A Heterogeneous Solution for Improving the Return on Investment of Requirements Traceability

Jane Cleland-Huang , Grant Zemont, Wiktor Lukasik
School of Computer Science, Telecommunications, and Information Systems
DePaul University
jhuang@cs.depaul.edu, grant@zemont.com, wiktor@wjlukasik.net

Abstract

This paper describes a best-of-breed approach to traceability, in which the return-on-investment of the requirements traceability effort is maximized through the strategic deployment of a heterogeneous set of traceability techniques. This contrasts with typical traceability practices that tend to utilize a single technique such as a matrix or tool embedded into a requirements management package even though it may not provide the optimal solution for the traceability needs of a diverse set of requirements. The proposed solution, named TraCS (Traceability for Complex Systems), defines project level trace strategies for categories of requirements and establishes links strategically in order to optimize returns of the traceability effort and minimize the risk inherent to software evolution. The paper provides a rationale for heterogeneous traceability, describes an extensible traceability framework, and then defines the process for establishing project level trace strategies. It concludes with an example drawn from a system to control chemical reactions at a catalyst plant.

1. Introduction

Requirements traceability provides useful support for many software engineering activities such as requirements validation, impact analysis, regression testing, and for tracking the rationales and trade-offs behind each requirement and its related implementation decisions [1,2,3,4]. At one end of the traceability spectrum, organizations developing high-assurance systems may invest considerable time and effort in the traceability process, while at the other end many organizations simply determine that the return-on-investment of the traceability effort does not warrant its inclusion within their software development process. Even when an organization does implement a traceability process, its effectiveness is often tempered by ongoing difficulties in establishing, maintaining, and fully utilizing traceability links. Gotel and Finkelstein reported on an extensive study in which they examined the problems and causes related to traceability [5,6]. Among other problems they determined that developers frequently failed to maintain links because of the inherent difficulties in coordinating tasks amongst team-members, and because in many cases the cost of implementing the trace appeared to the developers to outweigh its benefits. Domges et al reported that the excessive use of traceability can result in a tangle of links that are hard to maintain and very difficult to utilize [3], indicating that it simply may not make sense to trace every requirement at a fine-grained level. This observation was borne out by Ramesh who conducted an extensive study into industrial traceability practices and reported that more sophisticated traceability users recognized this problem and responded by combining the use of fine-grained and coarse-grained links according to factors such as the criticality of the requirement [7,8].

In addition to the need to apply different trace granularities, different types of requirements may also need to be traced in different ways. Functional requirements can generally be traced using traditional techniques such as matrices, graphs, or tools embedded into existing Requirements Management packages such as DOORS and Requisite Pro, while other types of requirements can often be more effectively traced using different and less traditional techniques. For example Event-based traceability (EBT) techniques can be used to trace performance related requirements to executable performance models [9,10,11], and quality requirements that tend to be implemented through known design patterns [12], can be linked using the invariants of those patterns as intermediaries [13]. Recent research has also shown that information retrieval (IR) techniques can be used to dynamically retrieve traceability links [14,15,16,17,18]. These dynamic techniques provide traceability without the need for link maintenance; however their usefulness is limited to tracing requirements in which there is a high lexical correlation between the requirement and traced artifact, and for requirement traces

Proceedings of the 12th IEEE International Requirements Engineering Conference (RE'04)
1090-705X/04 $ 20.00 IEEE

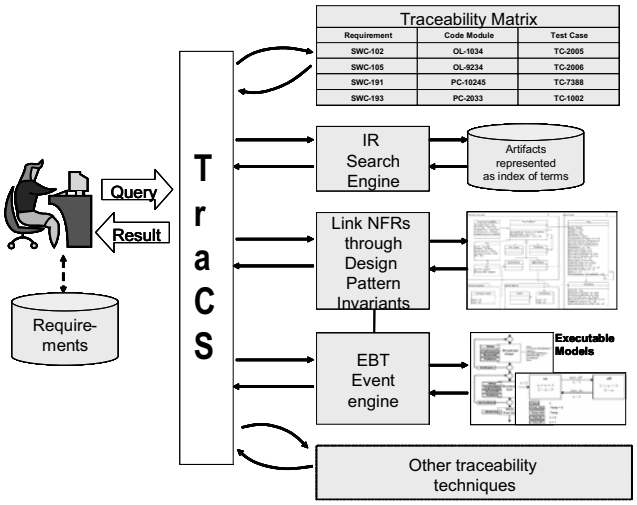


Figure 1. Extensible traceability system for supporting heterogeneous techniques.

in which the level of achievable precision from the IR technique is sufficient for the criticality of the trace.

There are benefits and tradeoffs related to the use of different types of traceability techniques, and combining these techniques into an integrated traceability scheme enables the deployment of a traceability process that is cost effective and can return significant value to an organization. This paper therefore proposes a best-of-breed approach to traceability in which heterogeneous techniques are selected and applied as appropriate according to the characteristics and traceability needs of each requirement. Individual traceability decisions are guided within the context of carefully defined project level trace strategies. A heterogeneous traceability framework named TraCS (Traceability for Complex Systems) is also defined. TraCS provides an extensible framework for integrating heterogeneous traceability techniques, in addition to a well-designed user interface that provides sufficient transparency to enable users to issue traceability queries without paying undue attention to the underlying traceability mechanism. When implemented, TraCS offers the potential for significant improvement in the return on investment (ROI) of requirements traceability.

In order to provide a context for this discussion, the following section briefly describes four different traceability techniques, and explains the advantages and disadvantages of each one. Section 3 then discusses the tension that exists between the traceability needs of an individual requirement and the realities of a sustainable project-level traceability process. Section 4 formulates a generic set of project level trace strategies, and section 5 explains how these high-level strategies can be used to guide individual traceability decisions at the requirement level. It also provides a financial model for comparing costs and returns of various trace methods. Section 6 then describes our TraCS prototype, and demonstrates its ability to facilitate heterogeneous traceability through providing a user-interface that hides much of the underlying trace mechanism from the general user. Finally, section 7 concludes with a discussion of the implications of this approach and ideas for future work.

2. Best of Breed Traceability

The requirements traceability solutions described in this paper are based on the premise that careful selection and application of appropriate traceability techniques results in an effective maintainable traceability scheme that delivers greater ROI than would be feasible from any single technique. This approach is depicted in Figure 1. The trade-offs of four such techniques are now discussed.

2.1 Simple Links

The simplest form of traceability involves establishing a link between two artifacts by using a traceability matrix [19, 20], hypertext link [21,22], graph [23], or a traceability tool embedded into a requirements management system such as DOORS. A requirement is traced to its impacted artifacts by use of a simple query that returns the set of all directly and indirectly impacted artifacts. The primary strengths of this approach are its simplicity and the fact that it is well used and understood in industry, and supported by numerous requirements management tools. The approach is applicable for many different requirements and artifacts. In practice many organizations apply this approach without attaching attributes to each link, thereby limiting their ability to support automated and meaningful trace queries. The major weaknesses of this approach include the long-term difficulty of maintaining large numbers of links, and the static nature of the links that limit the scope of potential automation.

2.2 Semantically Retrieved Links

Information Retrieval (IR) can be used in certain situations to dynamically generate links in place of user-defined explicit links. A query is constructed from the key words of the requirement to be traced, and based on the similarity of the query with artifacts in the search space, the retrieval algorithm then returns a set of likely links to the user. Research results in this area have returned links with varying recall and precision rates [14,15,17], and have demonstrated that in certain circumstances IR techniques provide a viable alternative to explicit user-defined links. Spanoudakis et al demonstrated that traceability links could similarly be established between requirements and UML documents by defining a set of heuristics that govern the generation of

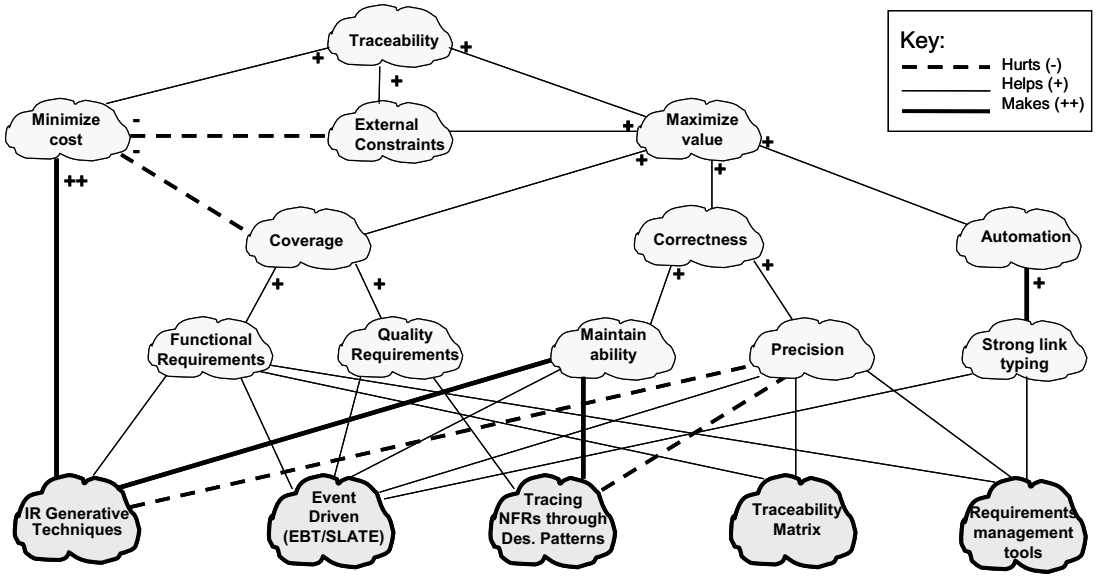


Figure 2. Softgoal Interdependency Goal showing traceability trade-offs.

specific types of links [18]. However IR based approaches can only work effectively if and when there is a high lexical correlation between the requirement and its impacted artifact, and are therefore less effective for certain types of quality requirements that may have a more global impact upon a system. IR techniques may also not be appropriate for tracing highly critical requirements as the level of achievable precision may not meet the required traceability standards. The primary advantage of IR traceability techniques is that they eliminate the need for maintaining links, and when applicable can eradicate the problem of outdated and incomplete links.

2.3 Executable Links

Certain types of requirements, especially performance related ones such as response time and throughput, are validated through the use of executable models and simulations [24,25]. In these cases, traceability links can be established between quantitative performance requirements and the variables within those models. These techniques have been demonstrated in previous work on Event-based traceability (EBT) [10,11] and have been implemented on a limited scale within commercially available tools such as SLATE (System Level Automation Tool for Engineers) [26]. Executable traceability links support a broad array of activities such as speculative impact analysis, in which a speculative change in a quantitatively defined performance requirement results in the propagation of the change to all impacted performance models, the injection of the speculative values into those models, subsequent re-execution of the models, and system-wide reporting of the impact of the change on the system. The strengths of the approach are its ability to trace the impact of changing requirements on the performance of the system.

2.4 Tracing NFRs through Design Patterns

Gross and Yu suggested the possibility of tracing non-functional requirements through the use of design patterns [12]. Cleland-Huang et al built on this idea by showing that pattern detection algorithms [27] could be employed to trace these NFRs to the code and UML models in which they are implemented [13]. When incorporated with coarse grained links, which establish the general cluster of classes in which a requirement is fulfilled, this technique can achieve fine-grained traceability at a low cost. This approach suffers from the same precision problems as its IR counterparts, but enables traceability coverage of a group of non-functional requirements that were previously only traceable through an extensive network of explicit links.

2.5 Heterogeneous Traceability Model

These examples by no means provide a complete coverage of all requirements traceability techniques. They do however serve to illustrate that different types of requirements are best traced using different methods, and also that competing techniques introduce their own tradeoffs at both the individual requirement level and at the project level. This observation suggests that an effective traceability solution might well utilize the strengths of more than one technique within a single project. The following section discusses some of the issues involved in selecting the correct technique for each individual requirement. A 'greedy' approach would result

in each requirement using the technique that best supported its own needs, however such an approach could result in the excessive use of links at the project level, and create a non-sustainable situation. Individual traceability decisions must therefore be made within the context of project level trace objectives.

3. Project Level Trace Objectives

The primary goal of traceability is to create a satisfactory ROI of the traceability effort through delivering sufficient benefits at reasonable cost and effort, where the term 'benefits' is defined according to the needs of the project. For example a high-assurance system might require a high-degree of traceability precision in order to ensure the safety of the system through ongoing validation of its critical requirements. In this case, high traceability costs would be justified by the benefits of additional support for mitigating risks during change management. In a less critical system for which safety were not an issue, the benefits of traceability might be measured solely according to the financial returns of the traceability effort.

In the Softgoal Interdependency graph (SIG) of Figure 2, the traceability softgoal is decomposed into sub-goals of *minimize costs* and *maximize value*. Satisficing these subgoals helps the traceability effort because either one can improve the ROI of the traceability effort. The additional sub-goal of *external constraints* represents the need to conform to laws and organizational regulations concerning traceability standards. The goal of *maximizing value* is then further decomposed into the sub-goals of *coverage*, *correctness,* and *automation,* each of which is discussed below. The traceability techniques described in the previous section, are then inserted as leaf nodes into the SIG to represent operationalizations, and the primary trade-offs between these operationalizations and each of the higher level goals are depicted and used to develop a set of generic project level trace strategies subsequently used for guiding lower level traceability decisions.

3.1 Automation

For the purposes of this paper, automation is defined as *the extent to which a traceability link supports automated queries*. Ramesh highlighted the importance of automation for supporting queries and increasing the usefulness of traceability links, in his analysis of the results of his study into traceability practices [7]. There are three primary levels of automation. These are non-automated, semi-automated, or fully-automated. Non-automated links require the user to manually traverse the links to identify related artifacts. Semi-automated links support queries that return lists of artifacts meeting the criterion of the query, while fully-automated links provide much higher-level support for utilizing the links to identify impacted artifacts, and for transmitting data and commands that trigger events within the impacted artifacts [26,10,11]. Because automation simplifies the task of utilizing existing links, and increases the usefulness of those links, it enhances the value of a traceability link.

3.2 Correctness

Correctness is the *extent to which a set of links conforms to the actual relationships that exist between artifacts.* Correctness problems come in two primary flavors, dependent upon the type of link. User-defined links suffer from correctness problems if developers initially create inaccurate links or if they fail to maintain existing links in an accurate state [6,9]. In contrast, dynamically generated links suffer from precision problems due to limitations in the algorithm or process used to generate links. Correctness is therefore refined into the sub-goals of maintainability and precision.

Maintainability is defined as the *ease by which an existing set of correct traceability links can be preserved and evolved in an accurate state.* The difficulties of maintaining a set of traceability links have been well documented by both practitioners and researchers [28, 6], and can result in the gradual degradation of traceability links over the lifetime of the system. Furthermore, the problem increases exponentially as the number of traceability links increase. An effective traceability solution must therefore provide support for the long-term maintenance and evolution of the artifacts it traces.

Maintainability can be improved by reducing the number of links; however this has the adverse effect of decreasing the scope of coverage of the traceability scheme. It is also possible to replace explicit links that require ongoing maintenance with dynamically generated ones. This approach introduces an additional trade-off, because existing techniques tend to generate links that

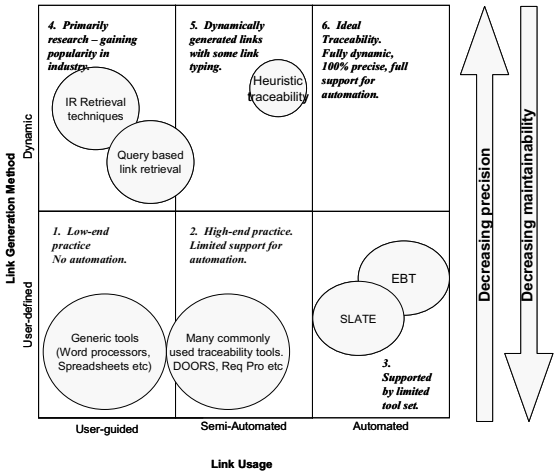


Figure 3. Current traceability practices

support only a limited level of automation. This trade-off is depicted in Figure 3, which plots traceability techniques according to their support for automation vs. their dynamic generative ability. This graph does not depict the precision of each technique, however in the ideal position, marked as area "6" on the graph, traceability links are dynamically generated, fully automated, and 100% precise, in order to eliminate the need for maintaining explicit traceability links. In reality, current practices tend to primarily fall in areas 1-5 of the graph. Area 1 depicts traceability techniques such as matrices that require developers to define links explicitly and that provide little support for automation. Techniques in areas 2 and 3 require the user to manually define links and related attributes, which then support some level of automation of traceability supported activities. Techniques in areas 4 and 5 generate links dynamically but then require user guided usage of these links. Traceability heuristics [18] offer promise for the generation of strongly typed and potentially automated links, however the general problem of precision still exists.

Maintainability can also be improved through explicit support for the change process. Most traceability tools such as DOORS, Requisite Pro, and Caliber, now implement a feature known as 'suspect links', to indicate where a link may point to a changed or outdated artifact. This is useful, but does not explicitly address the problem of providing support for the maintenance effort. Other tools such as EBT provide more specific support for link and artifact maintenance through applying the publish-subscribe metaphor to create traceability links, and recording change events in transaction logs to support a just-in-time resolution of change [9].

Precision is calculated by dividing the number of relevant links retrieved by the total number of links retrieved, and therefore measures the *exactness of the retrieval algorithm in discovering a correct set of links*. Precision is impacted by the sophistication of the algorithm and by characteristics of the requirements and other artifacts being traced. IR techniques typically attach a 'likelihood' index to each potential link, and only return those links that score above a certain threshold.

3.3 Coverage

Coverage is defined as the *extent to which the traceability scheme provides support for all requirements regardless of their type, geographical location, or level of abstraction*. As this definition suggests, there are three primary dimensions. These are 'depth' defined as *support for requirements at different levels of abstraction and criticality*; 'breadth', defined as *support for requirements of different types*; and 'geographical distribution', defined as *support for requirements and their related artifacts regardless of their local or remote allocation*.

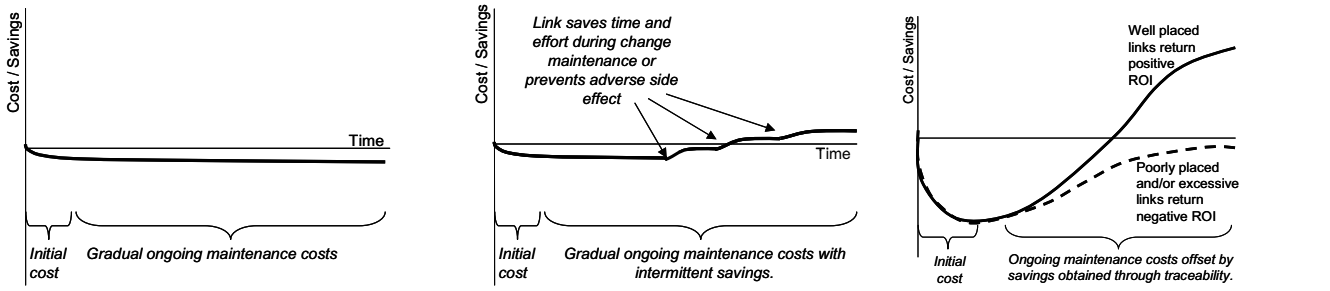
Furthermore, the concept of coverage also measures the extent to which all requirements are traced within each of these dimensions. Many existing traceability solutions provide adequate support for *depth* and *geographical* coverage, but *breadth* coverage introduces a greater challenge. This is illustrated by the fact that many organizations currently focus their traceability efforts almost entirely on functional requirements and fail to trace a much broader spectrum of non-functional requirements. As discussed in Section 2 of this paper, it is only by implementing diverse traceability strategies that traceability coverage can be expanded to a broader spectrum of requirements.

Unfortunately in all non-trivial systems, any attempt to trace ALL requirements using traditional traceability techniques, results in a very large number of links that have traditionally proven either very costly or almost impossible to maintain.

3.4 Return-on-investment

If however, the only traceability objective were to optimize value through the sub-goals of *maintainability*, *correctness*, and *automation*, the traceability task could be successfully accomplished through allocation of additional resources. In fact in certain critical systems in which safety is an overarching objective, traceability of critical components must be achieved despite its cost. However, in many business systems, the cost of traceability is clearly an issue, and therefore the softgoal to *minimize cost* acts as a critical constraint on the traceability process. Organizations need to have confidence that their investment into establishing and maintaining traces is worthwhile and will return significant benefits.

Figure 4a depicts the cash flow of a purely cost incurring trace activity termed a 'negative return' trace. In this case, costs are incurred for establishing and maintaining the trace, but no measurable benefits are ever returned. In contrast, Figure 4b represents the cash flow of a beneficial or 'positive return' trace, in which the initial and long-term traceability costs are offset by benefits experienced through using the trace. These benefits can be experienced in several ways such as when a trace enables developers to understand more quickly and more completely the impact of changing a requirement; through averting a major crisis that might have occurred as a result of a poorly understood change; or simply through fulfilling legislative traceability constraints. As a general strategy for achieving a positive project-level ROI, as depicted in Figure 4c, and in which the losses of *negative return* traces are more than offset by the returns of *positive* ones, it is necessary to maximize the number of *positive return* traces, and to minimize the number of *negative return* traces. This can be accomplished through project level trace strategies within a risk-driven process.



a. Negative return" trace occurs when a trace does not return value during change maintenance.

b. "Positive return" trace occurs when a trace saves time and effort.

c. Traceability ROI is positive when savings from positive traces, outweigh negative ones.

Figure 4. Traceability cashflow

4. Trace Strategies

Balancing the somewhat conflicting traceability sub-goals in an actual project requires the formation of project level trace strategies. Without such strategies, individual decisions made during the development process will be unlikely to support the global objectives of the project. This section defines strategies for balancing the goals described in the previous section through the application of appropriate traceability methods.

- ***Maximize the usage of dynamic link generation*** Dynamic link generation, through techniques such as IR or heuristic traceability provides an effective strategy for minimizing the number of links that need to be manually established and maintained. The strategy is therefore to replace explicit user-defined links with dynamically generated ones whenever possible. This strategy is effective when the requirement being traced exhibits sufficiently high lexical correlation with its traced artifacts to support dynamic generation; when the generated links are sufficiently 'typed' to support automation of subsequent software engineering tasks; and finally when the precision of the dynamically generated links is sufficient for the criticality and other characteristics of the requirement being traced.
- ***Trace for a purpose*** In order to minimize *negative return* traces, it is important to evaluate WHY a link is being created, so that the most appropriate and useful type of link can be deployed. If a potential link serves no clear purpose, then it should not be established. The temptation is to establish too many links, but as practice has clearly shown the impracticality of this approach, it is vital that all links serve a clear purpose, and are traced to the correct level of granularity to support that purpose.

Requirements that are traced to a high-level of precision exact a higher cost to establish and maintain, and are therefore only worthwhile under special circumstances such as for tracing highly critical requirements. Furthermore, a traceability system defines the relationships that exist between requirements and multiple artifacts, and it may frequently be the case that a single requirement will trace to several different artifacts, but at varied levels of abstraction. For example, it might be determined that a set of performance related requirements should be carefully and methodically traced to the variables within a performance simulation, while a single higher-level link to a related sequence diagram would be sufficient. The underlying principle is to only establish links that have a clearly defined purpose, and will return clearly defined value to the project.

- ***Select the most effective traceability mechanism*** This strategy proposes that the most effective mechanism be selected for each link. This means that a single requirement could be traced to one artifact using one technique and to another artifact using a different technique. Drawing upon the previous example of the performance requirement, the fine-grained dynamic links from the set of requirements to the performance simulation could be supported using EBT [10,11] or SLATE [26], while the simpler link to the sequence diagram could be supported using an IR generated link. This type of heterogeneous traceability is only feasible if the user interface protects the trace user from the intricacies of multiple trace techniques. In section 6, we discuss TraCS ability to provide this level of transparency.
- ***Differentiate between throw-away and long-term traces*** This strategy differentiates between traces that are useful only during development, and those that should be maintained in the long-term. During development it is important to trace each and every requirement to its implemented components in order to validate that all requirements have been fulfilled. However, as it is often not financially feasible to maintain each of these traces, certain links must either be thrown-away or stored only until such time as they become suspect. A similar concept in relation to modeling and long-term documentation is discussed by Ambler in [29].

5. Applying the Trace Strategies

In this section we describe a two-phase process for determining the right amount and right type of traceability for different parts of the project. In the first phase, high-level use cases or requirements are ranked according to

Use Case / High level requirement	Risk Analysis			Risk Category
	Change	Impact	Risk	
The vat controller shall process a batch of catalysts.	0.9	10	28	Substantial
All batches shall be traced.	0.6	8	17.6	Substantial
An advance schedule shall be maintained for future batch operations.	0.6	8	17.6	Substantial
The system shall manage the inventory of all chemicals stored in the plant.	0.3	10	16	Significant
The system shall schedule operators.	0.9	4	11.2	Significant
Users shall have access to a library of chemical structures and their interactions.	0.3	4	6.4	Nominal
The system shall manage certifications of all plant employees.	0.3	4	6.4	Nominal

Figure 5. "Failure to trace" risks for high-level use cases from a catalyst batch process

potential risks associated with not establishing adequate traceability. These risks are measured by the criticality and projected volatility of the use case. In the second phase, trace strategies are defined in order to provide adequate traceability coverage so as to successfully mitigate these risks. These strategies then constrain the choices that developers can make as they establish traces for lower level requirements related to each use case.

5.1 How much traceability?

Traceability links are designed to reduce the risk of change through providing developers with greater insight into the impact of that change. Risk is typically defined as Risk = Impact X Probability, where probability represents the probability of the occurrence of the risk, and impact represents the effect on the project if that risk actually were to occur.

Within the context of optimizing the ROI of the traceability effort, the objective is to reduce the risk that a costly and avoidable mistake will occur simply because a change is unsupported by traceability links. We define this type of risk as *Failure to Trace* risk (FTR), for which impact is defined as the *potential damage caused by the invalidation of a requirement*, and probability as *the likelihood or probability that this requirement, or the code in which it is implemented, will be changed or impacted by a future change in such a way as to cause its invalidation.* As it is extremely hard to predict whether a change might invalidate a requirement, the definition is simplified to the probability that *a requirement may be changed or impacted by a change.* This simplification enables probabilities to be assigned according to the perceived volatility of either the requirement or the classes and methods in which that requirement is implemented. Volatility of a use case in indicated by factors such as lack of clear stakeholder agreement, or unstable market conditions that might impact the future stability of the requirements.

Impact (I) is assessed using a scale from 0-10, measured as *critical* (10), *substantial* (8), *moderate* (6) or *limited* (4), or any intermediate value. Similarly probability of change (PC) is measured on a scale of 0-1 and categorized as *volatile* (0.9), *changeable* (0.6), or *stable* (0.3). 'Failure to Trace' risk is then calculated as: **FTR = [(PC * 2)+1] * I**.

These values and formula were obtained through analyzing the Software Requirement Specifications and traceability needs for three systems ranging from safety critical to low criticality. These included a system to control the operations of a batch process in a catalyst plant, an online banking system, and a library management system. Impact was weighted more heavily than change because critical requirements must be traced even if there is only a small probability of their being affected by future changes. Use cases are ranked according to FTR values and categorized according to *substantial, significant,* or *nominal* risk. Threshold values for each category must be determined on a project by project basis according to general project characteristics such as whether the system is safety critical.

A trace strategy is then defined to provide guidance for developers as they make low level trace decisions for requirements within each use case. For example, in the case of the Catalyst plant the following strategies were developed for each risk category.

- **Substantial:** Trace ≅ 50% of the most critical requirements explicitly. Trace NFR goals related to performance, security, and safety etc using Event Based Traceability (EBT) (< 5%). Trace remaining requirements using dynamic IR methods (≅ 45%).
- **Significant:** Trace ≅ 5% of the most critical requirements explicitly. Trace < 2% of NFR goals using EBT. Trace remainder of requirements using IR methods.
- **Nominal:** Trace all requirements using IR methods.

These goals are depicted in Figure 6. Assigning quantitative percentages to each use case provides guidance to the developers as they make individual low-level trace decisions and in effect acts very much like a budget. It also provides a metric against which actual traces can be compared in order to evaluate and improve the future efficiency of traceability practices. For example, a developer working on a use case designated as critical knows that she is budgeted to set up explicit links for approximately half the requirements, whereas for use

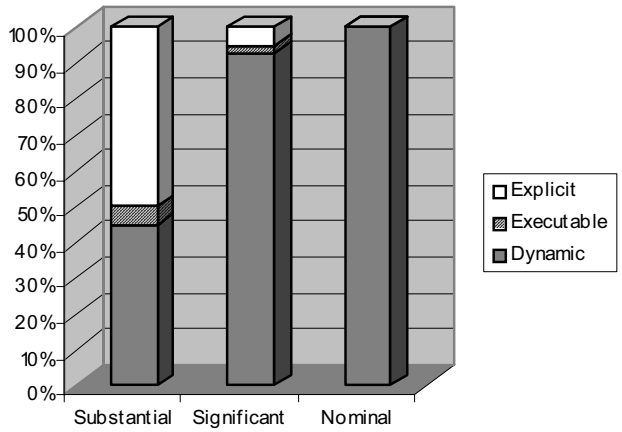


Figure 6. Trace strategies by risk category

cases with nominal risks the strategy dictates that no explicit links should be established. The level of control over traceability decisions is determined by the abstraction level of the requirements or use cases for which FTR is calculated, with the extreme case being that FTR is calculated for each individual requirement.

5.2 Define a Trace Strategy

Developers also need to make meaningful decisions about which artifacts a requirement should be traced to. For example, it may well be sufficient to trace one requirement to its code and related test cases, while another critical performance requirement might warrant additional traces to sequence diagrams and to a software performance execution graph. Applying the strategy of *trace for a purpose* clearly helps achieve project level objectives through minimizing the number of links for each requirement. Developers must determine WHY a requirement needs to be traced, and then select a minimal subset of artifacts that should be traced in order to support this task. The most appropriate explicit traceability technique available is then selected for each selected artifact. As a result of this step, certain artifacts will be designated to be traced, and others will remain untraced.

Remembering that a satisfactory ROI is achieved when the right amount of carefully selected links are deployed, a financial analysis is conducted to evaluate the success of this goal. The primary traceability costs that should be factored into the equation include: 1. The costs of establishing and maintaining explicit links. 2. The costs of executing a manual traceability analysis when a change occurs and no traceability links are available. 3. Unknown penalties that result from errors that occur through failure to adequately trace the impact of a change. These penalties are hard to predict and could range from small and easily fixed errors to events that cause millions of dollars in downtime, or even catastrophic failure of the software system. TraCS minimizes the risk of these penalties through providing more rigorous traceability of critical requirements, and leaving only non-critical requirements with small probability of change entirely untraced.

5.3 A Model for Comparing Trace Techniques

Figure 7 provides a model for comparing the traceability costs of TraCS against other traceability approaches. In this example, 90% of requirements can be traced relatively easily, while the remaining 10% have a more global impact upon the system and are hard to trace. TraCS traces an additional 3% of the requirements using non-traditional techniques described earlier. Although the numbers in this example are not taken from real projects, they are based on reasonable estimates and so provide a realistic analysis of the costs of each method. It is expected that users of this analysis model would plug their own numbers into the spreadsheet in order to evaluate their own traceability options.

The identified costs are discounted to today's values, in order to gain a better understanding for the financial trade-offs of the cost of establishing explicit links during the first year of the project versus deferring expenditures by resorting to manual impact analysis throughout the entire 5 years of the analysis period. A discount rate of 10% per year is applied. It is further assumed that all explicit links are established in year 1, and that manual traceability costs are dispersed equally over all five years of the project. Due to space constraints, these discount calculations are not explained here, however a more complete discussion of present value and associated concepts is given in [30].

In case A, all traceable artifacts are linked using a matrix, at a cost of $15 per link to set up (calculated as 15 minutes of time at $60 per hour), and an estimated average cost of $5 per link to maintain over five years. This figure is low, because many links have no need for change maintenance. In this scenario, no manual traces are needed, and the total cost is calculated at $45,000, with a discounted cost of $40,909. In case B, no traceability links are maintained and all changes are managed through brute force analysis. Each such analysis is calculated to take 1.5 hours at $60 per hour. If we assume that 15% of the total requirements change over five years, then the total cost of performing manual impact analysis is $33,750 or $25,588 in discounted terms. Case C represents a more typical traceability pattern in which developers trace only critical requirements. Costs include link set-up and manual traces for unlinked requirements that change, and calculate to $39,375 or $33,248 in discounted terms. This approach differs from TraCS because it does not take advantage of alternate traceability techniques such as IR or EBT. In the TraCS example of case D, setting up matrix style links for critical requirements costs $12,500, and manually tracing the smaller percentage of non-critical requirements that need to be traced costs $8,625.

	A. Complete matrix coverage	B. No trace-ability coverage	C. Partial matrix coverage	D. Heterogeneous Traceability
No. of requirements	500	500	500	500
Avg. no. of links per requirement	5	5	5	5
% Explicit links	90%	0%	45%	25%
% Dynamic links	0%	0%	0%	45%
% Non-links	0%	90%	45%	23%
% Non-traceable (Certain NFRs etc)	10%	10%	10%	7%
TraCS Risk Assessment overhead	0%	0%	0%	**$4,000**
Explicit links				
Setup cost per link	$15	n/a	$15	$15
Avg. maintenance cost per link over 5 years.	$5	n/a	$5	$5
Total cost of explicit links	**$45,000**	**$0**	**$22,500**	**$12,500**
Manual Traces				
Cost of manual trace	n/a	$90	$90	$90
5 year % of requirements manually traced	n/a	15%	8%	4%
Total cost of manual traces	**$0**	**$33,750**	**$16,875**	**$8,625**
Total Cost of Traceability	**$45,000**	**$33,750**	**$39,375**	**$25,125**
Discounted Cost 10%, Costs dispersed over 5 yrs	**$40,909**	**$25,588**	**$33,248**	**$21,903**
Level of Risk Mitigation	**High**	**Low**	**Medium**	**High**

Figure7. A comparison of traceability costs and risks of various trace methods

An additional overhead cost of $4,000 (20 use cases, 2 hours each, $100 per hour) is added to account for the risk assessment process. These costs sum to $21,125, or $17,903 in terms of discounted cost. The costs of the "no trace" option could be lowered if the system were very stable and requirements did not change significantly following deployment. However, if this were the case, the TraCS process would also generate a less expensive solution involving fewer implicit links, and an increased number of untraced requirements.

6. The TraCS Prototype

The heterogeneous techniques described in this paper have been demonstrated through our TraCS prototype. The prototype tool incorporates explicit user defined links through use of a traceability matrix, dynamic link generation using an IR tool that we have developed, and EBT traces of performance. The prototype allows a user to enter a change request which is then input as a query into the IR tool in order to retrieve a set of potentially impacted requirements. The user then selects a subset of these requirements and requests an impact analysis report. The analysis is performed using the TraCS system, in which each of the selected requirements is traced using any combination of the traceability methods previously designated by the developer for the selected requirement. An integrated impact report is then returned to the user showing all of the results.

Through publishing a standard API, and designing the architecture of the system to simplify the addition of new traceability methods, TraCS enables an organization to incorporate customized trace tools into the framework. In its current state, TraCS requires the developer establishing traces to understand individual tool interfaces, but it successfully hides these details from more casual users who need to query the system.

7. Conclusions

This paper has proposed an approach to traceability in which heterogeneous techniques are applied synergistically to improve the ROI of the traceability effort. This approach enables organizations to fine-tune their traceability effort in order to provide greater support for the more critical and volatile parts of their software systems, and to extend the feasible coverage of requirements traceability by integrating new techniques appropriate for previously untraced sets of requirements. The investment into the traceability process can be guided by the characteristics of the project, and the benefits of tracing different requirements. The TraCS prototype has demonstrated the feasibility of integrating a diverse set of trace methods.

Future work will involve implementing TraCS as a fully extensible system in order to support the integration of a broader spectrum of traceability techniques. We also plan to integrate a guided enactment domain to support the user through the process of selecting appropriate trace methods [31]. Currently users must make these decisions themselves based upon the trace strategies for the current use case. Additional empirical work is also needed to

measure the ROI of individual traces in industrial applications.

Acknowledgments

The work described in this paper was partially funded by NSF grant CCR-0306303. We would also like to acknowledge the help of MS students from DePaul University who were involved in this project. These include Dat Lee who constructed the TraCS prototype, Dolores Santillan who conducted related usability studies, and other students who developed the various trace tools used by TraCS.

References

1. M. Jarke, "Requirements Traceability", *Comm. of the ACM,* Vol. 41, No. 12, Dec. 1998, pp. 32-36.
2. D. Leffingwell, "*Calculating Your Return on Investment from More Effective Requirements Management"*, Rational Corp. http://www.rational.com/products/whitepapers
3. R. Domges and K. Pohl, "Adapting Traceability Environments to Project Specific Needs", *Comm. of the ACM,* Vol. 41, No. 12, 1998, pp. 55-62.
4. R. Chardon and M. Dorfman, "Early Experiences with Requirements Traceability in an Industrial Environment", Industrial Presentation, IEEE International Symposium on Reqs Engineering, Toronto, Sept, 2001.
5. O. Gotel, and A. Finkelstein, "An Analysis of the Requirements Traceability Problem," *Proc. First Int'l Conf. Reqs Eng.,* 1994, pp. 94-101.
6. O. Gotel and A. Finkelstein, "Contribution Structures", *Procs of the 2nd IEEE In,tl symposium on Reqs Eng.,*1995.
7. B. Ramesh, and M. Jarke, "Toward Reference Models for Requirements Traceability", ", *IEEE Trans. on Software Engineering,* Vol. 27, No. 1, Jan 2001, pp. 58-92.
8. B. Ramesh, "Factors Influencing Requirements Traceability Practice", *Comm. of the ACM,* Vol 41, No. 12, Dec. 1998, pp. 37 – 44.
9. J.Cleland-Huang, C.K.Chang, and M. Christensen, "Robust Requirements Traceability for Handling Evolutionary Change", IEEE Trans. on Software Eng, Vol. 29, No. 9, Sept. 2003, pp. 796-810.
10. J. Cleland-Huang, C.K.Chang, and J.Wise, "Automating Performance Related Impact Analysis through Event Based Traceability", Reqs Engineering Journal, Springer-Verlag, Vol. 8, No. 3, Aug. 2003, pp. 171-182.
11. J. Cleland-Huang, C .K.Chang, H. Hu, K. Javvaji, G. Sethi, and J. Xia, "Requirements Driven Impact Analysis of System Performance", *IEEE Proc. of the Joint Conf. on Reqs Eng.*, Essen, Sept. 2002.
12. D. Gross and E. Yu, "From Non-Functional Requirements to Design through Patterns", Reqs Engineering Journal, Vol 6, No. 1, 2001, pp. 18-36.
13. J. Cleland-Huang and D. Schmelzer, "Dynamically Tracing Non-Functional Requirements through Design Pattern Invariants", Workshop on Traceability in Emerging Forms of Software Engineering, Montreal, October, 2003.
14 J.Huffman Hayes, A. Dekhtyar, and J. Osborne, "Improving Requirements Tracing via Information Retrieval", *IEEE Proc. of the Reqs Engineering Conference,* Monterey, CA, Sept. 2003, pp.138-150.
15 J. Cleland-Huang, R. Settimi, W. Lukasik, and Y. Chen, "Dynamic Retrieval of Impacted Software Artifacts", *Midwest Software Eng. Conf.*, Chicago, June 2003.
16. A. Marcus and J. Maletic, "Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing", *IEEE Intn'l Conf on Software Eng,* May 2003, Portland, pp. 125-132.
17. G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, E. Merlo, "Recovering Traceability Links between Code and Documentation", *IEEE Trans. On Software Engineering, Vol. 28, No. 10, pp. 970-983.*
18. G. Spanoudakis, "Plausible and Adaptive Requirements Traceability Structures", 14th ACM International Conference on Software engineering and knowledge engineering, July 15-19, 2002, Ischia, Italy. Pp. 135-142.
19. D. Leffingwell, "*Calculating Your Return on Investment from More Effective Requirements Management"*, Rational Software Corporation. Available online at http://www.rational.com/products/whitepapers
20 A.M. Davis, *Software Requirements: Analysis and Specification,* Prentice-Hall, 1990.
21 H. Kaindle, The Missing Link in Requirements Engineering, *ACM SIGSOFT Software Engineering Notes,* Vol. 18, No. 2, pp.30-39.
22 M. Glinz, "A Lightweight Approach to Consistency of Scenarios and Class Models", *4th Intn'l Conf. On Reqs Engineering,* 2000.
23 F.A.C. Pinheiro and J.A.Goguen, "An Object-Oriented Tool for Tracing Requirements", *IEEE Software*, Vol. 13, No. 2, Mar. 1996, pp. 52-64.
24 R.K. Jain, *The Art of Computer Systems Performance Analysis : Techniques for Experimental Design, Measurement, Simulation, and Modeling,* John Wiley & Sons, April 1991.
25 C. Smith and L.Williams *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*, Addison-Wesley, 2002
26 SLATE (System Level Automation Tool for Enterprises), Electronic Data Systems Corp., http://www.sdrc.com/
27 Dirk Heuzeroth, Thomas Holl, Gustav Högström, Welf Löwe, *Automatic Design Pattern Detection*, 11th International Workshop on Program Comprehension, co-located with 25th International Conference on Software Engineering, Portland, IEEE, May 2003.
28 J. Martin and C. McClure, *Software Maintenance: The Problem and Its Solution*. Englewood Cliffs, N.J. Prentice-Hall, 1983.
29 S.Ambler and R. Jeffries, *Agile Modeling, Effective Practices for Extreme Programming and the Unified Process*, John Wiley & Sons; March 22, 2002
30 M. Denne and J. Cleland-Huang, *Software by Numbers, Low-Risk, High-Return Development,* Prentice-Hall, Oct. 2003.
31 K. Pohl, K. Weidenhaupt, R. Domges, P. Haumer, M. Jarke, and R. Klamma, "PRIME–Toward Process-Integrated Modeling Environments," ACM Trans. Software Eng. and Methodology, Vol. 8, No. 4, Oct. 1999, pp. 343-410.