Modeling of Requirements Tracing

Matthias Heindl[1], Stefan Biffl[2]

[1] Support Center Configuration Management, Siemens Program and Systems Engineering, Siemens AG Austria, Gudrunstrasse 11, A-1100 Vienna, Austria
Matthias.a.Heindl@siemens.com
[2] Institute of Software Technology and Interactive Systems, Vienna University of Technology, Favoritenstrasse 9/188, A-1040 Vienna, Austria
Stefan.Biffl@tuwien.ac.at

Abstract. Software customers want both sufficient product quality and agile response to requirements changes. Formal software requirements tracing helps to systematically determine the impact of changes and to keep track of development artifacts that need to be re-tested when requirements change. However, full tracing of all requirements on the most detailed level can be very expensive and time consuming. In the paper an initial "tracing activity model" is introduced along with a framework that allows measuring the expected cost and benefit of tracing approaches. In a feasibility study a subset of the activities belonging to the model has been applied to compare three tracing strategies: agile, "just in time" tracing, and fully formal tracing. The study focused on re-testing and it has been performed in the context of an industry project where the customer was a large financial service provider.In the study a) the model was found useful to capture costs and benefits of the tracing activities and to compare different strategies; b) a combination of tracing approaches proved helpful in balancing agility and formalism.

Keywords: Software Requirements Tracing, Re-Test, Tracing Activity Model, Feasibility study.

1 Introduction

The main goal of software development projects is to develop software that fulfills the requirements of most important stakeholders, i.e., customers and users. However, in typical projects requirements tend to change throughout the project, e.g. due to revised customer needs or modifications in the target environments. These changes of requirements may introduce significant extra effort and risk, which need to be assessed realistically when change requests come up, e.g. test cases have to be adapted in order to test the implementation against the revised requirements. Thus, software test managers need to understand the likely impact of requirement changes on product quality and needs for re-testing (regression testing) to continuously balance agile reaction to requirements changes with systematic quality assurance activities.

An approach to support the assessment of the impact of requirements changes is formal requirements tracing, which helps to determine necessary changes in the design and implementation as well as needs for re-testing existing code more quickly and accurately. Requirements tracing is formally defined as the ability to follow the life of a requirement in a forward and backward direction [11], e.g. by explicitly capturing relationships between requirements and related artifacts. For example, a trace between a requirement and a test case indicates that the test case checks code against the requirement.

Such traces can be used for change impact analysis: if a requirement changes, a test engineer can efficiently follow the traces from the requirement to the related test cases and identify the correct test cases that have to be checked, adapted, and re-run to systematically re-test the software product.

However, in a real-world project full tracing of all requirements on the most detailed level can be very expensive and time consuming. Thus, the costs and benefits to support the desired fast and complete change impact analysis need to be investigated with empirical data. While there are many methods and techniques on how to technically store requirements traces, there is very few systematic discussion on how to measure and compare the tracing effort and effectiveness of tracing strategies in an application scenario such as re-testing.

This paper proposes an initial *tracing activity model* (TAM), a framework to systematically describe and help determine the likely efforts and benefits, like reduced expected delay and risk, of the tracing process in the context of a usage scenario such as re-testing of software. The TAM defines common elements of various requirements tracing approaches: trace specification, generation, deterioration, validation, rework, and application; and parameters influencing each activity like number of units to trace, average effort per unit to trace, and requirements volatility.

The model can support requirements and test managers in comparing requirements tracing strategies, e.g. for tailoring the expected re-test effort and risk based on selected parameters: process alternatives, expected test-case creation effort, and expected change-request severity. We apply the TAM in a feasibility study that compares effort, risk, and delay of three tracing strategies: no tracing at all (no-T), full formal tracing (full-T) for re-testing, and value-based tracing (value-T).

The remainder of the paper is organized as follows: Section 2 summarizes related work on requirements tracing and requirements-based testing; Section 3 introduces the tracing activity model and research objectives. Section 4 outlines the feasibility study and summarizes the results. Section 5 discusses the results and limitations of the study and lessons learned; finally Section 6 concludes and suggests further work.

2 Related Work on Requirements Tracing and Re-Testing

Several approaches have been proposed to effectively and efficiently capture traces for certain trace applications like change impact analysis and testing [1][4][7].

Many standards for systems development such as the US Department of Defense (DoD) standard 2167A mandate requirements traceability practice [23]. Gotel and

Finkelstein [11] define requirements tracing as the ability to follow the life of a requirement in both a backward and forward direction. Requirements traceability is an issue for an organization to reach CMMI level 3 making tracing an issue that many maturing software development organizations have to consider: the assessment for maturity level 3 there contains questions concerning requirements tracing: whether requirements traces are applied to design and code and whether requirements traces are used in the test phases.

The tracing community, e.g., at the Automated software engineering (ASE) tracing workshop TEFSE [7][8], traditionally puts somewhat more weight on technology than on process improvement. Basic techniques for requirements tracing are cross referencing schemes [9], key phrase dependencies [18], templates, RT matrices, hypertext [20], and integration documents [21]. These techniques differ in the quantity and diversity of information they can trace between, in the number of interconnections between information they can control, and in the extent to which they can maintain requirements traces when faced with ongoing changes to requirements.

Commercial requirements management tools like Doors, Requisite Pro, or Serena RM provide the facility to relate (i.e. create traces between) items stored in a database. These tools also automatically indicate which artifacts are effected if a single requirement changes (suspect traces). However, the tools do not automate the generation of trace links (capturing a dependency between two artifacts as a trace), which remains a manual, expensive, and error-prone activity.

Watkins and Neal [24] report how requirements traceability aids project managers in: accountability, verification (testing), consistency checking of models, identification of conflicting requirements, change management and maintenance, and cost reduction.

Gotel and Finkelstein [11] also state the requirements traceability problem, caused by the efforts necessary to capture and maintain traces. Thus, to optimize the cost-benefit of requirements tracing, a range of approaches focused on effort reduction for requirements tracing. In general, there are two effort reduction strategies: (1) automation, and (2) value-based software engineering.

1. Automation. Multiple approaches have been developed to automate trace generation: Egyed has developed the Trace/Analyzer technique that automatically acquires trace links based on analyzing the trace log of executing key system scenarios [5][6]. He further defines the tracing parameters: precision (e.g., traces into source code at method, class, or package level), correctness (wrong vs. missing traces), and completeness. Other researchers have exploited information retrieval techniques to automatically derive similarities between source code and documentation [1], or between different high-level and low-level requirements [17]. Rule-based approaches have been developed that make use of matching patterns to identify trace links between requirements and UML object models represented in XML [25]. Neumüller and Grünbacher developed APIS [22], a data warehouse strategy for requirements tracing. Cleland-Huang *et al.* adopt an event-based architecture that links requirements and derived artifacts using the publish-subscribe relationship [3].

2. Value-based software engineering. The purpose of a value-based requirements tracing approach is not to reduce effort of each unit to trace (like automation) but to

trace all requirements with varying levels of precision, and thereby reduce the overall effort for requirements tracing [13], e.g., high-priority requirements are traced with a higher level of precision (e.g., at source code method level), while low-priority requirements are traced with lower precision (e.g., at source code package level).

The effort used to capture traces should be justifiable with the effort that could be saved by using these traces in software engineering activities like change impact analysis, testing [4][16][19], or consistency checking. It is a matter of balancing agility and formalism to come close to an optimal level of cost-benefit [2]. The approaches described above serve the purpose of reducing the effort to capture traces.

Effort for capturing tracing + effort for trace application by using traces < effort for trace application without using traces Eqn (**1**)

Equation 1 captures this idea from a value-based perspective: to achieve a positive return on investment of requirements tracing the effort of generating and using traces should be lower than the effort for a trace application without traces. Such trace applications are (amongst others) change impact analysis and re-testing [10]. Besides effort of capturing traces, reduction of risk due to missed traces and delay due to the need to update traces are criteria that determine the usefulness of tracing approaches for software engineering activities.

Changes of requirements affect test cases and other artifacts [12]. Change impact analysis is the activity where the impacts of a requirement's change on other artifacts are identified [18]. Usually, all artifacts have to be scanned for needed adoptions when a change request for a requirement occurs. A trace-based approach relates requirements with other artifacts to indicate interdependencies. These relations (traces) can be used during change impact analysis for more efficient and more correct identification of potential change locations.

In [13] we proposed an initial cost-benefit model, where the following parameters that influence the cost-benefit of RT are identified: number of requirements and artifacts to be traced, volatility of requirements, and effort for tracing. In [14] we further discussed the effects of trace correctness as parameter influencing the cost-benefit of RT. However, tracing activities were not modeled explicitly, which would facilitate a more systematic discussion of the merits of different tracing approaches.

3 An Initial Tracing Activity Model

Most work in requirements tracing research has focused more on technology than on processes supported by this technology to generate and use traces. For the systematic comparison of tracing alternatives we propose in this section a process model, the tracing activity model (TAM), which contains the activities and parameters found in research tracing approaches; we label the framework as initial, although it is based on a systematic review literature and tracing activities in practice, as the external validation process has not yet concluded. The model can be used as basis to formally evaluate and compare tracing approaches as well as their costs and benefits.

3.1 Tracing Process Variants, Activities, and Parameters

The tracing activity model (TAM) in Figure 1 depicts a set of activities to provide and maintain the benefits of traces over time. The model is a framework to measure the cost and benefit of requirements tracing in order to compare several tracing strategies for a development project. The framework is based on previous work that identified tracing parameters, e.g., [3][7][8][13][14][17].

The activities in the model are building blocks identified from practice and literature and follow the life cycle of a set of traces.

Trace Specification is the activity where the project manager defines the types of traces that the project team should capture and maintain. For example, the project manager can decide to capture traces between requirements, source code elements, and test cases. This activity influences tracing effort based on the following parameters: Number of artifacts to be traced, number of traces, and artifacts to be traced. Other relevant parameters are tracing scope, precision of traces [7][8][13],

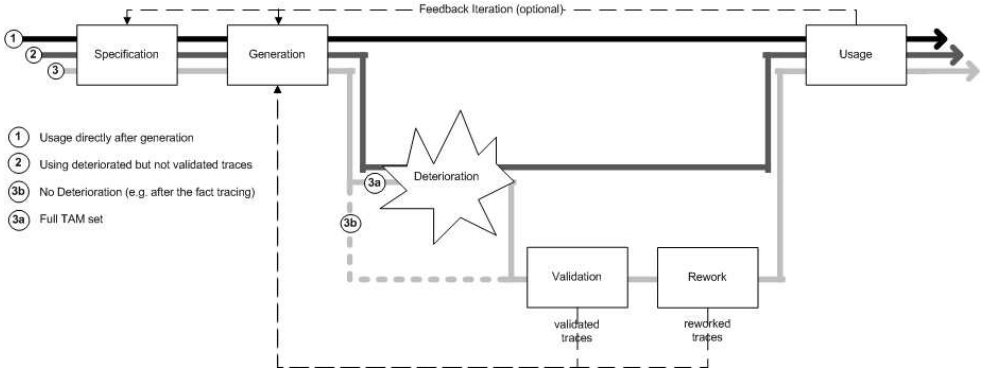

Fig. 1. Tracing activity model: Activities and process variants.

Trace Generation. Trace generation is the activity of identifying and explicitly capturing traces between artifacts. Methods for trace generation range from manually capturing traces in matrices or requirements management tools that automatically create traces between artifacts based. The effort to generate traces in a project depends on the following parameters [13]:

- Number of requirements: in a software development project; the effort for tracing increases with increasing number of requirements.
- Number of artifacts to be traced to the higher the number of artifacts, the higher is the effort to create traces between them.
- Average trace effort per unit to trace, which depends on the used tools and the point in time of tracing.

Other relevant parameters are: number of traces, tool support, point in time of trace generation in the software development process, complexity/size of tracing objects, value of traces [13], correctness and completeness of traces.

Trace Deterioration. Trace deterioration is more the impact of external events than an activity. Traces can degrade over time as related artifacts change. If only the artifacts are updated, e.g., due to change requests, and the traceability information is not updated, the set of existing traces is likely to get less valid over time. Deterioration of traces affects the value of traces, because it reduces the correctness and completeness of traces.

Trace Validation and Rework. Trace validation is the activity that checks if the existing traceability information is valid or needs to be updated, e.g., identify missing trace links. In the example above, when artifact *A* changes fundamentally so that there is no longer a relationship to artifact *B*, trace validation would check the trace between *A* and *B* and flag it as obsolete. Trace validation is necessary to keep the trace set (traceability information) correct and up to date, so that the traces are still useful when used, e.g., for change impact analyses. We call the updating of traces "trace rework". Trace validation and trace rework are often performed together as they ensure correct and up-to-date traces and counter trace deterioration effects. The effort for validation and rework depend partly on the volatility of requirements.

The tracing activities are not necessarily performed in sequence. Furthermore, some activities are mandatory, like trace generation, whereas other activities are optional, as indicated by the arrows in Figure 1:

- *Trace Usage directly after generation (process variant 1 in figure 1):* Trace deterioration depends on the changes made to certain artifacts. If traces stay valid over time and do not deteriorate, validation and rework are not necessary so that the existing traces can be used, e.g., for change impact analyses.
- *Using deteriorated traces (process variant 2 in figure 1)* without validating and reworking them before is possible, but reduces the traces' benefits, because wrong or missing traces may hinder the supported activity more than they help
- *No Deterioration (process variant 3b in figure 1):* Traces can be validated after generation whenever the project manager wants, even when they did not deteriorate.

Trace Usage. Finally, traceability information is used as input to tracing applications like change impact analysis, testing, or consistency checking [24]. The overall effort of such a tracing application is expected be lowered by using traces. The benefits of tracing during trace usage depend on parameters explained in [15].

The cost-benefit of requirements traceability can be determined as the balance of efforts necessary to generate, validate, and rework traces (cost); and saved efforts during trace usage, reduced risk and delay of tracing (benefits during change impact analysis). To maximize the net gain of requirements tracing the effort of generating, validating and reworking traces can be minimized, or the saved effort of trace usage can be maximized.

3.2 Research Objectives

The value of tracing comes from using the trace information in an activity such as re-testing that is likely to be considerably harder, more expensive, or to take longer without appropriate traces. If a usage scenario of tracing is well defined, trace generation can be tailored to provide appropriate traces more effectively and efficiently. Keeping traceability in the face of artifact changes takes further maintenance efforts.

The tracing activity model allows to formally define tracing strategies for a usage scenario by selecting the activities to be performed and by setting or varying the activity parameters.

We address the following research question:

- *RQ1: How useful is the TAM to model requirements tracing strategies and to determine and compare their efforts?*
- *RQ: To what extent can we balance the agility of a re-testing approach without using traces and the formalism of a systematic tracing approach for re-testing with a value-based approach?*

In order to evaluate the usefulness of the tracing activity model we conducted a small feasibility study in the finance domain, where we applied the TAM to 3 tracing strategies for the trace application re-testing. We discussed the usefulness of the re-testing strategies and the tracing model with development experts. If useful, the lessons learned from our evaluation could be a basis for extrapolation of tracing strategies and cost-benefit parameters to larger projects.

Re-testing is a software engineering activity that can be supported well by requirements tracing. The goal of a trace-based testing approach can be to make testing less expensive, less risky, and to reduce the delay. For a positive return on investment of tracing the effort to generate and maintain traces plus the effort of re-testing has to be lower than the effort of testing without tracing support.

4 Application of the TAM in an Industrial Feasibility Study

This section describes a feasibility study to validate the initial TAM framework concept. Together with practitioners from the quality assurance department of a large financial service provider we modeled 3 tracing strategies by using TAM building blocks and parameters and calculated tracing efforts of each strategy, their risks and delay in order to support the practitioners in deciding which tracing strategy provides the best support for re-testing in the practitioners' particular project context. This section describes an overview how we modeled each tracing strategy; detailed information of the study context can be found in the technical report [15].

The main focus of the study was to compare the efforts of each tracing strategy and the expected benefits of trace usage for re-testing. The TAM output variables were (1) the total effort of re-testing, (2) the risk of each strategy, and (3) the delay. Input variables were parameters covered the number of test cases, effort to create a trace, effort to create a test case, change impact analysis effort, etc. (see [15] for a comprehensive list of parameters).

Based on discussions with the experts in the industry environment and suggestions from literature we defined and compared 3 tracing strategies for re-testing: no tracing at all (no-T), full formal tracing (full-T), and value-based tracing (value-T). The data from this study can provide an initial snapshot in a typical scenario to find out whether the framework are useful to provide data and the proposed tracing strategies seem worthwhile for further discussion.

No tracing at all (no-T). As a baseline strategy we used the no-T strategy, which was the standard strategy in the feasibility study context; in this traditional re-testing process there is no trace support. Thus the activities of the tracing activity model are not performed and re-testing has to cope without traces: For each change request, the testers create new test cases instead of re-using and adapting existing ones. Obsolete test cases are replaced by new ones in order to avoid the risk of having redundant or inconsistent test cases, and to make sure everything is tested and test cases are still valuable after the change.

$$E(\text{no-T}) = \#cr * \#tc * tcn + dor. \qquad \text{Eqn (2a)}$$

E(no-T) = 20 change requests * 6 test cases* 1hrs + 6*700*8 min = 120 hrs + 560 hrs = **680 hrs**. Eqn (**2b**)

Equation (2a) calculates the overall re-testing effort following the no-T strategy: for each change request (#cr), new test cases are created with the expected effort (#tc* tcn). Finally, the testers have to check newly created test cases with existing test cases and delete redundant (obsolete) old test cases (dor).

In the particular study the total effort for no-T was as calculated in Eqn 2b (see [15] for detailed explanation.

Full formal tracing (full-T) for re-testing. In the full-T strategy, testers systematically establish traceability by relating requirements and test cases (full tracing) via a tool, the Mercury Test Director. When a change request occurs, they check, and adapt existing test cases whenever possible; else they create new test cases.

$$E(\text{full-T}) = \#tntc * te + cia\_T * \#cr + tcnra * \#tc * \#cr \qquad \text{Eqn (3)}$$

Equation (3) calculates the overall re-testing effort following the full-T strategy: The formula consists of 3 parts: (a) upfront traceability effort (#tntc * te), which establishes traceability for each existing test case, (b) the effort to identify affected test cases for each change request (cia_T * #cr), and (c) the effort needed to either reuse (tcr) or adapt (tca) existing test cases, depending on the severity of the change requests (#cr). If existing test cases can neither be reused nor adapted, new test cases have to be developed (tcn).

The shares of test cases that can be reused, adapted, or need to be created anew typically has an important impact on the overall effort of re-testing.

The effort of full-T for change impact analysis depends on how many traces between requirements and test cases can be reused, have to be adapted, or must be

created. These values depend on the type of change request, as not every change request effects artifacts in the same way, e.g., there are simple low-effort change requests, e.g., affecting locally the user interface, whereas more severe change requests may need more extensive adaptations in several software product parts. Eqn 4a and 4b depict the efforts for full-T.

CIA_T effort overall = 54 + 86 + 33 hrs = 173 hours Eqn (**4a**)

E(full-T) = upfront trace effort + CIA_T = 350 + 173 = **523 hrs** Eqn (**4b**)

Based on effort reports for typical change requests in the case study context we categorized change request into the classes: Mini (small), Midi (medium), and Maxi (severe) (see [15] for details).

Value-based tracing (value-T) is a hybrid between full-T and ad-hoc tracing. Usually the upfront effort for full-T is considerably high, because all existing requirements have to be traced to test cases. value-T tries to reduce this tracing effort by establishing traceability on a coarse level (to test case packages instead of particular test cases) and to refine them ad-hoc when necessary. That means that all requirements are traced to test case packages and when change requests occur for some requirements, the traces from these test cases are refined to particular test cases to improve change impact analysis. Equation (5) calculates the overall re-testing effort following the value-T strategy:

E(value-T) = upfront trace effort (on package level) + change impact analysis (value-T) Eqn (**5a**)

E(value-T) = 70 hrs + 325 hrs = **395 hrs** Eqn (**5b**)

The upfront tracing effort for value-T is lower since traces have to be captured on more coarse level of detail than with full-T (70 hrs in comparison to 350 hrs with full-T). The change impact analysis effort for value-T consists of refining traces from changing requirements to the affected test case packages. The effort for identifying particular test cases by refinement was 325 hrs in the study resulting in a total effort of 395 hrs for the value-based tracing strategy to support re-testing.

5 Discussion

The purpose of the case study was to evaluate the feasibility of the TAM to model tracing strategies, in our case with focus on effort, also considering delay, and risk.

For practical reasons, the case study size and context was chosen to allow evaluating the approaches in a reasonable amount of time. However, the case study project setting seems typical in the company and financial service sector; the project context allows reasonable insight into the feasibility of the trace-based re-testing strategy in this environment.

In the feasibility study project, we deliberately applied a simple process variant from the TAM focusing on the activities trace specification, trace generation and the usage of generated traces for re-testing. Trace deterioration, validation and re-work were not enacted; rather we assumed for trace usage all generated traces to be correct. While this reduction of scope limits the experience this focus was found beneficial to make sure that the proposed process is actually applied in the practical setting.

As with any empirical study the external validity of only one study can not be sufficient for general guidelines, but needs careful examination in a range of representative settings. Furthermore, we analysed only a simple instantiation of the tracing activity model in the case study; consisting of trace generation and trace usage, but without considering trace deterioration, and consequently neither trace validation nor rework. In practice incorrect traces and trace deterioration can considerably lower tracing benefits and need to be investigated.

Modelling the 3 tracing strategies by using TAM activities and parameters provided data points for effort of each strategy. As these are single data points in a specific study setting, we see the results as snap shots, which should motivate further data collection to allow statistical data analysis and sensitivity analysis.

Comparing the 680 person hours effort of the no-T strategy, where new test cases are created for each test case, with the full-T alternative, with 523 person hours, full-T takes around 20% less effort. In this case the upfront investment into traceability pays off. In many cases, full tracing (tracing each requirement to each relevant test case) can cause considerably high effort which may prevent tracing in practice. Here, the study results suggest that the value-based strategy to trace requirements to test cases on a coarse level and refine them later on demand to be a promising approach that can significantly save efforts.

Besides effort, the alternatives also differ in delay when traces can be used for the trace application, in our case re-testing. value-T has a larger delay, because trace refinement has to be done before re-test. Concerning risk, no-T would be more risky if obsolete test cases were not checked. Inconsistent or redundant test case sets could then result in increased hidden testing effort or lower-quality test sets.

Lessons Learned from the Feasibility Study. The tracing activity model was found useful for systematically modeling the tracing alternatives, e.g., no tracing, systematic full tracing, and value-based tracing for the certain tracing application re-testing. The model helps make alternative strategies comparable, as it makes the main tracing activities explicit and allows mapping relevant parameters that influence tracing costs and benefits. Some input parameters (like number of change requests in the project, or effort to create a test case) had to be estimated based on practitioners' experience. Other data elements could be measured in the project context, e.g., number of requirements. The TAM allows choosing from the listed tracing activities and parameters and selecting the relevant ones to model tracing strategies for a particular usage scenario.

According to the expert feedback the calculated efforts provide a good input to reason about which tracing strategy seems most beneficial in a particular project context.

The lessons learned of our study for trace-support change impact analysis are:

- TAM provides useful building blocks for reasoning about relevant parameters (efforts, risks, etc.) of a tracing strategy and estimation of outcomes in advance helps to rationally discuss candidates for the best-fitting strategy.
- The volatility of traces is a major risk for full tracing. In volatile parts of the project, agile (just in time) or value-based approaches are favorable as full tracing has a particularly high risk of loosing upfront investments in tracing in these volatile areas.
- Full tracing provides detailed traces, which are particularly useful for situations when artifacts are not volatile and quick feedback is at a premium, e.g., for comprehensive cross checks at milestone reviews.
- If calculated efforts of tracing strategies do not differ significantly, choose a value-based strategy to provide full (complete) traceability at a coarse level of detail. This coarse-level traceability can than be refined on demand with reasonable total effort for change impact analysis.

6 Conclusion and Further Work

In the paper we proposed an initial tracing activity model (TAM) as a framework for defining and comparing tracing strategies for various contexts. For each tracing activity, relevant parameters were identified from related work and practice and mapped into the model. The model allows to systematically compare tracing strategy activities, their costs and benefits. We performed a small study in the financial service domain, where we evaluated the feasibility of the tracing activity model.

Main results of the study are: a) The model was found useful to capture costs and benefits of the tracing activities and compare different strategies; b) for volatile projects or project parts just-in-time tracing seems favorable; c) for parts that need quick feedback detailed upfront preparation of traces can be warranted; d) a combination of upfront tracing on a coarse level of detail (e.g. package or class level) and just-in-time detailed tracing of really needed traces can help balancing agility (important from the project point of view) and formality (that allows evidence-based software process improvement and is important from the software organization point of view).

Further work will be a) to use TAM as a framework for a systematic literature review concerning requirements tracing and b) to apply TAM for studies on tracing strategies in other contexts.

References

1. G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. IEEE Transactions on Software Engineering, 28(10):970–983, 2002.
2. Boehm, Turner. Balancing Agility and Discipline, Addison Wesley, 2005

3. J. Cleland-Huang, G. Zemont, W. Lukasik, A Heterogeneous Solution for Improving the Return on Investment of Requirements Traceability, RE 2004, 230-239
4. S. Elbaum, D. Gable, G. Rothermel, Understanding and Measuring the Sources of Variation in the Prioritization of Regression Test Suites, IEEE METRICS 2001
5. A. Egyed, A Scenario-Driven Approach to Trace Dependency Analysis, IEEE Transactions on Software Engineering, Vol. 29, No. 2, February 2003
6. A. Egyed, P. Grünbacher, Automating Requirements Traceability: Beyond the Record & Replay Paradigm, Proceedings 17th International Conference on Automated Software Engineering, ASE 2002, pp. 163-171,Edinburgh
7. A. Egyed, S. Biffl, M. Heindl, P. Grünbacher, Determining the cost-quality trade-off for automated software traceability, ASE 2005: 360-363
8. A. Egyed, S. Biffl, M. Heindl, P. Grünbacher, A value-based approach for understanding cost-benefit trade-offs during automated software traceability, Proc. 3rd int. workshop on Traceability in emerging forms of SE (TEFSE 05), Long Beach, California
9. M.W. Evans, "The Software Factory", John Wiley & Sons, 1989
10. P. G. Frankl , G. Rothermel, K. Sayre, An Empirical Comparison of Two Safe Regression Test Selection Techniques, Proceedings of the 2003 International Symposium on Empirical Software Engineering (ISESE'03)
11. O. C. Z. Gotel, A. C. W. Finkelstein, An analysis of the requirements traceability problem, 1st International Conference on Requirements Engineering, pp. 94-101, 1994
12. S.D.P. Harker, K.D. Eason, The Change and Evolution of Requirements as a Challenge to the Practice of Software Engineering, IEEE, 1992
13. M. Heindl, S. Biffl, A Case Study on Value-Based Requirements Tracing, Proc. ESEC/FSE 2005, pp. 60-69.
14. M. Heindl, S. Biffl, The Impact of Trace Correctness Assumptions, 5th ACM/IEEE International Symposium on Empirical Software Engineering 2006 (ISESE 2006)
15. M. Heindl, S. Biffl, An Initial Tracing Activity Model to Balance Tracing Agility and Formalism - Requirements Tracing Strategies for Change Impact Analysis and Re-Testing, Technical Report, TU Wien, 2007 (http://qse.ifs.tuwien.ac.at/publications)
16. P. Hsia, J. Gao, J. Samuel, D. Kung, Y. Toyoshima, C. Chen, Behavior-based Acceptance Testing of Software Systems: A Formal Scenario Approach, IEEE, 1994
17. J. Huffman Hayes, A. Dekhtyar, S. Karthikeyan Sundaram, Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods, IEEE Trans. on Software Engineering, Vol. 32, No. 1, January 2006
18. J. Jackson, A Keyphrase Based Traceability Scheme, IEE Colloquium on Tools and Techniques for Maintaining Traceability during Design, 1991, pp.2-1-2/4
19. N. Juristo , A. M. Moreno1, S. Vegas, Reviewing 25 Years of Testing Technique Experiments, Journal Empirical Software Engineering, Issue Volume 9, Numbers 1-2 / March, 2004, Pages 7-44
20. H. Kaindl, "The Missing Link in Requirements Engineering", ACM SigSoft Soft. Eng. Notes, vol. 18, no. 2, pp. 30-39, 1993
21. M. Lefering, "An Incremental Integration Tool between Requirements Engineering and Programming in the Large", Proc. IEEE International Symp. on Requirements Engineering, San Diego, California, Jan. 4-6, pp. 82-89, 1993
22. C. Neumüller, P. Grünbacher, Automating Software Traceability in Very Small Companies: A Case Study and Lessons Learned, Proc. IEEE Automated SE 2006, 145-156
23. B. Ramesh, T.Powers, C. Stubbs, M. Edwards, Implementing Requirements Traceability: A Case Study, IEEE, 1995
24. R. Watkins, M. Neal, Why and how of Requirements Tracing, IEEE Software, vol. 11, no. 7, pp. 104-106, July 1994
25. A. Zisman, G. Spanoudakis, E. Perez-Minana, and P. Krause. Tracing software requirements artefacts. 2003.