Published in Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE), in conjunction with the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE), Long Beach, CA. November 8th, 2005.

A Value-Based Approach for Understanding Cost-Benefit Trade-Offs During Automated Software Traceability

Alexander Egyed
Teknowledge Corporation
4640 Admiralty Way
Marina Del Rey, CA, USA
aegyed@acm.org

Stefan Biffl   Matthias Heindl
Software Tech. & Interactive Systems
Vienna University of Technology
A-1040 Vienna, Austria
{biffl,heindl}@qse.ifs.tuwien.ac.at

Paul Grünbacher
Sys. Engineering & Automation
Johannes Kepler University
4040 Linz, Austria
Paul.Gruenbacher@jku.at

ABSTRACT
Many software development standards mandate establishing trace links among software artifacts such as requirements, architectural elements, or source code. However, for typical real-world systems it is currently too expensive and error prone to generate highly detailed trace links. We previously developed an approach to semi-automatically generate trace links and analyzed cost-benefit trade-offs in this context. We consider it as imperative to include value considerations into planning the generation of trace dependencies. This paper discusses three key trade-off decisions for planning the trace generation process: (a) the level of detail of traces among artifacts; (b) the value of the artifacts that are traced; and (c) the points in time of trace generation (early vs. late). We present cost-benefit considerations, empirical data, and argue for a pragmatic value-based planning approach.

Categories and Subject Descriptors
D.2.1 [**Requirements/Specifications**], D.2.7 [**Distribution, Maintenance, and Enhancement**]

General Terms
Documentation, Design, Experimentation.

Keywords
Value-Based Software Engineering, Automated Trace Analysis, Cost-Benefit Trade-Off, Requirements Engineering.

1. INTRODUCTION
Establishing and maintaining trace links places a big burden on software engineers. There are tools available that provide the infrastructure for managing trace links (e.g., case tools, requirements management tools). However, these tools do not free the engineers from identifying links or from ensuring their validity over time. Traceability of any kind is therefore hardly adopted in industry, mainly due to cost and complexity issues [1].

Yet, the generation of trace links is increasingly mandated through standards and industry is moving to adopt these standards and even impose them on subcontractors (such as CMMI level 3).

There is thus a growing need to overcome the traceability problem and researchers have been developing approaches for generating trace links to assist the engineers [1,8,16]. These approaches bring some relief but they strongly rely on the quality of the input. Imprecise and sloppy input generally results in lower-quality trace links, i.e., false trace links (false positives) or missing trace links (false negatives). In the quest to produce a perfect set of trace links (without false positives or false negatives) one tends to forget that there is a significant cost-quality trade-off involved, which affects the usage intensity of trace links in practice.

1.1 Automatic Trace Link Generation
As manual trace link definition tends to be costly and error prone, we use in our studies an automated approach called Trace/Analyzer [7]. This approach uses software-artifacts-to-code mappings as input and generates trace links among software artifacts as output. In simple terms, the Trace/Analyzer approach generates a trace link if and only if two artifacts overlap in their common use of source code[1]. Such overlap may be obscured in various ways (e.g., uncertainty, grouping, utility code) [8], but the results are still usable in practice.

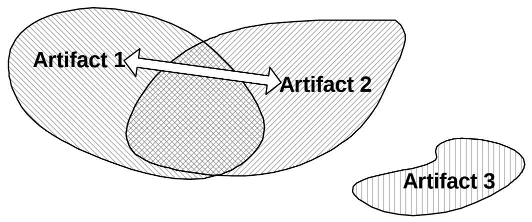

Figure 1. Trace/Analyzer generates a trace link between two artifacts with overlapping code.

However, the quality of the generated trace links is strongly affected by the level of detail of the input. It is up to the engineers to define whether they map artifacts to code on the level of packages, classes, methods, or even individual lines.

1.2 Value-Based Software Engineering
We believe that value considerations are needed for planning software traceability in a sustainable way. Currently, the indirect contribution of trace links to product value is often based on a

[1] Note: not every overlap results in a trace link. In previous work we identified the problem of “utility code” that should be ignored during trace generation. This issue was investigated in detail in [8].

value-neutral and purely technical perception. This leads to an underestimation of the need to align the incentives of success-critical stakeholders to generate and fully exploit the potential of trace links. Some initial results have been reported that consider value aspects in requirements traceability [9,13]. However, these reports have not conducted a cost-benefit analysis to find out when and how intensive tracing is worthwhile in a specific context.

This paper proposes a value-based approach to trace generation and rework. The recently defined paradigm of value-based software engineering [2] brings a fresh perspective on trace generation and maintenance. A motivation for value-based software engineering is that "much of current software engineering practice and research is done in a value-neutral setting, in which every requirement, use case, object, and defect is treated as equally important" [3]. The premise of value-based software development is that not every software artifact should be considered equally important. In the context of software traceability we thus hypothesize that not all trace links are equally important. We expect that taking a value-based perspective can reduce costs by directing efforts to software artifacts with a higher perceived stakeholder value.

1.3 Research Questions: Cost-Benefit Trade-Off of Trace Link Generation Alternatives

The main issues of our research is to investigate the impact of trace link generation quality (effort/cost) on the quality of applications that analyze trace links, such as: change impact analysis completeness analysis, or consistency checking; proof of compliance between contractor and supplier requirements, proof of compliance between acceptance test cases and user requirements, requirements coverage of test cases. A better understanding of this relationship allows improved planning of Intrace generation for aligning the cost-benefit considerations of the involved stakeholders.

In this paper we discuss three key trade-off issues for planning the trace generation process: (a) the level of detail of traces among artifacts (package, class, method levels); (b) the value of the artifacts that are traced (high-value artifacts justify a higher level of tracing effort); and (c) the points in time of trace generation (early vs. late). We present cost-benefit considerations , empirical data, and argue for a pragmatic value-based planning approach.

The simple question we asked ourselves was whether an increase in the quality of trace links does justify the extra costs. While it is out of the scope of this paper to provide a generally valid answer, we can describe cost-benefit implications of trace generation and later trace link rework. We also suggest value-based software engineering [3] as a possible solution to maximize the benefits of trace links in relation to their cost.

The following three issues are discussed in more detail in the remainder of this paper: (1) Is a perfect set of artifact-to-code mappings necessary as input to trace link generation? Are false positives and false negatives acceptable? What are the implications of errors in trace links? (2) Is it necessary to have all trace links on the same level of detail? Are trace links of different quality acceptable? (3) What happens if we discover during trace generation that a previously computed trace links is of insufficient quality for the planned purpose?

2. QUALITY ISSUES OF TRACE DEPENDENCIES

The benefits of trace links[2] are a direct function of their usefulness to applications/humans consuming them. Trace links are consumed by applications like requirements conflict analysis, consistency checking, and change impact analysis. Engineers use trace links for navigation to quickly locate related artifacts. The benefits of trace links depend on the project context, the contribution of the application to the project, and the quality of the traces as input to the application. The benefit has to be determined in context as precondition to optimize the investment in input.

Trace links are either generated manually (by the engineers) or (semi-)automatically based on some initial input. Furthermore, trace links can be generated as soon as new artifacts are created or ”on demand”, i.e., right before the analysis of trace links. These parameters may have significant impact on the cost and benefit of trace links in a project.

Manual trace generation needs systematic guidance. Otherwise, the quality of traces may be insufficient for applications. Also, the generation of trace links at the time they are needed by an application may cause significant delays at sensitive points of time (e.g., during the final stages before acceptance by the customer). Furthermore, the original developers may no longer be available or important details of their work may have been forgotten leading to a much more expensive and error-prone identification of the trace links.

(Semi-)automated approaches typically require some input but are able to compute (some) trace links without additional intervention and at significantly less additional cost than needed for manual trace generation[3]. In case of the Trace/Analyzer approach, the input is given in form of initial software artifacts to source code mappings. This input has to be generated manually or through testing as discussed in [7]. The advantage of the Trace/Analyzer approach is that the required input only rises linearly with the size of the software product although the number of trace links normally rises exponentially [10]. As discussed above, the input to the Trace/Analyzer can be provided at arbitrary levels of detail – mappings between artifact and packages, classes, methods, or, even, lines of code. Irrespective of the level of detail, the input may contain errors (i.e., a wrong or missing mapping) which negatively affect the correctness of the generated trace links.

3. KEY DECISIONS OF TRACE PLANNING

In this section we take a look at three key dimensions of trace generation that have an impact on trace planning and the cost-benefit of the involved stakeholder: (a) empirical data on the investment into trace link generation at different levels of detail; (b) trade-off models for the investment in links among artifacts of

[2] In this paper we focus on direct benefits from trace link applications; note that important indirect benefits such as standard compliance may a major motivation for tracing.

[3] Note that changes in the mix of manual and automated tracing can have significant impact on the cost of trace generation; for simplicity we assume in this work a stable mix of manual and automated tracing.

different levels of value; and (c) a cost-benefit trade-off example regarding investment at different points in time.

3.1 Empirical Studies on Level of Detail and Quality of Trace Link Generation

In recent empirical studies [12,13] we found only limited economic value in improving the level of detail of trace links beyond a certain level. Some trace applications may not need trace link input on a very fine-grained level of detail. Further, we observed a diseconomy of scale: the quality of trace links did not rise linearly with the effort invested for tracing on a more detailed level, but additional investment had lower additional impact on the quality of generated trace links (depicted below in Figure 2 for the open-source ArgoUML modeling tool suite).

ArgoUML consists of 49 packages, 645 classes, and over 6000 methods. The requirements-to-classes-input was an order of magnitude more expensive to generate than the requirements-to-packages-input. In turn, the requirements-to-methods-input was another order of magnitude more expensive to generate. However, one would expect that an increasing level detail would define the code overlap in more exact terms and thus produce better quality trace links. We thus hypothesized that the level of granularity of the input directly affects the quality of the generated links (i.e., its false positive rate). Figure 2 confirms this hypothesis by depicting a decreasing rate of false positives (y-axis) with increasing level of detail (x-axis). However, the number of false positives was not reduced in a linear rate.

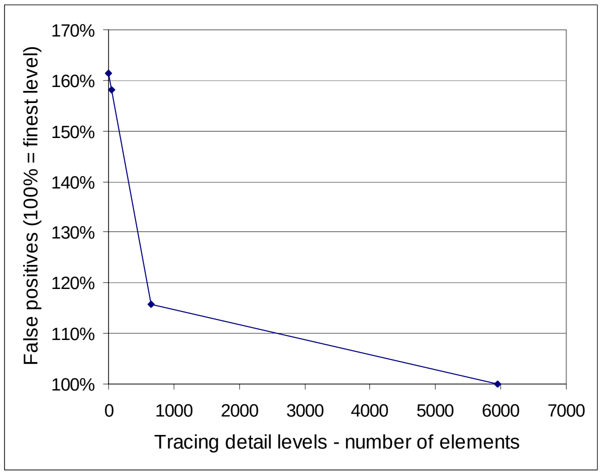

Figure 2. Marginally decreasing share of false trace links with strongly increasing input level of detail.

A 10-fold effort increase by providing input in form of classes (more detail) instead of packages resulted in a 42% reduction of (the known set of) false positives in the set of generated trace links among the input software artifacts. This is only a 2-fold increase in output quality for a 10-fold increase in input cost. Even worse, another 10-fold increase in input cost by providing the input in form of mappings to methods instead of classes only resulted in an additional 16% reduction of (the known set of) false positives. This is a rather low increase in quality for another 10-fold increase in cost (see incline of slopes in Figure 2). Note: since we did not know the actual number of false positives, we based this data on the finest level of granularity (method-level). This level is labeled 100% on the y-axis.

We conducted similar experiments with two other case studies [12] and observed similar diseconomies of scale. It must be noted that these experiments only considered the initial cost of generating trace links with the Trace/Analyzer approach. There is also a cost for maintaining trace links over time. Thus we assess a lower boundary of the true cost of traceability.

3.2 Value of Artifacts and Value of Trace Links

It is generally true that applications consuming traceability links can produce 100% correct results only if the input is 100% correct (in some cases not even then). Consumers of trace links are no exception. Since it is hard to produce 100% correct and complete trace links, it is clear that the application will suffer. We showed earlier that it is one order of magnitude cheaper to produce input for the Trace/Analyzer on the level of artifact-to-class mapping instead of the artifact-to-method mapping. This significant saving only results in a 16% quality reduction (false positives) of the resulting trace links.

However, such an across-the-board quality reduction is a value-neutral solution because it affects the quality of all trace links equally. While an engineer may be willing to sacrifice benefits to save cost, we believe that such a process must be guidable. A better solution would be to generate trace links of some minimal quality initially and to rework them later when a higher quality is needed. This solution assumes that (a) not every trace link is needed and generating and reworking trace links is wasteful in cases where they are not needed; (b) some applications may only require a certain quality and generating and reworking trace links is wasteful if the quality improvements do not translate into benefits.

Value-based software engineering places value on different software artifacts. For example, requirements can be classified as *critical*, *important*, or *nice to have*. Even if the "nice to have" requirements are implemented, their correctness is not as important as "critical" requirements. We believe the Trace/Analyzer should be enhanced to consider such value information through the use of granularity: low-value artifacts are mapped to the class level; high-value artifacts are mapped to the method level.

Table 1. Artifact value and resulting trace link quality.

		Artifact 1	
		Low value	**High value**
Artifact 2	**Low value**	Low-detail trace link	Low-detail trace link
	High value	Low-detail trace link	High-detail trace link

In other words, a higher-value artifact, such as a critical requirement, is mapped on a finer level of detail than a lower-value artifact. Since the Trace/Analyzer determines trace links based on overlaps, it will produce trace links of the highest quality among high-value artifacts because of their finer-grained

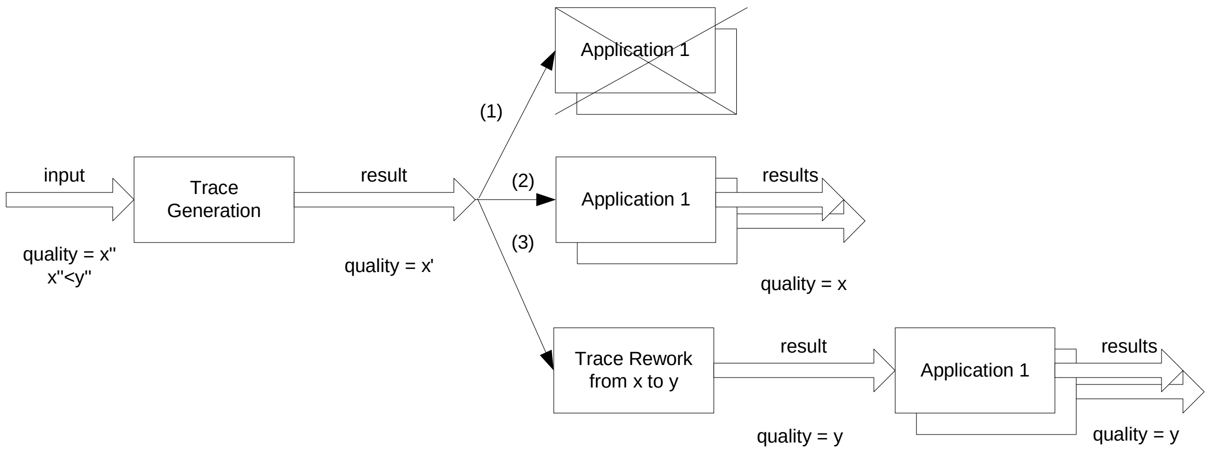

Figure 3. The cost-benefit implication of trace generation and trace rework

overlaps. Likewise, the quality of trace links among lower-value artifacts is worse because their overlaps are based on a coarser-grained level of detail. Table 1 summarizes the four types of overlaps and their quality implications. The value classification thus directly translates to quality implications for trace links – and even more importantly, it also translates to their use in applications. For example, the requirements conflict analysis produces higher quality conflicts among higher quality requirements. However, this solution still places equal value on all pieces of source code. That is, the artifact-to-code mapping is done equally for the entire code of an artifact. This is also unnecessary. Trace links are established on the basis of overlaps. If there is no overlap between two high-value artifacts, then there is no need to create artifact-to-code mappings entirely on a finer level of detail. Only the overlaps among high-value requirements need a finer-grained level of detail. This is because the quality of a trace link is a direct result of the *weakest level of granularity* of the involved artifacts.

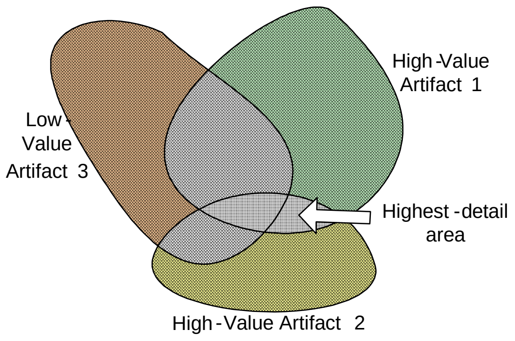

Figure 4. High level of detail only necessary for overlaps among high-value artifacts.

Figure 4 depicts this issue for three artifacts where artifacts 1 and 2 are of high value and artifact 3 is of low value. Areas of artifacts that do not overlap with other artifacts do not cause trace links. There is thus no benefit in investing effort into these areas. Areas where a high-value artifact overlaps with a low-value artifact only need to be considered on the level of granularity of the low-level artifact. There is no benefit in investing effort in increasing the granularity of one of the overlapping artifacts without doing the same for the other artifact. Only the overlapping areas of high-value artifacts should be of the same, finest level of detail.

But how do we know about the overlapping areas before we have gathered and analyzed the input of the Trace/Analyzer? This can be done by initially requiring all input in form of the coarsest-level of detail. The mapping of those overlaps can then be refined, if they belong to high-value artifacts.

3.3 Investment into Tracing at Different Times in Development

The trace links are, by themselves, of little benefit. Their benefit is usually a direct result of their usefulness to support applications that require trace links as input. For example, we previously developed an approach to requirements conflict analysis [11] that takes requirements, requirements trace links, and requirements classifications as input and computes potential requirements conflicts as a result. A false trace link (false positive) may result in a false conflict and a missing trace link (false negative) may result in a missing conflict. The quality of the trace links thus directly affects the quality of the requirements conflict analysis (an application).

If the quality of the generated trace links is low then the quality of the results produced by applications consuming these links is also reduced. Thus, an important value consideration is how much is "good enough" [3]? Is there a quality threshold trace links must meet for them to be useful for follow-on applications? Since low-quality traces are cheaper to produce than high-quality traces, a follow-on question is whether there is a cost-benefit trade-off where the cost of producing higher-quality traces is higher than the benefits gained by their use? Recall that in case of the Trace/Analyzer approach, it takes an order of magnitude more input cost to marginally improve the quality of trace links. It is not obvious that this increase in input cost is justifiable in a project context. Figure 3 depicts three options for possible cost-benefit implications of trace generation, based on the assumption of .less-than-perfect traces, i.e. daily tracing reality.

Trace generation requires a certain input and produces some trace links as output. The cost of this input typically directly relates to effort (usually with some manual overhead), must be offset by the benefits gained from the results of the trace analysis applications.

If the quality of trace links is below the level of usefulness for some application, then the trace links serve no real purpose. Thus, the cost can never be recouped as there are no benefits. Trace generation in this situation is not economical (option 1).

If the quality of the trace links is above the usefulness threshold then the trace links are useful to applications and generate some benefit. The benefit is offset not only by the cost of the trace generation but also the cost of the application. However, an important consideration is that higher-quality trace links do not necessarily translate into more benefits. As we have seen in Figure 2, increasing the quality of trace links can come at a high cost which may never be recouped by the application. Traces of too high quality may thus also uneconomical (option 2).

If the trace generation produces trace links of insufficient quality then there is the option of a later trace rework (option 3). Trace rework improves the quality of trace links, but the cost of trace generation followed by later trace rework is likely to be higher than having done the initial trace generation to the desired quality because (1) knowledgeable engineers may have left the project or (2) they may not remember the solution details well enough. Trace rework is thus a way of improving the quality of trace links to make them useful for applications but at the expense of additional cost which reduces the cost-benefit ratio.

3.4 Impact of change over time on trace quality

The initial saved effort on trace generation is counteracted by loss of rework in case the less detailed traces are inadequate. The amount of rework depends on how much of the work not done has to be done and how much more difficult this is relative to generating a trace at development time by the original developers.

However, trace links also degrade over time while the software product evolves (i.e., as the product changes, its traces must be updated). Consequently their application suffers. Therefore, the benefits of the application of trace links change with time even if the input stays the same (assuming that the software product is evolved during that time). Even (semi-)automated approaches are affected by this erosion. For example, every source code change potentially affects the mapping between the artifacts and source code and thus the input to the Trace/Analyzer approach may become increasingly incorrect over time. It follows that the trace links generated by the Trace/Analyzer approach decrease in quality over time. Trace rework is thus necessary even if the initial trace generation produced sufficient quality trace links. To minimize the cost of trace rework, it should be done at the same time the software product is changed to avoid delays during their application and to benefit from the fresh knowledge. Still, it is not obvious what changes to a software product cause changes to its trace links, which may keep engineers from keeping trace links current and thus loose the potential benefit.

In summary trace planning has to face difficult decisions: a low trace quality may be a cost saving measure initially but it may factually be counter productive because low-quality trace links may not be useful later and thus generate no benefit. A high traceability quality may be needlessly expensive and thus may also be counter productive. And, the cost of trace rework must be considered, especially if the trace links are generated early on while the software product evolves.

4. Related Work

A work similar to our approach has been presented in [5]. This paper describes an approach named TraCS (Traceability for Complex Systems) to maximize the return-on-investment of the requirements traceability effort through the strategic deployment of a heterogeneous set of traceability techniques. Links are established strategically to optimize the ROI while minimizing the risk inherent to software evolution.

The approach presented in this paper is also related to other approaches aiming at automating requirements traceability.

Antoniol *et al.* discuss a technique for automatically recovering traceability links between object-oriented design models and code based on determining the similarity of paired elements from design and code [1]. Murphy *et al.* [14] present an approach for automating the identification of links between high-level models and source code based on software reflexion models.

Spanoudakis *et al.* [16] have contributed a rule-based approach for automatically generating and maintaining traceability relations. Cysneiros, Zisman, and Spanoudakis have also demonstrated how their approach allows to establish links between organizational models specified in *i** and software systems models represented in UML [6].

In the Goal-Centric Traceability (GCT) approach Cleland-Huang *et al.* model non-functional requirements and their interdependencies as softgoals in a Softgoal Interdependency Graph. In their approach a probabilistic network model is used to retrieve links between classes affected by a functional change and elements within the graph [4].

A forward engineering approach is taken by Richardson and Green [15] in the area of program synthesis. Traceability relations are automatically derived between parts of a specification and parts of the synthesized program.

These approaches however, do not consider cost-quality considerations. The recently defined paradigm of value-based software engineering [2,3] brings a new view into the trace analysis research area. Taking a value-based perspective can help save cost and by emphasizing investing effort on software artifacts with a perceived higher stakeholder value. Some initial results have been reported that consider value aspects in requirements traceability [9,12,13]. However, these approaches have not conducted a cost-quality analysis to find out when and how intensive tracing in a specific context is worthwhile.

5. CONCLUSIONS AND FURTHER WORK

As traceability is mandated by software standards, software engineers and managers need support to plan the generation of trace links: (a) the level of detail of trace links between artifacts and (b) the effort for trace generation at different times during development.

In this paper we applied principles of value-based software engineering to traceability and raised the issue of the value of trace links and the level of effort investment into generating and maintaining/reworking trace links. Based on an initial cost-benefit model we explored several options to guide the effort of trace generation with three parameters: (a) the level of detail of traces among artifacts (package, class, method levels); (b) the value of the artifacts that are traced (high-value artifacts justify a higher level of tracing effort); and (c) the points in time of trace generation (early vs. late).

While we could show the need for better understanding the cost and benefit of both trace generation and usage during trace analysis, we see some fundamental open issues that need further work and discussion at the workshop: (1) How can we determine the benefit of traceability in some tangible measure such as “saved engineering hours” that allows balancing these benefits with the investment into trace generation?; (2) How can we describe the relationship between the quality of input traces to trace analysis application and the quality of the output of such analysis applications (compare Figure 3)?

The ability to answer these questions will largely determine whether an alignment of the stakeholder views on cost and benefits of tracing can be achieved, which in turn will determine the rate of adoption of tracing in practice.

6. REFERENCES

[1] Antoniol G., Caprile B., Potrich A., and Tonella P.: Design-Code Traceability Recovery: Selecting the Basic Linkage Properties. Journal Science of Computer Programming 40(2-3), 2001, 213-234.

[2] Biffl, S., Aurum, A., Boehm, B. W., et al: Value-based Software Engineering. Springer Verlag, 2005.

[3] Boehm B.: Value-Based Software Engineering. Software Engineering Notes 28(2), 2003, 1-12.

[4] Cleland-Huang J., Settimi R., BenKhadra O., Berezhanskaya E., and Christina S.: "Goal-centric traceability for managing non-functional requirements," Proceedings of the 27th International Conference on Software Engineering, St. Louis, MO, 2005, pp.362-371.

[5] Cleland-Huang, J., Zemont, G., and Lukasik, W.: "A Heterogeneous Solution for Improving the Return on Investment of Requirements Traceability," Proceedings of the International Conference on Requirements Engineering (RE), Kyoto, Japan, 2004, pp.230-239.

[6] Cysneiros F.G., Zisman A., and Spanoudakis G.A.: " Traceability Approach for i* and UML Models," Proceedings of Software Engineering for Large-Scale Multi-Agent Systems Workshop Report (SELMAS 03), Portland, OR, 2003.

[7] Egyed A.: A Scenario-Driven Approach to Trace Dependency Analysis. IEEE Transactions on Software Engineering (TSE) 29(2), 2003, 116-132.

[8] Egyed, A.: "Resolving Uncertainties during Trace Analysis," Proceedings of the 12th ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE), Newport Beach, California, November 2004, pp.3-12.

[9] Egyed, A.: "Tailoring Software Traceability to Value-Based Needs," Book Chapter in Value-Based Software Engineering, Springer Verlag, 2005.

[10] Egyed, A. and Grünbacher, P.: "Automating Requirements Traceability - Beyond the Record and Replay Paradigm," Proceedings of the 17th International Conference on Automated Software Engineering (ASE), Edinburgh, Scottland, UK, September 2002, pp.pp. 163-171.

[11] Egyed A. and Grünbacher P.: Identifying Requirements Conflicts and Cooperation: How Quality Attributes and Automated Traceability Can Help. IEEE Software 21(6), 2004, 50-58.

[12] Egyed, A., Heindl, M., Biffl, S., and Grünbacher, P.: "Determining the Cost-Quality Trade-off for Automated Software Traceability," Proceedings 20th IEEE/ACM Int. Conference on Automated Software Engineering (ASE), Long Beach, CA, 2005, pp.(to appear as a 4-page short paper).

[13] Heindl, M. and Biffl, S.: "A Process for Value-based Requirements Tracing - A Case Study on the Impact on Cost and Benefit," Proceedings of the European Software Engineering Conference and Foundations of Software Engineering (ESEC/FSE), Lisboa, Portugal, September 2005.

[14] Murphy, G. C., Notkin, D., and Sullivan, K.: "Software Reflexion Models: Bridging the Gap Between Source and High-Level Models," Proceedings of the 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering, New York, NY, October 1995, pp.18-28.

[15] Richardson, J. and Green, J.: "Automating traceability for generated software artifacts," Proceedings of the 19th Automated Software Engineering Conference (ASE), Linz, Austria, 2004, pp.24-33.

[16] Spanoudakis G., Zisman A., Perez-Minana E., and Krause P. Rule-based generation of requirements traceability relations. Journal of Systems and Software 72(2), 2004, 105-127.