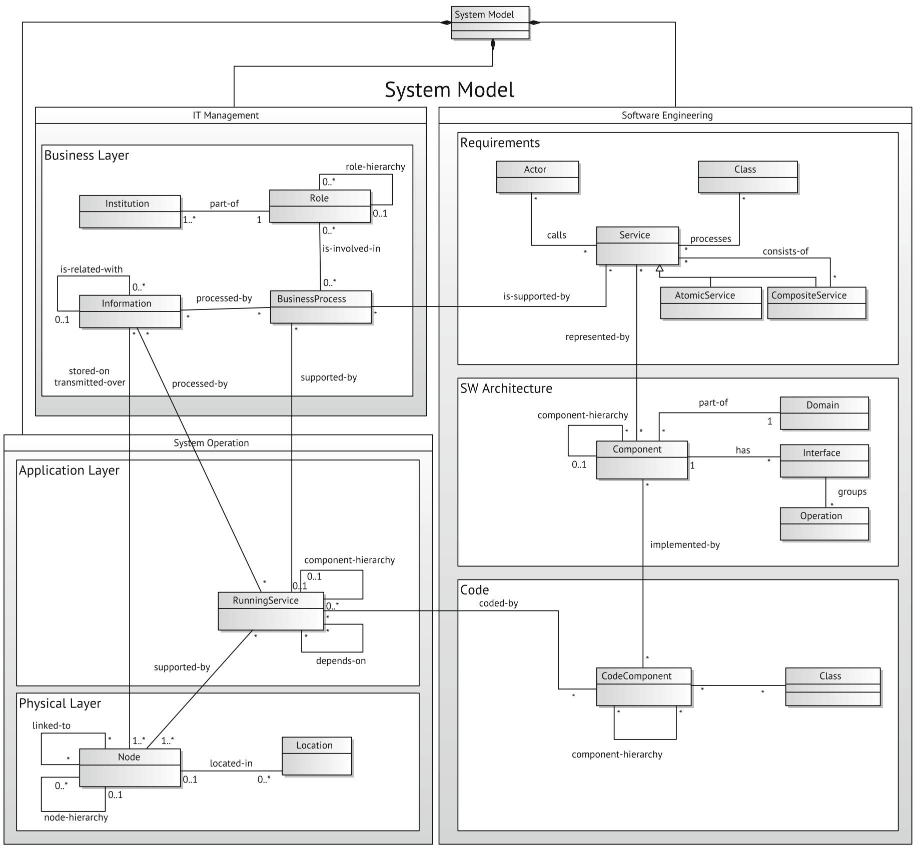

Fig. 1. Reference functional system meta model of living models

whereas IT architects may use a UML tool for modeling and analyzing a system's components.

Third, Living Models comprise close coupling of models and the runtime as a prerequisite that stakeholders can take proper decisions. The minimum requirement for this coupling is a process defining the responsibilities, the points of time when the synchronization between model and runtime takes place and the automated and manual tasks within the synchronization. Examples for such synchronization processes are model-based software development (i.e., generating runtime artifacts out of models) [32, 47] or semi-automated workflow-controlled model maintenance [13].

A fourth major principle is concerned with the collaboration of stakeholders. Living Models proposes a change-driven interaction between stakeholders. This comprises *change events* (e.g., a modified model element or a time event), *change propagation* to linked elements and *change handling* encompassing coordinated

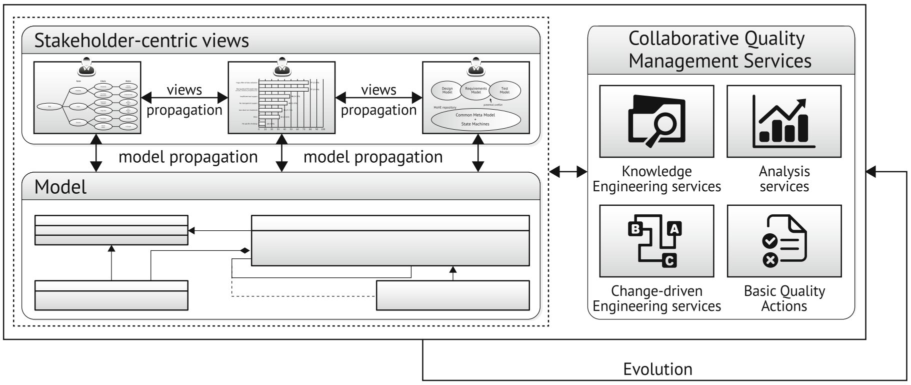
Stakeholder-centric views
views propagation
views propagation
model propagation
model propagation
Model
Collaborative Quality Management Services
Knowledge Engineering services
Analysis services
Change-driven Engineering services
Basic Quality Actions
Evolution

Fig. 2. Core concepts of living models

manual and automated actions. As a reference model we employ state machines attached with model elements to control the change-driven workflows (Fig. 2).

3.2 Collaborative Quality Management Services

The Living Models paradigm incorporates the following categories of executable software quality management services:

– **Knowledge Engineering Services** provide each stakeholder with information at the appropriate level of abstraction, way of presentation and degree of quality (e.g., preciseness, actuality). The information may potentially originate from all data resources in the software life cycle, comprising, for example, software portfolio management, business process modeling, requirements engineering, test reports, code, bug tracking, and many more. Examples of Knowledge Engineering services are services for information aggregation, visualization and maintenance, for example sophisticated views for Enterprise Architecture Models or code bases [25] and semi-automated maintenance services for Enterprise Architecture Models [11].
– **Analysis Services** apply assistive analysis techniques at model instance and meta model level and on runtime data to enhance model quality and to control processes. Examples for analysis services comprise analysis of code level information (service code, tests, bugs) [1], model consistency checks [9,27] or risk models [15,35,46] controlling test processes.
– **Change-Driven Engineering Services** coordinate quality relevant actions of stakeholders in dynamic contexts. These services track and propagate changes and coordinate change handling. Services of this kind can be found in many tools, e.g., in form of tickets in requirements and defect management tools.
– **Basic Quality Actions** refer to any quality related action performed on a homogeneous data base or in a given tool, either in a (semi-)automated or

manual way. This, for example, comprises all testing services, code generation out of models, reviews or static code analysis. Since in this paper we are focusing on collaborative aspects, basic quality actions are not considered further.

The above categorization of collaborative quality management services may not be sharp in all cases. For instance, data visualization may require sophisticated data analysis, and thus may adhere both to Knowledge Management and Analysis services.

The concept of Living Models has been developed in several third party funded projects. It has been materialized in research prototypes and tool environments of industrial partners. Case studies have been conducted in manifold contexts, including security requirements engineering [3], enterprise architecture management [11] and system testing [14].

4 Traceability Categories

Traceability between artifacts is the core concept to provide the services listed before in the context of the model-based collaborative software quality management framework of Living Models. Having also in mind the aforementioned benefits of traceability classifications, and the services of Living Models, we identified the following traceability categories.

Design Time Traceability. Design time traceability denotes traceability at the level of design time and management artifacts. This includes both the well-known notions of horizontal and vertical traceability in literature, as defined, for example, in [22,49]. Horizontal traceability concerns the interconnection of data at the same level of abstraction (e.g., recording dependencies between requirements), whereas vertical traceability concerns the interconnection of data across levels of abstraction (e.g., linking requirements with software architecture artifacts or linking application level model elements with infrastructure model elements in an Enterprise Architecture Model).

Deployment Traceability. Deployment traceability is the traceability between design time artifacts and information collected at runtime (e.g., deployed system components, runtime events, key performance indicators like service duration). An example for deployment traceability is the attachment of workflow models with the maximum number of running instances.

Evolution Traceability. Evolution traceability is the traceability of information across versions, like versions of component implementations or models. Evolution traceability may vary in its granularity and kind of representation. For instance, model history may be documented on model element level (documenting the history of each model element) or on model type level (documenting the history of the whole model), and may be stored as change operation or as sequence of model versions [2].

Stakeholder Action Traceability. Stakeholder action traceability means the association of events or actions with the stakeholders involved. Stakeholder action traceability has many facets since the tracking may vary in the kind of information that is tracked and in the way how stakeholders are represented (e.g., as institutions, roles or persons). Accordingly, stakeholder traceability might be unwanted or even legally prohibited due to privacy reasons.

5 Traceability Supporting Collaborative Software Quality Management Services

In this section we interrelate the categorized kinds of software quality management services with the categorized kinds of traceability (Fig. 3). We do this by referring to *approaches from literature* which consider applications of traceability within the domain of software quality management services. The following categorization may not be sharp in all cases, due to the previously mentioned non-sharp categorization of Collaborative Software Quality Management Services. Some of the examples may fall into several service and/or traceability types but are representative for the category where they are listed.

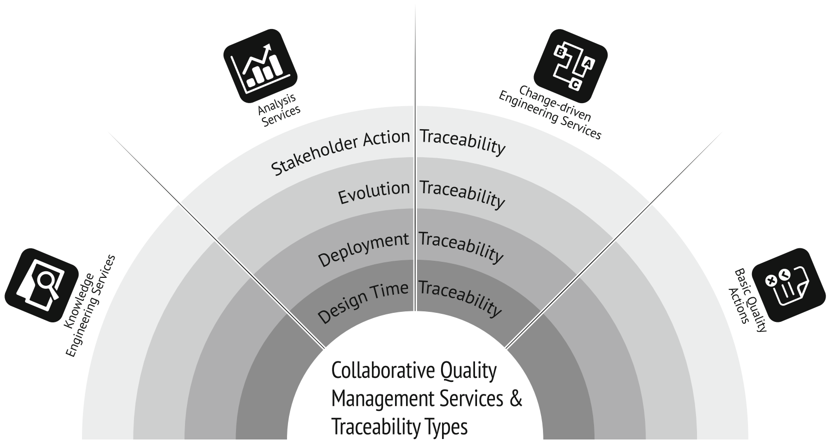
Analysis Services
Change-driven Engineering Services
Knowledge Engineering Services
Basic Quality Actions
Stakeholder Action Traceability
Evolution Traceability
Deployment Traceability
Design Time Traceability
Collaborative Quality Management Services & Traceability Types

Fig. 3. Collaborative quality management services and traceability types

5.1 Traceability Support for Knowledge Engineering Services

Design Time Traceability. Mohan and Ramesh [34] discuss the key role played by a traceability-based knowledge management system in documenting design decisions associated with various configurations of basic building blocks on which service family architectures are based on and in tracing variability. A framework for managing traceability knowledge for the design and development of e-service families, based on the REMAP (Representation and Maintenance of Process Knowledge) environment [41], is presented. Using the case study, they illustrate the importance of using such a knowledge management system for the design and development of service families. Furthermore, in [38] traceability across design artifacts in software project, i.e., from product specification to interface requirements specification, software requirement specification and system components, is addressed.

Deployment Traceability. Conklin and Begeman [7] describe an application-specific hypertext system designed to facilitate capturing of early design deliberations. As consequence, traceability relations may lead to the reuse of system components when these components are related to requirements of existing systems that are similar to requirements of new systems.

Evolution Traceability. Goknil et al. [20] present a tool built to support reasoning about requirements relations, as well as consistency checking of relations and for inferring new relations. This tool supports better understanding of dependencies between requirements, tracking model history and changes (either at model element or type level).

Stakeholder Action Traceability. Li and Maalej [26] present a comparative study of common traceability visualization techniques, including an experiment and interviews with 24 participants. This study supports the finding that traceability, in general, helps users in describing and tracking the relationships between software artifacts, while different visualization techniques (matrices, graphs, hyperlinks, lists) visualize these relationships and help users to access and understand them.

5.2 Traceability Support for Analysis Services

Design Time Traceability. Felderer et al. [15] and Stallbaum et al. [46] apply risk models based on traceable design artifacts for testing purposes.

Deployment Traceability. Gander et al. [19] present a pluggable framework for multi-level security monitoring of workflows, which links event information and modeling specification in order to perform compliance detection and anomaly detection. In [51] authors propose a traceability based knowledge management approach to support adaptation of workflows capability in Workflow Management Systems. They present a framework for representing traceability knowledge to capture the context in which workflows are specified and evolved. They also discuss the functionalities of a knowledge management system that can support dynamic reconfiguration of workflows in the context of changing business processes as well as for maintaining their integrity.

Evolution Traceability. Hata et al. [23] present a fine-grained version control system for Java called Historage. This system targets Java software and conducts fine-grained prediction of bugs with well-known historical metrics (fine-grained module histories). Maletic et al. [28] present an XML based approach to support the evolution of model-to-model traceability links is presented. This approach allows for versioning and differencing of specific elements of the models versus just lines or whole files.

Stakeholder Action Traceability. Zimmermann et al. [52] apply data mining of version histories in order to guide programmers along related changes.

5.3 Traceability Support for Change-Driven Engineering Services

Design Time Traceability. Felderer et al. [14] present a model-driven system testing methodology for service-centric systems called Telling TestStories, its tool implementation and the underlying model validation mechanism. This methodology is based on tightly integrated but separated platform-independent requirements, system and test models. Telling TestStories is capable of test-driven development on the model level and provides full traceability between all system and testing artifacts. Change propagation between the system, test and requirements artifacts is used for regression test derivation. In [48] authors discuss facilitating change management in geographically distributed software engineering by effective discovery and establishment of dependency links using domain models which provide a common reference point. The proposed method advocates the use of domain models throughout the whole development life-cycle and is apt to facilitate multi-site software engineering.

Deployment Traceability. The authors of [4,12,24] deal with synchronization of Enterprise Architecture Models with the runtime environment. The first paper investigates a specific Enterprise Service Bus (ESB) considered as the nervous system of an enterprise interconnecting business applications and processes as an information source. A vendor-specific ESB data model is reverse-engineered and transformation rules for three representative EA information models are derived. These transformation rules are further employed to perform automated model transformations making the first step towards an automated EA documentation. In the second paper the authors propose network scanning for automatic data collection and uses an existing software tool for generating EA models based on the IT infrastructure of enterprises. The third paper presents (semi-)automated processes for maintaining enterprise architecture models by gathering information from both human input and technical interfaces and discusses implementation issues for realizing the processes in practice. This work aims toward the direction of minimizing manual work for EAM by automation and increasing EA data quality attributes such as consistency and actuality.

Table 1. Examples of traceability supporting Collaborative Software Quality Management Services

	Knowledge management	Data analysis	Change-driven engineering
Design time traceability	Framework for managing traceability knowledge for the design and development of e-service families [34] based on the REMAP (Representation and Maintenance of Process Knowledge) environment [41] Traceability across design artifacts in software project, i.e., from product specification to interface requirements specification, software requirement specification and system components [38]	Risk models based on design artifacts are used for testing purposes [15,46] Identify components and objects which satisfies a requirement [10] Test procedures, if traceable to requirements or designs, can be modified when errors are discovered [40] Rigorous system testing by supporting vertical traceability; Rigorous vertical software system testing In IDE [42] System testing by relating requirements with test models and indicating routes for demonstrating product compliance [45]	Change propagation between the system, test and requirements artifacts is used for regression test derivation [14] Propagation of changes during redesign [8] Facilitating change management in geographically distributed software engineering by effective discovery and establishment of dependency links using domain models [48]
Deployment traceability	Traceability may lead to the reuse of system components when these components are related to requirements of existing systems [45]	Multi-level security monitoring of workflows [19] Dynamic reconfiguration of workflows in the context of changing business processes as well as for maintaining their integrity [51]	Synchronisation of Enterprise Architecture Models with the runtime environment [4,12,24]
Evolution traceability	Tracking model history and changes (either at model element or type level) [20]	Prediction of bugs based on fine-grained module histories [23] An XML based approach to support the evolution of model-to-model traceability links [28]	Event-Based Traceability for Managing Evolutionary Change [6]
Stakeholder action traceability	Visualization of traceability [26]	Data mining of version histories in order to guide programmers along related changes [52]	Change-driven collaborative security requirements management [43] - maintenance and evaluation of security requirements in multi-user environments through state-based workflows

Evolution Traceability. Cleland-Huang et al. [6] propose a new method of event-based traceability for managing evolutionary change, which is applicable even in a heterogeneous and globally distributed development environment. Traceable artifacts are no longer tightly coupled but are linked through an event service, which creates an environment in which change is handled more efficiently, and artifacts and their related links are maintained in a restorable state. The method also supports enhanced project management for the process of updating and maintaining the system artifacts.

Stakeholder Action Traceability. Sillaber and Breu [43] elaborate on change-driven collaborative security requirements management, maintenance and evaluation of security requirements in multi-user environments through state-based workflows.

Summarizing, we can state that there are many existing approaches in research that focus on *design time* traceability, as visible in Table 1. For the use of *deployment*, *evolution* and *stakeholder action* traceability much less research has been conducted. However, in each category recent research could be identified [12,23,26].

6 Conclusion

In this paper we first presented novel categories of software quality management services, i.e., knowledge engineering, analysis, and change-driven engineering, and types of traceability, i.e., design-time, deployment, evolution, and stakeholder action traceability, derived from the conceptual framework of Living Models for collaborative software quality management. These trace types empower the software quality management services of Living Models. Through interrelating the traceability and quality management services we demonstrated not only the bandwidth of software quality management services exploiting traceability but also the need for further research in this area.

In our future work we will further develop our classification of trace types, for example by identifying trace types for supporting IT management, systems operation and software engineering. In addition, we will invest efforts in developing and integrating powerful interactive visual solutions, based on premises from this paper, to support the exploration of trace links belonging to the newly identified trace categories.

Acknowledgements. This work was partially funded by the research project QE LaB - Living Models for Open Systems (www.qe-lab.at).

References

1. Binkley, D.: Source code analysis: a road map. In: Future of Software Engineering, 2007. FOSE 2007, pp. 104–119. IEEE (2007)
2. Breu, R., Agreiter, B., Farwick, M., Felderer, M., Hafner, M., Innerhofer-Oberperfler, F.: Living models - ten principles for change-driven software engineering. Int. J. Softw. Informatics **5**(1–2), 267–290 (2011)
3. Breu, R., Hafner, M., Innerhofer-Oberperfler, F., Wozak, F.: Model-driven security engineering of service oriented systems. In: Kaschek, R., Kop, C., Steinberger, C., Fliedl, G. (eds.) UNISCON 2008. LNBIP, vol. 5, pp. 59–71. Springer, Heidelberg (2008). doi:10.1007/978-3-540-78942-0_8
4. Buschle, M., Grunow, S., Matthes, F., Ekstedt, M., Hauder, M., Roth, S.: Automating enterprise architecture documentation using an enterprise service bus. In: 18th Americas Conference on Information Systems, AMCIS 2012 (2012)
5. Capgemini: World quality report 2011/12 (2011)
6. Cleland-Huang, J., Chang, C.K., Christensen, M.: Event-based traceability for managing evolutionary change. IEEE Trans. Softw. Eng. **29**(9), 796–810 (2003)
7. Conklin, J., Begeman, M.L.: gIBIS: a hypertext tool for exploratory policy discussion. ACM Trans. Inf. Syst. (TOIS) **6**(4), 303–331 (1988)

8. Dömges, R., Pohl, K.: Adapting traceability environments to project-specific needs. Commun. ACM **41**(12), 54–62 (1998)
9. Egyed, A.: Instant consistency checking for the UML. In: Proceedings of the 28th International Conference on Software Engineering, pp. 381–390. ACM (2006)
10. Egyed, A., Grünbacher, P.: Supporting software understanding with automated requirements traceability. Int. J. Softw. Eng. Knowl. Eng. **15**(05), 783–810 (2005)
11. Farwick, M., Schweda, C., Breu, R., Voges, K., Hanschke, I.: On enterprise architecture change events. In: Aier, S., Ekstedt, M., Matthes, F., Proper, E., Sanz, J.L. (eds.) TEAR and PRET 2012. LNBIP, vol. 131, pp. 129–145. Springer, Heidelberg (2012). doi:10.1007/978-3-642-34163-2_8
12. Farwick, M., Agreiter, B., Breu, R., Ryll, S., Voges, K., Hanschke, I.: Automation processes for enterprise architecture management. In: 2011 15th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW), pp. 340–349. IEEE (2011)
13. Farwick, M., Pasquazzo, W., Breu, R., Schweda, C.M., Voges, K., Hanschke, I.: A meta-model for automated enterprise architecture model maintenance. In: EDOC, pp. 1–10 (2012)
14. Felderer, M., Chimiak-Opoka, J., Zech, P., Haisjackl, C., Fiedler, F., Breu, R.: Model validation in a tool-based methodology for system testing of service-oriented systems. Int. J. Adv. Softw. **4**(1 and 2), 129–143 (2011)
15. Felderer, M., Haisjackl, C., Breu, R., Motz, J.: Integrating manual and automatic risk assessment for risk-based testing. In: Biffl, S., Winkler, D., Bergsmann, J. (eds.) SWQD 2012. LNBIP, vol. 94, pp. 159–180. Springer, Heidelberg (2012). doi:10.1007/978-3-642-27213-4_11
16. Felderer, M., Agreiter, B., Breu, R.: Evolution of security requirements tests for service–centric systems. In: Erlingsson, Ú., Wieringa, R., Zannone, N. (eds.) ESSoS 2011. LNCS, vol. 6542, pp. 181–194. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19125-1_14
17. Felderer, M., Zech, P., Fiedler, F., Breu, R.: A tool-based methodology for system testing of service-oriented systems. In: 2010 Second International Conference on Advances in System Testing and Validation Lifecycle (VALID), pp. 108–113. IEEE (2010)
18. Galvao, I., Goknil, A.: Survey of traceability approaches in model-driven engineering. In: 11th IEEE International Enterprise Distributed Object Computing Conference, 2007. EDOC 2007, p. 313. IEEE (2007)
19. Gander, M., Katt, B., Felderer, M., Breu, R.: Towards a model- and learning-based framework for security anomaly detection. In: Beckert, B., Damiani, F., Boer, F.S., Bonsangue, M.M. (eds.) FMCO 2011. LNCS, vol. 7542, pp. 150–168. Springer, Heidelberg (2013). doi:10.1007/978-3-642-35887-6_8
20. Goknil, A., Kurtev, I., van den Berg, K., Veldhuis, J.W.: Semantics of trace relations in requirements models for consistency checking and inferencing. Softw. Syst. Model. **10**(1), 31–54 (2011)
21. Gotel, O., Finkelstein, C.: An analysis of the requirements traceability problem. In: Proceedings of the First International Conference on Requirements Engineering, 1994, pp. 94–101. IEEE (1994)
22. Gotel, O., Finkelstein, A.: Contribution structures [requirements artifacts]. In: Proceedings of the Second IEEE International Symposium on Requirements Engineering, 1995, pp. 100–107. IEEE (1995)
23. Hata, H., Mizuno, O., Kikuno, T.: Bug prediction based on fine-grained module histories. In: 2012 34th International Conference on Software Engineering (ICSE), pp. 200–210. IEEE (2012)

24. Holm, H., Buschle, M., Lagerström, R., Ekstedt, M.: Automatic data collection for enterprise architecture models. Softw. Syst. Model. **13**, 825–841 (2012)
25. Lanza, M., Marinescu, R.: Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-oriented Systems. Springer, Heidelberg (2006). doi:10.1007/3-540-39538-5
26. Li, Y., Maalej, W.: Which traceability visualization is suitable in this context? A comparative study. In: Regnell, B., Damian, D. (eds.) REFSQ 2012. LNCS, vol. 7195, pp. 194–210. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28714-5_17
27. Lucas, F., Molina, F., Toval, A.: A systematic review of UML model consistency management. Inf. Softw. Technol. **51**(12), 1631–1645 (2009)
28. Maletic, J.I., Collard, M.L., Simoes, B.: An XML based approach to support the evolution of model-to-model traceability links. In: Automated Software Engineering: Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering, vol. 8, pp. 67–72 (2005)
29. Marcus, A., Xie, X., Poshyvanyk, D.: When and how to visualize traceability links? In: Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering, pp. 56–61. ACM (2005)
30. Margaria, T., Steffen, B.: Service engineering: linking business and it. Computer **39**(10), 45–55 (2006)
31. Margaria, T., Steffen, B.: Business process modelling in the jABC: the one-thing-approach. In: Handbook of Research on Business Process Modeling, pp. 1–26 (2009)
32. Margaria, T., Steffen, B.: Continuous model-driven engineering. Computer **42**(10), 106–109 (2009)
33. Maté, A., Trujillo, J.: A trace metamodel proposal based on the model driven architecture framework for the traceability of user requirements in data warehouses. In: Mouratidis, H., Rolland, C. (eds.) CAiSE 2011. LNCS, vol. 6741, pp. 123–137. Springer, Heidelberg (2011). doi:10.1007/978-3-642-21640-4_11
34. Mohan, K., Ramesh, B.: Managing variability with traceability in product and service families. In: Proceedings of the 35th Annual Hawaii International Conference on System Sciences, 2002. HICSS, pp. 1309–1317. IEEE (2002)
35. Neubauer, J., Windmüller, S., Steffen, B.: Risk-based testing via active continuous quality control. Int. J. Softw. Tools Technol. Transf. **16**(5), 569–591 (2014)
36. Olsen, G.K., Oldevik, J.: Scenarios of traceability in model to text transformations. In: Akehurst, D.H., Vogel, R., Paige, R.F. (eds.) ECMDA-FA 2007. LNCS, vol. 4530, pp. 144–156. Springer, Heidelberg (2007). doi:10.1007/978-3-540-72901-3_11
37. Paige, R.F., Olsen, G.K., Kolovos, D.S., Zschaler, S., Power, C.: Building model-driven engineering traceability classifications (2008)
38. Ramesh, B., Powers, T., Stubbs, C., Edwards, M.: Implementing requirements traceability: a case study. In: Proceedings of the Second IEEE International Symposium on Requirements Engineering, pp. 89–95. IEEE (1995)
39. Ramesh, B.: Representing and reasoning with traceability in model life cycle management. Ann. Oper. Res. **75**, 123–145 (1997)
40. Ramesh, B., Jarke, M.: Toward reference models for requirements traceability. IEEE Trans. Softw. Eng. **27**(1), 58–93 (2001)
41. Ramesh, B., Tiwana, A.: Supporting collaborative process knowledge management in new product development teams. Dec. Support Syst. **27**(1), 213–235 (1999)
42. Seo, K.I., Choi, E.M.: Rigorous vertical software system testing in ide. In: 5th ACIS International Conference on Software Engineering Research, Management & Applications, 2007. SERA 2007, pp. 847–854. IEEE (2007)

43. Sillaber, C., Breu, R.: Managing legal compliance through security requirements across service provider chains: a case study on the german federal data protection act. In: GI-Jahrestagung, pp. 1306–1317 (2012)
44. Sneed, H.M.: Testing web services in the cloud. In: Winkler, D., Biffl, S., Bergsmann, J. (eds.) SWQD 2013. LNBIP, vol. 133, pp. 70–88. Springer, Heidelberg (2013). doi:10.1007/978-3-642-35702-2_6
45. Spanoudakis, G., Zisman, A.: Software traceability a roadmap. Handb. Softw. Eng. Knowl. Eng. **3**, 395–428 (2005)
46. Stallbaum, H., Metzger, A., Pohl, K.: An automated technique for risk-based test case generation and prioritization. In: Proceedings of the 3rd International Workshop on Automation of Software Test, pp. 67–70. ACM (2008)
47. Steffen, B., Margaria, T., Nagel, R., Jörges, S., Kubczak, C.: Model-driven development with the jABC. In: Bin, E., Ziv, A., Ur, S. (eds.) HVC 2006. LNCS, vol. 4383, pp. 92–108. Springer, Heidelberg (2007). doi:10.1007/978-3-540-70889-6_7
48. Strasunskas, D., Hakkarainen, S.E.: Domain model-driven software engineering: a method for discovery of dependency links. Inf. Softw. Technol. **54**, 1239–1249 (2012)
49. Von Knethen, A., Paech, B.: A survey on tracing approaches in practice and research. Frauenhofer Institut Experimentelles Software Engineering, IESE-Report No 95 (2002)
50. Winkler, S., Pilgrim, J.: A survey of traceability in requirements engineering and model-driven development. Softw. Syst. Model. (SoSyM) **9**(4), 529–565 (2010)
51. Xu, P., Ramesh, B.: Supporting workflow management systems with traceability. In: Proceedings of the 35th Annual Hawaii International Conference on System Sciences, pp. 1519–1528. IEEE (2002)
52. Zimmermann, T., Weibgerber, P., Diehl, S., Zeller, A.: Mining version histories to guide software changes. In: Proceedings of 26th International Conference on Software Engineering, 2004. ICSE 2004, pp. 563–572. IEEE (2004)