A STREAMLINED, COST-EFFECTIVE DATABASE APPROACH TO MANAGING REQUIREMENTS TRACEABILITY

BY

Andrew Kannenberg

Submitted to the graduate degree program in Computer Science and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

_____
Dr. Hossein Saiedian
Professor and Thesis Adviser

Committee members: _____
Dr. Gunes Ercal-Ozkaya
Assistant Professor

_____
Dr. Prasad Kulkarni
Assistant Professor

Date defended: _____

The Thesis Committee for Andrew Kannenberg certifies
that this is the approved version of the following thesis:

A STREAMLINED, COST-EFFECTIVE DATABASE APPROACH TO MANAGING REQUIREMENTS TRACEABILITY

Committee:

_____
Dr. Hossein Saiedian
Professor and Thesis Adviser

_____
Dr. Gunes Ercal-Ozkaya
Assistant Professor

_____
Dr. Prasad Kulkarni
Assistant Professor

Date approved: _____

Abstract

Requirements traceability offers many benefits to software projects, and it has been identified as critical for successful development. However, numerous challenges face the implementation of traceability in the software engineering industry. Some of these challenges can be overcome through organizational policy and procedure changes, but the lack of cost-effective traceability models and tools remains an open problem. Many methods of implementing traceability exist, but each implementation method has its own limitations.

A novel, cost-effective solution for the traceability tool problem is proposed, prototyped and tested in a case study using an actual aviation software project. Quantitative metrics from the case study are presented and a qualitative analysis is performed to demonstrate the viability of the proposed solution for the traceability tool problem. The results show that the proposed method offers significant advantages over implementing traceability manually or using existing commercial traceability approaches.

Contents

1 Introduction 1
1.1 Justification 2
1.2 The Traceability Problem 4
1.3 Significance 5
1.4 Expected Contributions 6
1.5 Evaluation Criteria 6
1.6 Thesis Organization 7

2 Background 9
2.1 Traceability Definitions 9
2.1.1 Pre- and Post-Requirements Traceability 10
2.1.2 Traceability Practices 11
2.1.3 Traceability Users 12
2.2 The Importance of Traceability 13
2.2.1 Project Management 14
2.2.2 Process Visibility 15
2.2.3 Verification and Validation 17
2.2.4 Maintenance 19
2.3 Traceability Methods 20
2.3.1 Traceability Matrices 20
2.3.2 Hyperlinks 22
2.3.3 Commercial off the Shelf (COTS) Tools 23
2.3.4 Proposed Methods 25
2.3.5 Other Methods 27
2.4 Challenges Facing Traceability 28
2.4.1 Cost. 28
2.4.2 Managing Change 29
2.4.3 Different Stakeholder Viewpoints 30
2.4.4 Organizational Problems 31
2.4.5 Poor Tool Support 33

3 An Investigation of the Traceability Tool Problem 35
3.1 Traceability Mandates in the Aviation Software Industry 36
3.2 An Analysis of Current Aviation Software Traceability Methods 38

3.2.1 Manual Traceability Methods 39
3.2.2 Telelogic's DOORS 42
3.2.3 Other Methods 47

4 A Solution for the Traceability Tool Problem 49
4.1 A Proposal for a Database-Based Approach to Traceability 49
4.2 Prototyping the Database-Based Approach to Traceability 51
4.2.1 Identifying the Necessary Traceability Data 52
4.2.2 Designing the Database 52
4.2.3 Creating the Software Wrapper for the Database 55

5 A Practical Case Study 62
5.1 Software Project Background 62
5.1.1 Initial Traceability Implementation 63
5.2 Database-Based Traceability Tool Case Study 65
5.2.1 Preparation for Use of the Database Tool 65
5.2.2 Using the Database Tool 66
5.2.3 Reviewing the New Traceability Method 68
5.3 Current State of the Database-Based Traceability Tool 69

6 Evaluation and Analysis 71
6.1 Quantitative Metrics 71
6.1.1 Comparison with Past Project Results Using Manual Methods 71
6.1.2 Cost Comparison with Traceability Alternatives 77
6.2 Qualitative Analysis 80
6.2.1 Database Tool Strengths 80
6.2.2 Potential Areas of Improvement 82

7 Conclusions and Future Work 85
7.1 Conclusions 85
7.2 Summary of Contributions 87
7.3 Future Work 87

Bibliography 89

List of Figures

Figure 2-1. Links in Pre- and Post-Requirements Traceability. 11
Figure 3-1. Traceability Data Required by DO-178B. 37
Figure 3-2. Traceability Methods Used for Aviation Software Projects. 39
Figure 3-3. Example Traceability Data Generated by DOORS (Telelogic 2007). 46
Figure 4-1. Entity-Relationship Diagram for the Database. 53
Figure 4-2. Simplistic Idea for Database Relation. 54
Figure 4-3. Normalized Database Design. 55
Figure 4-4. Database Tool User Interface. 59
Figure 6-1. Development Time Required for Traceability Methods. 72
Figure 6-2. Time Spent on Traceability Activities During a Software Release. 73
Figure 6-3. Time Spent on Traceability Activities at the End of a Software Release 74
Figure 6-4. Number of Errors Detected in the Traceability Data. 76
Figure 6-5. Start-up Cost Comparison. 77
Figure 6-6. Cost Comparison per Software Release. 78

List of Tables

Table 1-1. Comparison of the Standish Group's 1994 and 2006 Results. 3
Table 2-1. Example Traceability Matrix. 20
Table 5-1. Example Traceability Matrix Output from the Database-Based Tool..... 69
Table 7-1. Comparison of the Database-Based Tool with Manual Methods. 86
Table 7-2. Comparison of the Database-Based Tool with Telelogic's DOORS. 86

Chapter 1

Introduction

In modern times, software products have been increasingly deployed in complex and potentially dangerous products such as weapons systems, aircraft, medical devices, spacecraft, and satellites. These products can be viewed as critical because failure of these types of systems could result in loss of life, significant environmental damage, and major financial loss. This might lead one to believe that care would be taken to implement these software products using proven, reproducible methods. Unfortunately, this is not always the case.

In the past, numerous catastrophic software failures have been documented including the Therac 25 incidents (Leveson & Turner 1993), the London ambulance system (Finkelstein & Dowell 1996), and the Ariane 5 launch failure (Nuseibeh 1997). A study performed in 1994 by the Standish Group found that 53% of software projects failed outright and another 31% were challenged by extreme budget overruns. The software engineering discipline was clearly in need of a major overhaul to address these problems.

Many responses to the high rate of software project failures have been proposed. Some of the more well-known examples include the Software Engineering

Institute's Capability Maturity Model (Paulk, Curtis, Chrissis & Weber 1993) which was superseded by the Capability Maturity Model Integration (Chrissis, Konrad & Shrum 2003), the International Organization for Standardization's 9001:2000 (2000) for software development, and the Institute of Electrical and Electronics Engineers' J-STD-016 (1995). The United States government has also issued and/or accepted many standards regulating software development including the DOD-STD-2167A (U.S. DoD 1988) (superseded in 1994 by MIL-STD-498) standard for government contractors and the DO-178B (RTCA 1992) standard for aviation products.

1.1 Justification

One feature all of these standards for software development have in common is that they all impose traceability practices on the software development process (ISO 2000; Paulk et al. 1993; Chrissis et al. 2003; U.S DoD 1988 and 1994; RTCA 1992). Specific examples include the Capability Maturity Model Integration (CMMI) which requires bidirectional traceability of requirements to be implemented for an organization to achieve CMMI maturity level 2 and the DO-178B standard for aviation software which requires traceability to be implemented for aviation software to be certified for use. The fact that traceability is mandated by these standards is not surprising because the engineering and scientific disciplines have stressed the importance of being able to reproduce results long before the age of computing, and traceability provides a technique to do so by mapping the steps taken throughout the lifecycle of a project (Egyed 2001). If this is done comprehensively, an outline of

how a problem is transformed into a solution can be created. This is just as important in software development as it is in other engineering and scientific disciplines (Swartout & Balzer 1982).

Several independent researchers have discovered that inadequate traceability is an important contributing factor to software project failures and budget overruns (Leffingwell 1997; Domges & Pohl 1998). As a response, there has been a recent outpouring of research and literature on the subject of traceability (Ramesh & Jarke 2001), and many companies and governmental institutions have been striving to improve their traceability practices. These efforts have not been in vain. An updated study by the Standish Group in 2006 showed that only 19% of software projects failed outright, with another 46% challenged by budget overruns. These results are compared with the 1994 Standish Group study results in Table 1-1 to show the improvement that has occurred in the software engineering industry.

Table 1-1. Comparison of the Standish Group's 1994 and 2006 Results.

Year	Failed Projects	Challenged Projects	Successful Projects
1994	53%	31%	16%
2006	19%	46%	35%

Clearly the software industry has taken great strides since 1994, but there remains room for improvement.

1.2 The Traceability Problem

Although the importance of traceability appears to be well-accepted in the software engineering industry, research suggests that many organizations still do not understand the principles of traceability and are struggling with implementing traceability practices in the software development lifecycle (Jarke 1998; Ramesh 1998; Ramesh & Jarke 2001; Egyed 2002). The United States Department of Defense serves as an example of this by spending approximately 4 percent of its information technology budget on traceability activities, often without receiving much value for its money. This occurs primarily because the traceability standards are vague, traceability models and mechanisms are not well understood, and the implementation of traceability is haphazard (Ramesh & Jarke 2001).

Because of the many standards mandating traceability as an important practice for software projects, one would expect quality traceability practices to be firmly ingrained throughout the software engineering industry. Unfortunately, this is not the case. Many organizations do not even attempt to implement traceability while others only do so in a haphazard manner.

Why is this? Perhaps it is because manual methods for implementing traceability are time-consuming and error-prone. However, this cannot be the only reason because alternatives to manual traceability methods exist. The International Council on Systems Engineering (2008) has identified 31 different tools that claim to provide full traceability support. In spite of the large number of available traceability tools, the adoption rate throughout industry is surprisingly low. A general study of

the software engineering industry performed by Gills (2005) found that approximately one-third of the organizations studied utilized tools to assist with traceability. Even an aviation software-specific study, a field where traceability is mandated by governmental regulations, discovered that only half of the organizations surveyed use specialized tools to implement traceability (Lempia & Miller 2006).

1.3 Significance

If there truly are 31 tools that provide full support for traceability, then why are they not widely deployed throughout the industry and why are quality traceability practices not more prevalent? The author believes this is because the traceability tools that currently exist are inadequate and provide only simplistic support for traceability activities. If currently existing tools were adequate for the needs of the industry, then it would be reasonable to expect that their adoption rate would be much closer to 100%, especially in the area of aviation software due to its mandates for traceability practices.

Ramesh (1998) found that traceability is error-prone, time-consuming, and impossible to maintain for all but the smallest projects without the use of automated tools. Therefore, it follows that without feasible automated alternatives to manual implementations of traceability, traceability practices for all but the smallest software projects are almost certainly doomed to failure.

1.4 Expected Contributions

The goal of this thesis is to promote improvements in traceability practices in the software engineering industry by studying the feasibility of implementing cost-effective automated traceability techniques for software projects. Many researchers have claimed that existing traceability tools are inadequate and have major shortcomings (Spanoudakis, Zisman, Perez-Minana & Krause 2004; Ramesh and Jarke 2001; Cleland-Huang, Chang & Christensen 2003; Naslavsky, Alspaugh, Richardson & Ziv 2005). Therefore, existing traceability methods and tools will be investigated and evaluated, and their strengths and weaknesses discussed in order to determine if they are inadequate. In addition, a streamlined, cost-effective database-based traceability approach intended to address the shortcomings of existing traceability tools will be proposed, developed, tested, and evaluated. The purpose of this new approach is to devise a novel traceability method that is capable of automating traceability practices in a cost-effective manner without the major shortcomings of existing commercial tools.

1.5 Evaluation Criteria

In order to evaluate the proposed approach to automating traceability activities, the method will be prototyped and tested in a case study using an actual project in the software engineering industry. Metrics will be collected and compared to traceability methods used on the project in the past. These metrics will serve to demonstrate the viability of the proposed approach to traceability in terms of the overall time required

to gather traceability data and generate traceability artifacts as well as the number of errors detected in the resulting traceability data. Metrics will be presented graphically, and the strengths and weaknesses of the proposed approach will be discussed. In addition, a cost analysis will be performed in order to compare the overall cost of implementing traceability using the proposed method in comparison with using manual methods and existing commercial tools. A qualitative analysis will also be performed to further determine the viability of the proposed approach.

1.6 Thesis Organization

This thesis is organized into the following chapters:

- **Chapter 1: Introduction** – An introduction to the problem, its significance, justification for this research, expected contributions, and evaluation criteria for the proposed solution.
- **Chapter 2: Background** – An introduction to traceability, benefits provided by traceability, past and present traceability implementation methods, and a discussion of challenges facing the implementation of traceability.
- **Chapter 3: An Investigation of the Traceability Tool Problem** – A detailed investigation of the problems with existing methods of implementing traceability.
- **Chapter 4: A Proposed Solution to the Traceability Tool Problem** – A streamlined, cost-effective database-based traceability approach is proposed in order to address the problem of the lack of quality traceability tools.

- **Chapter 5: Case Study –** A case study for the proposed solution to the traceability tool problem is described.
- **Chapter 6: Evaluation and Analysis** – The experimental results, quantitative metrics, and qualitative analysis.
- **Chapter 7: Conclusions and Future Work** – The conclusions and future work related to this topic.

Chapter 2

Background

Before it is possible to understand the reasons for the challenges facing the implementation of traceability in the software engineering industry today, it is important to have a good understanding of what traceability is as well as a background in past and present approaches to implementing traceability. This chapter provides an introduction to important traceability concepts including the benefits provided by traceability, traceability implementation methods, and challenges facing the implementation of traceability.

2.1 Traceability Definitions

An understanding of the basic concepts of software traceability is required before more advanced topics such as traceability tools can be studied and understood. The classical definition of traceability was presented in 1994 by Gotel and Finkelstein as "the ability to describe and follow the life of a requirement, in both a forward and backward direction." Although this definition was written a long time ago (at least in terms of computer science development), it is still accurate today and provides several discussion points.

2.1.1 Pre- and Post-Requirements Traceability

The idea that requirement life needs to be described and followed in both a forward and backward direction has given rise to two additional terms: pre-requirements traceability and post-requirements traceability. Pre-requirements traceability describes the life of a requirement in a backward direction while post-requirements traceability describes a requirement's life in a forward direction. Pre-requirements traceability is used to describe the life of a requirement before it was formally defined while post-requirements traceability describes the life of a requirement that results from its formal specification (Li, Vaughn & Saiedian 2002).

Both pre- and post-requirements traceability includes three elements: requirements, artifacts, and links. Requirements can be defined as current or future needs that must be fulfilled (Karlsson 1996); thus, requirements are used to define the capabilities of a system. Artifacts include information produced or modified as a part of the engineering process (Ramesh & Jarke 2001). This term is used to characterize items such as requirements documents, design documents, source code and test cases. Links describe a distinct relationship between two artifacts (Cleland-Huang et al. 2003).

Davis (1990) suggests that complete traceability requires four kinds of links:

- Forward from requirements: Indicates links that go from a requirement to an artifact.
- Backward to requirements: Indicates links that go from an artifact in the development of the system to a requirement.

- Forward to requirements: Indicates links from a source of a requirement to the requirement itself.
- Backward from requirements: Indicates links that go from a requirement to the source of a requirement.

This idea fits well with the definitions of pre- and post-requirements traceability because the first two kinds of links are included in post-requirements traceability and the latter two are a part of pre-requirements traceability. Figure 2-1 illustrates this concept graphically.

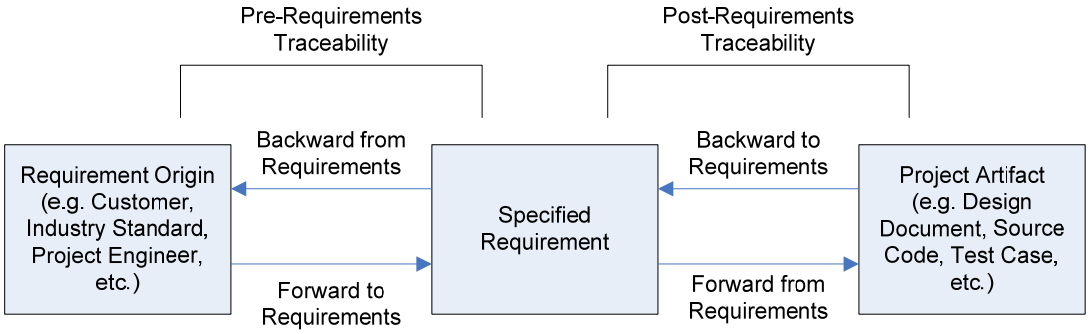

Figure 2-1. Links in Pre- and Post-Requirements Traceability.

2.1.2 Traceability Practices

Heindl and Biffl (2005) suggested that there are three possible practices for performing requirements tracing: ad hoc tracing, full tracing and value based requirements tracing. Each of these methods differs in its level of completeness and overall value to the organization.

Ad hoc tracing refers to development where traceability is not maintained, but is instead created only when it is needed and only for requirements that it is needed

for. Although this may sound efficient, there are many hidden costs in terms of research time and the risk of finding that significant rework is required.

Full tracing is performed when every existing requirement is traced with the same amount of effort and precision. Traces are typically maintained during development. Although there is significantly less risk for rework for full tracing than ad hoc tracing, Heindl and Biffl (2005) argue that it can be highly inefficient and expensive. In spite of the cost, certain projects, such as those following governmental standards, may be required to implement full tracing by the project sponsors.

Value based requirement tracing prioritizes all requirements in the system, and the amount of time and effort expended on tracing each requirement depends on the priority of that requirement. Proper analysis of the importance of each requirement can be difficult to perform, but if done correctly, value based traceability offers many of the benefits of full tracing at a significantly reduced cost.

2.1.3 Traceability Users

Although traceability is commonly practiced in the software industry today, there still remains significant variation in the quality of the practice (Palmer 1997). Because of this, some researchers have suggested dividing traceability users into two groups: low-end users and high-end users (Matthias 1998; Ramesh 1998). These types of users have different viewpoints about the purpose of traceability; therefore, they tend to approach traceability in different ways.

Low-end users typically express an immature attitude towards traceability. They view traceability as something forced upon them as a defense against lawsuits by upper management, project sponsors, or governmental regulation. This is particularly common in safety-critical industries such as the aviation and medical industries (Jarke 1998). Low-end users use simple schemes to implement traceability such as manually created traceability matrices. Many times these traceability schemes are not well-maintained because they are viewed as expensive overhead (Zemont 2005).

High-end users view traceability as an important and cost-effective part of the software development process. Because of this viewpoint, high-end users utilize more complex tools for traceability in order to provide a complete view of the system. Such users are careful to maintain traceability linkages as changes occur throughout the duration of a project (Zemont 2005). These efforts are not wasted. A study by Ramesh in 1998 concluded that software systems built by high-end users have a higher probability of meeting customers' needs and are easier to maintain than those built by low-end users.

2.2 The Importance of Traceability

Traceability has been demonstrated to provide many benefits to organizations that make proper use of traceability techniques. This is why traceability is an important component of many standards for software development such as the CMMI and ISO 9001:2000 for software development. Important benefits from traceability can be

realized in the following areas: project management, process visibility, verification and validation, and maintenance.

2.2.1 Project Management

The benefits of traceability to project management in the software engineering industry are numerous. Traceability provides project managers with the tools that they need to effectively control the development process and manage change, risk, and project finances (Palmer 1997).

Perhaps one of the largest benefits that traceability offers is the ability to manage change. Change in a software project can be very costly, and research has indicated that it is inevitable in software projects (Harker, Eason & Dobson 1993). Traceability offers project managers the ability to estimate the impact of a proposed change by mapping the fan-out impact of the change.

A requirement change proposed by a customer has the potential to impact other requirements documents, project design, code, test cases, and other artifacts depending on how far along the project is in the software development cycle. By following links implemented as a part of traceability, a project manager can quickly see how many artifacts will be affected by a proposed change, and can make an informed decision about the costs and risks associated with that change. Because of this, traceability can act as a bridge between changing customer needs and system evolution (Jarke 1998).

Traceability also gives project managers important insight into the development process for a project. A well-defined traceability scheme allows managers to identify design, code, test plans, and test cases that can be reused within a project (Compuware 2004). Reuse throughout a software project saves development time and money. Managers can also utilize traceability links to identify potential requirements conflicts early in the development process (Heindl & Biffl 2005). These conflicts can be resolved before further development time is spent on them. Early requirement conflict resolution results in significant financial savings because it has been proven that fixing problems late in the software development cycle costs much more than fixing them early (Boehm 2003).

Project managers can also utilize traceability to assist in measuring project progress. As requirements are traced to code and later to test cases, management can estimate the completion status based on how many requirements have been traced to artifacts created later in the development cycle (Zemont 2005). This information can be used to estimate the schedule for a project during development and can be used to assess risk. As an organization's use of traceability matures, they can even utilize traceability on historic projects to make estimates for future projects.

2.2.2 Process Visibility

Another area that traceability can assist with is providing process visibility. Insight into the development process for project managers has already been discussed, but traceability provides process visibility for more than just managers. In fact,

traceability has been found to be important for version control and configuration management of artifacts produced throughout a project lifecycle (Macfarlane & Reilly 1995). These activities are essential for providing process visibility for everyone involved in a software project.

Improved process visibility from traceability can be used to facilitate team communication for the duration of a software project (Compuware 2004). Through traceability, each team member has access to contextual information that can assist them in determining where a requirement came from, its importance, how it was implemented, and how it was tested. This information is essential for requirements to be implemented correctly.

Traceability can also be viewed as a customer satisfaction issue. If a project is audited, or in the case of a lawsuit, traceability can be used to prove that particular requirements were implemented and tested (Watkins & Neal 1994). The availability of this information also increases customer confidence and satisfaction because it reassures customers that they will receive the product that they requested.

Traceability can also be used to comply with standards. Many governmental standards such as DOD-STD-2167A and DO-178B require traceability to be implemented. For the government to accept software projects, they must conform to the traceability requirements imposed by the governing standard under which they were developed. Similarly, for an organization to be certified to the Software Engineering Institute's CMMI level 2, a certain level of traceability must exist.

Higher levels of the CMMI model require even more sophisticated forms of traceability to be implemented (Chrissis et al. 2003).

Another form of process visibility provided by traceability is improved access to information in large documents. Many sizable software projects produce a significant amount of documentation, with each document potentially containing hundreds or thousands of pages. Traceability links can save stakeholders from being forced to manually search through artifacts for related items (Palmer 1997).

2.2.3 Verification and Validation

The most significant benefits provided by traceability can be realized during the verification and validation stages of a software project. Traceability offers the ability to assess the system functionality on a per requirement basis all the way from the origin of each requirement through the testing of each requirement. Without traceability, it is impossible to demonstrate that a system has been fully verified and validated.

Properly implemented, traceability can be used to prove that the system complies with the requirements and that the requirements have been implemented correctly (Ramesh, Stubbs, Powers & Edwards 1995). If a requirement can be traced forward to a design artifact, it validates that the requirement has been designed into the system. Likewise, if a requirement can be traced forward to the code, it validates that the requirement was implemented. Similarly, if a requirement can be traced to a test case, it demonstrates that the requirement has been verified through testing. Test

cases should also trace back to code and code to design to ensure that test cases completely test the code and that the code originated from the design (Ramesh & Jarke 2001; Wiegers 2003; Watkins & Neal 1994). If any of these traces are missing, it means that the design, code, and/or testing needs to be updated in order to complete the tracing so the system can be demonstrated to be fully verified and validated.

Traceability is also important for ensuring that the system is not over-designed or over-implemented. If parts of the design or code cannot be traced back to requirements, this is evidence of the creation of unspecified features, which is known as feature creep or gold-plating (Cleland-Huang et al. 2003; Muvuti & Lungu 2004). Feature creep is a significant drain on both time and resources and should be avoided; however, its presence can be difficult to detect without traceability.

Conflicting requirements can also be identified early using traceability. If conflicting requirements exist, it is impossible to build and successfully verify and validate a system. If the conflicting requirements are not discovered until late in the development process, they are more difficult to correct than if they are discovered early. If requirements trace to each other or trace to the same portion of the design or code, they should be analyzed to determine if they conflict (Egyed & Grunbacher 2004). If so, the requirements should be corrected as soon as possible in order to minimize the cost of correcting them.

Research performed by Gills (2005) indicates that the quality of the testing process is directly related to the quality of the traceability scheme employed. This is because quality testing must be based not only on observable functionality, but also

on requirements, design, and source code. Only traceability can provide the necessary linkages between each of these artifacts to allow for high-quality testing. Without traceability, the system can still be tested to some degree, but systematic testing is impossible.

2.2.4 Maintenance

Traceability is also a valuable tool during the maintenance phase of a software project for many of the same reasons that it is valuable for project management. Initially defined requirements for a software project often change even after the project is completed (Heindl & Biffl 2005), and it is important to be able to assess the potential impact of these changes.

Traceability makes it easy to determine what requirements, design, code, and test cases need to be updated to fulfill a change request made during the maintenance phase of a software project. This allows for estimates of the time and cost required to make a change. The chance of inadvertently failing to update one or more artifacts associated with a change is also lessened when traceability is implemented (Zemont 2005).

In any software project, there is some element of risk that defects will be discovered that will need to be corrected during the maintenance phase of the project. Traceability serves as a risk mitigation factor since it can be used to quickly point out the affected areas of the system (Zemont 2005). This makes defects discovered during maintenance easier to correct in a timely manner.

2.3 Traceability Methods

Now that the importance of traceability has been established, it is important to have an understanding of the methodologies that can be used to implement traceability. Throughout industry, many different methods are used and several theoretical models have been proposed. A complete analysis of all traceability methodologies in existence is beyond the scope of this thesis; however, a brief introduction to some of the more commonly used methods and more interesting theoretical models will be provided.

2.3.1 Traceability Matrices

Traceability matrices are the simplest method that can be used to capture traceability information. A traceability matrix can be defined as "a table that illustrates logical links between individual functional requirements and other system artifacts" (Wiegers 2003). Since traceability matrices are in tabular form, they typically are created using a spreadsheet or a table in a word processor and are independent of the artifacts that they capture traceability information for. An example of a traceability matrix is shown in Table 2-1.

Table 2-1. Example Traceability Matrix.

System Requirement	Software Requirement	Design Element	Code Module	Test Case
005-00150-80#00505	005-00150-85#00112	Airspeed Calculation	calculate_airspeed()	tc_103.doc
005-00150-80#00506	005-00150-85#00234	Airspeed Display	display_airspeed()	tc_125.doc

Table 2-1 demonstrates several important traceability matrix concepts. Requirements are listed using unique identification values. This is done to ensure that the precise requirement being traced is clear, and it also makes it easier to locate a particular requirement in a requirements management system. Requirements can trace to other requirements in a traceability matrix; in this example, high-level system requirements are traced to lower-level software requirements which are then traced to design elements, code modules, and test cases. The exact artifacts included in a traceability matrix may vary on a project-by-project basis, but it is reasonable to expect at least one set of requirements, design, code, and test cases to appear.

Traceability matrices offer several advantages. They are simple to implement and do not require the use of special tools. This is important because a study performed by Gills (2005) of 32 software projects from information technology companies that implement traceability found that 53.7% of the projects did not make use of special tools to assist with traceability practices. For smaller projects, manually created traceability matrices are ideal since they are simple to create and do not require specialized tool support. Additionally, traceability matrices show links in both a forward and backward direction which provides visibility into the overall structure of the system.

Unfortunately, there are several disadvantages to traceability matrices. Because these matrices are created manually, they require a significant amount of work to create for larger projects. As a software system grows in size and complexity, the number of links that need to be captured in a traceability matrix

grows exponentially (Cleland-Huang et al. 2003). After the traceability matrix is fully created, it must be maintained whenever changes are made to the system. This requires discipline and a large amount of manual link checking throughout the traceability matrix. Because of this, it is easy for a traceability matrix to become out of sync with the current set of requirements and other system artifacts. Therefore, traceability matrices are not well-suited for large projects or projects that experience a significant amount of change.

2.3.2 Hyperlinks

Hyperlinks can be used as an alternative to traceability matrices for implementing traceability. Many of the same strengths and weaknesses are shared by each of these methods. Hyperlinks implement traceability by representing traceability relationships as hyperlinks between elements of the project artifacts. These hyperlinks can be embedded directly in the artifacts themselves, or they can be stored independently in a traceability matrix of hyperlinks.

The main advantage of hyperlinks is that they can be followed to quickly analyze traceability relationships between artifacts. Certain projects may also benefit from the ability to embed traceability information within existing artifacts without needing to create a separate traceability artifact. Hyperlinks can be directional or non-directional, which allows for forward or backwards traceability or both, depending on the needs of the project (Munson & Nguyen 2005).

Similar to traceability matrices, hyperlinks provide the advantages of being simple and not necessarily requiring the use of special tools. However, use of hyperlinks may require project artifacts to be stored in a hypertext compatible format such as HTML or XML. Hyperlinks also share the disadvantages of traceability matrices in that they can be tedious to create and maintain for large projects or projects that experience a significant amount of change. Therefore, hyperlinks may be ideal for smaller projects, but other methods may be better for large projects.

2.3.3 Commercial off the Shelf (COTS) Tools

Many commercial off the shelf tools exist that claim to assist with the implementation of traceability. The International Council on Systems Engineering (2008) has a survey which lists 31 distinct tools which claim to offer full support for traceability analysis. Many of these tools are obscure and not widely used while others such as Telelogic's DOORS (2007) and IBM Rational Software's RequisitePro (2007) have seen wide acceptance in industry.

Providing a complete overview of all COTS tools for traceability is beyond the scope of this thesis; therefore, only general statements about the capabilities and advantages and disadvantages of these types of tools will be presented as background information. A more detailed study of one of the more popular tools, Telelogic's DOORS, is presented in Chapter 3. Each tool has its own strengths and weaknesses, but all of these types of tools share several key features, benefits, and limitations.

Three common aspects of traceability are supported by the COTS tools available today: identifying inconsistencies, providing visibility into existing links from source to implementation, and verification of requirements (Li et al. 2002). COTS tools allow users to identity inconsistencies such as untraced requirements or other system elements. The robustness of this feature varies between tools, but all traceability COTS tools provide at least primitive support for this feature. Such tools allow users to follow links in both a backward and forward direction in order to see precisely where each link comes from and goes. Some tools offer graphical support for this feature which can speed link navigation. Verification that requirements have been implemented and tested is also supported in COTS tools. The status of individual requirements can be monitored, and events can be triggered when the status of specific requirements change.

COTS tools provide an advantage in that COTS tool users are not responsible for maintaining a separate method of traceability implementation. Traceability information is stored inside of the tool, and reports showing the project's traceability can be generated on demand. Additionally, these tools can highlight links that have become suspect due to changes in the system. This reduces the difficulty of maintaining traceability information when the system undergoes change.

Unfortunately, there are also many disadvantages to using COTS tools. Cost is one major disadvantage. Although the licensing fees vary per tool, the price tends to be thousands of dollars up front per license in addition to yearly maintenance fees. Because of this, the cost of using COTS tools is often prohibitive, even for fairly

small teams. Such tools are also decoupled from the development environment, meaning that important traceability information such as code modules that implement requirements may not be available (Naslavsky et al. 2005). For this reason, Ramesh (1998) has concluded that COTS tools are mostly used by low-end users and have "very limited utility in capturing dynamic traceability information."

COTS tools are typically marketed as complete requirements management packages, which means that traceability is only one added feature (Gills 2005). The traceability features usually only work if the project methodology is based around the tool itself. Unless the project is developed from the ground up using a particular tool, the tool is unable to provide much benefit without significant rework. Support for heterogeneous computing environments is also lacking (Song, Hasling, Mangla & Sherman 1998).

2.3.4 Proposed Methods

Several methods for partially automating the implementation of traceability beyond the simplistic automation present in currently available COTS tools have been proposed in the literature. Unfortunately, COTS tool support for these methods is not widely available, which means that an organization would need to develop in-house tools in order to use them. Because of this, only a brief background description of these methodologies is provided. Interested readers are encouraged to view the references provided for each method for a complete overview.

Event-based traceability has been proposed as a method for automating much of the traceability process based upon change events (Cleland-Huang et al. 2003). In this method, changes in the system are events which trigger updates to the traceability data. The authors admit that this methodology has not been previously supported; therefore, they developed their own proprietary tool in order to test the feasibility of the system. Initial results appear to demonstrate the feasibility of event-based traceability, but longer-term studies are currently underway.

Scenario-based traceability has also been proposed for partially automating traceability (Egyed 2001). This method generates traceability data based on test scenarios which are executed on a working system. For this system to function, three things are required: a working system, a software model of the system, and executable test cases or scenarios. This means that scenario-based traceability is not feasible during the early stages of development of a project. Tool support is also lacking, as various tools can be used to assist with the process, but none are available that fully implement scenario-based traceability (Zemont 2005).

Automated information retrieval techniques have also been proposed. These methods use an indexing process and a querying mechanism to establish links between artifacts which are returned to the user (Zemont 2005). Unfortunately, information retrieval is hampered by a significant error rate, where incorrect traceability links are returned. This means that manual intervention is necessary to verify that the linkages returned are correct. The speed of information retrieval mechanisms is typically at odds with the amount of precision returned in the results

(Hayes, Dekhtyar, & Osborne 2003). Therefore, not only does information retrieval require the use of special information retrieval tools designed to return traceability information, but it also can be a slow process that lacks precise results.

2.3.5 Other Methods

Many additional methods for representing traceability have been proposed, but it is beyond the scope of this thesis to analyze every method of providing traceability in existence. Many of these other methods are not widely used but are mentioned here for completeness. Interested readers are directed to investigate the sources referenced for each method for further information.

Additional methodologies for implementing traceability include cross-referencing schemes (Evans 1989), keyphrase dependencies (Jackson 1991), templates (Interactive Development Environments 1991), integration documents (Lefering 1993), assumption-based truth maintenance networks (Smithers, Tang & Tomes 1991), and constraint networks (Bowen, O'Grady & Smith 1990). Each of these methods provide unique methodologies for implementing traceability; however, tools for many of these methods are hard to obtain, and they are not widely utilized in practice.

It should also be mentioned that in spite of all the benefits provided by traceability, certain projects may not need it. For example, a project with a very short development cycle may not need the information provided by traceability (Watkins &

Neal 1994). Additionally, some organizations develop their own custom tools and techniques for implementing traceability.

2.4 Challenges Facing Traceability

In spite of the benefits that traceability offers to the software engineering industry, its practice faces many challenges. These challenges can be identified under the areas of cost in terms of time and effort, the difficulty of maintaining traceability through change, different viewpoints on traceability held by various project stakeholders, organizational problems and politics, and poor tool support.

2.4.1 Cost

Probably the biggest challenge facing the implementation of traceability is simply the costs involved. If traceability could be implemented easily and cheaply, every project would use it. Unfortunately, this is not the case. As a system grows in size and complexity, capturing the requirement traces quickly becomes complex and expensive (Heindl & Biffl 2005). Because of this, the initial budget for a project implementing traceability must be greater than that of a project without it. These initial costs will be offset later in the development cycle through the benefits that traceability provides, but the high up-front costs can be a deterrent.

One method of dealing with the high cost of traceability is to practice value based requirement tracing instead of full tracing (Heindl & Biffl 2005). Since value based requirement tracing focuses on the most important requirements instead of

tracing all requirements equally, it can save a significant amount of time and effort. However, for this tracing practice to work, there needs to be a clear understanding of the importance of each requirement in the system. Additionally, value based requirement tracing might not be an option if full tracing is a requirement of the customer or the development process standards used for the project.

Alternatively, the high costs of traceability can be approached with the attitude that the initial costs will save much greater costs further along in the development process due to the benefits that traceability offers in the areas of management, verification and validation, and maintenance. This method does not solve the problem of the high up-front costs involved with traceability, but it does promote a healthy attitude towards managing costs for the entire duration of a project instead of merely looking at the short-term.

2.4.2 Managing Change

Maintaining traceability through changes to the system is another significant challenge. Studies have shown that change can be expected throughout the lifecycle of almost every software project (Wiegers 2003; Boehm 2003). Whenever such changes occur, it is necessary to update the traceability data to reflect these changes. This requires discipline on the part of those making the change to update the traceability data, and it can be costly in terms of time and effort when the changes are extensive. Unfortunately, strong discipline in maintaining the accuracy of traceability is uncommon, leading to a practice of disregarding traceability information in many

organizations (Clarke, Harrision, Ossher & Tarr 1999). This is unfortunate because most of the benefits of traceability are lost if this occurs.

Dealing with change and its impact on traceability is a difficult prospect. Some COTS tools offer assistance with identifying the impact of change on the existing traceability data; however, much manual time and effort is still required to update the traceability data (Cleland-Huang, Chang & Ge 2002). Alternatively, training can help users understand the importance of discipline in maintaining traceability data when changes occur. Focusing on the long-term benefits of traceability instead of the short-term costs can help an organization sustain a healthy attitude toward the costs of maintaining traceability data amidst change.

2.4.3 Different Stakeholder Viewpoints

A contributing factor to poor support for traceability may be the fact that many different viewpoints regarding traceability exist, even among different stakeholders on a project. These different viewpoints exist primarily because current software engineering standards typically require traceability to be implemented but provide little guidance as to why and how it should be performed (Ramesh & Jarke 2001).

Project sponsors and upper management often view traceability as something that needs to be implemented merely to comply with standards (Ramesh 1998). This leads to a desire to spend as little time as possible on traceability because the benefits outside of standards compliance are not well-understood. This viewpoint will likely

conflict with that of project engineers familiar with the importance of traceability who will want to ensure that the traceability performed is complete and correct.

The perceived traceability needs of each project stakeholder can differ based on their individual goals and priorities (Ramesh & Edwards 1993). This can lead to a lack of cooperation and coordination between different stakeholders responsible for maintaining traceability for a project. This makes it difficult to keep traceability data in sync with the system as it changes which in turn can lead to less reliance on the traceability data if it is viewed as being inaccurate.

Perhaps the best way to deal with the problem of different stakeholder viewpoints on traceability is to create an organizational policy on traceability to apply uniformly to all projects. Because the standards requiring traceability are vague, organizations have a lot of leeway to set their own procedures in place for implementing traceability. This can reduce the amount of confusion about traceability, and leads to more consistent viewpoints among the stakeholders involved.

2.4.4 Organizational Problems

Organizational problems also provide a significant challenge to the implementation of traceability. Many organizations that are composed primarily of low-end users view traceability as a mandate from sponsors or for compliance with standards (Ramesh 1998). Typically these organizations do not have a commitment to comprehensive

traceability practices. This leads to an ad-hoc practice of traceability, where traceability data is created and maintained haphazardly.

Lack of training poses another challenge (Gotel & Finkelstein 1994). Many organizations do not train their employees about the importance of traceability and this subject is typically not emphasized in undergraduate education at universities. This can lead to resentment on the part of those tasked with creating and maintaining traceability information. They may view the added workload as impacting their productivity due to a lack of understanding of why traceability is important.

Politics can also play a role. Individuals may be concerned that traceability data will be used against them in performance reviews or as a threat to their job security (Jarke 1998). This issue can arise because the individual who captures a piece of traceability information is usually not the one who makes use of it later. Those involved with creating and maintaining traceability data may feel that they are helping others to look good while reducing their own productivity.

The easiest way to correct organizational problems related to traceability is through use of policy and training. If an organization has clear policies in place about traceability and provides training on how to comply with these policies, it is likely that traceability will be implemented in a thorough manner consistent with policy (Ramesh 1998).

Traceability data should never be used for performance evaluations (Ramesh 1998). Doing so just makes people resentful. Instead, incentives should be offered

for those involved with traceability to help ameliorate the fact that the creators and maintainers of traceability data are often not the ones who benefit from its existence.

2.4.5 Poor Tool Support

Poor tool support is perhaps one of the biggest challenges to the implementation of traceability. Even though INCOSE (2008) has listed 31 different tools that claim to provide full traceability support, existing tools provide only simplistic support for traceability (Ramesh & Jarke 2001). Surprisingly, the tools that are available do not fully automate the entire traceability process; instead, they require users to manually update many aspects of the traceability data. This has led some researchers to conclude that poor tool support is the root cause for the lack of implementation of traceability (Spanoudakis et al. 2004).

Although most tools do support the identification of impacted artifacts when changes occur, they typically do not provide assistance with updating the traceability links or ensuring that the links and affected artifacts are updated in a timely manner (Cleland-Huang et al. 2003). This means that even when tools are used, the traceability information is not always maintained, nor can it always be trusted to be up to date and accurate. This problem is exacerbated by the fact that tools typically only allow primitive actions to be taken in regards to traceability.

Another issue with tools is that they often suffer problems with poor integration and inflexibility (Gotel & Finkelstein 1994). This has led at least one researcher to conclude that existing traceability tools have been developed mostly for

research purposes, and that many projects are still waiting for tools that do not require a particular development or testing methodology (Gills 2005).

Few solutions are available for the problem of poor tool support for traceability. Many organizations shun COTS tools altogether due to their high cost and inflexibility and instead make use of manual methods such as traceability matrices. Another approach common among high-end users is to develop elaborate in-house tools and utilities to implement traceability (Ramesh & Jarke 2001). Unfortunately, this approach is not always feasible because many organizations do not have the manpower or the knowledge necessary to develop such tools. Therefore, poor tool support for traceability remains an open problem at this time, a problem that is investigated further in Chapter 3.

Chapter 3

An Investigation of the Traceability Tool Problem

The lack of quality traceability tools for automating traceability activities is a serious problem in the software engineering industry because it is a known fact that as a system grows in size and complexity, the amount of time and effort required to manually capture traceability data grows exponentially (Cleland-Huang et al. 2003). This leads some organizations to discard traceability completely. This is not a good approach because traceability provides many important benefits to software engineering projects. Additionally, many software projects are driven by governmental or customer mandates to implement traceability.

This chapter performs an investigation into the problem of the lack of quality traceability tools with a focus on the aviation software sector of the software engineering industry. Aviation software was chosen for this investigation because software in this sector is required to implement traceability by the Federal Aviation Administration (FAA), a branch of the government that oversees and certifies software intended for use in aviation. Because of this, aviation software developers have significant motivation to utilize the best available traceability tools since they are required to implement traceability by the government. This motivation is not

necessarily present in the software engineering industry as a whole because software projects in most other fields are not required to implement traceability.

3.1 Traceability Mandates in the Aviation Software Industry

Many commercial software projects are able to get by without implementing traceability. It is likely that the quality of the product suffers in these cases, but most commercial projects do not have regulations governing their development that mandate traceability to be implemented. This is not the case in the aviation software industry. Traceability is a non-negotiable software quality attribute in aviation software due to strict requirements for traceability that are enforced by the FAA. For aviation software to be certified for use, it must meet criteria imposed by certain certification specifications such as RTCA's Software Considerations in Airborne Systems and Equipment Certification (DO-178B) (1992). Several of these criteria are related to traceability.

Specifically, DO-178B mandates the following forms of traceability:

- Between system requirements and software design data
- Between system requirements and software requirements
- Between software requirements and source code
- Between software requirements and test cases
- Between source code and test cases

System requirements are high-level requirements that provide an abstracted view of the complete software system. The software design flows from the high-level

system requirements. Software requirements are detailed requirements about how the system works. Typically, these requirements are derived from the high-level system requirements. Source code and test cases are primarily driven by the software requirements. DO-178B also includes mandates about the amount of code coverage that must be gathered by test cases based on the criticality of the functions implemented by the code. Therefore, test cases are also partially driven by the source code.

In addition, each of the traceability links required by DO-178B must be bi-directional. However, it is not required that separate traceability artifacts be produced for each of these traceability mandates. It is acceptable to present a single traceability artifact that demonstrates how traceability flows throughout the system from system requirements through test cases. This concept is illustrated graphically in Figure 3-1.

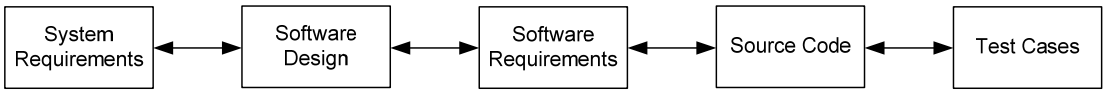

Figure 3-1. Traceability Data Required by DO-178B.

DO-178B's traceability mandates are similar to recommended traceability practices throughout the software engineering industry. However, unlike general software engineering projects, compliance with traceability mandates must be demonstrated in order for aviation software to be certified for use. Compliance is shown through the creation and review of traceability artifacts. Reviews are typically performed by trained designated engineering representatives (DERs) who work at or

consult for the company creating the software. When performing reviews, DERs are considered to be working for the FAA. If an artifact is not accepted by a DER during a review, it must continue to be revised and re-reviewed until the DER accepts the artifact before the corresponding software can be certified. Traceability documentation must also be retained and presented to the FAA for review upon request or in the case of an audit.

3.2 An Analysis of Current Aviation Software Traceability Methods

A study by Lempia and Miller (2006) of companies known to be working in the aviation software industry found that approximately half of these companies use manual methods of implementing traceability by capturing traceability data in general purpose office software such as Microsoft Word or Excel. Of the companies that utilize traceability tools of any kind, nearly all of them use Telelogic's DOORS. A few companies developed their own proprietary tools, and a small number use other tools such as IBM's Requisite Pro. The full breakdown of traceability methods used throughout the aviation software industry is shown graphically in Figure 3-2.

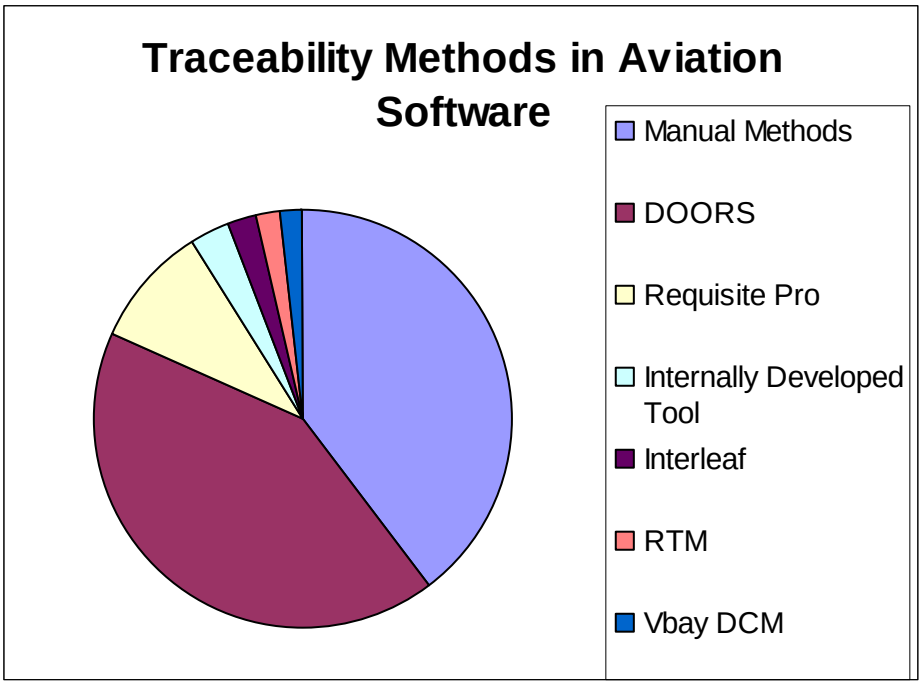

Figure 3-2. Traceability Methods Used for Aviation Software Projects.

There is a nearly even split in the aviation software industry between companies that use manual traceability methods by capturing traceability data in general purpose office software and those that use Telelogic's DOORS to partially automate the process. Only a small number of companies use other tools. Therefore, manual traceability methods and DOORS have been selected for further analysis since they are the most commonly used traceability methods throughout the industry.

3.2.1 Manual Traceability Methods

Manual traceability methods are those which require all traceability information to be captured manually. Traceability data is typically recorded in general purpose office

software such as in a spreadsheet or a word-processor document. Usually the traceability data is presented in tabular form in what is known as a traceability matrix.

Manual traceability methods do have some advantages. They do not require any special tools to create, and they are simple to edit. This makes them ideal for small projects that do not have a large number of requirements. Traceability matrices also show links in both a forward and backward direction which meets one of the DO-178B requirements for traceability artifacts. Because of this, tools that partially automate the traceability process often present traceability data in the form of automatically generated traceability matrices.

Unfortunately, the disadvantages of manual traceability methods far outweigh the advantages for medium and large software projects. Cleland-Huang et al. (2003) found that the number of traceability links that need to be captured grows exponentially with the size and complexity of the software system. This means that manually capturing traceability data for a large software project requires an extreme amount of time and effort. In the author's own experience working on a large aviation software project, a manually created traceability matrix artifact required input from 23 software engineers and took five weeks to complete in addition to a full day spent correcting errors found during a review.

Manual traceability methods also are very vulnerable to changes in the system. If changes occur to any elements captured in the traceability data, the affected portions of the traceability data must be updated manually. This requires discipline and a significant amount of time and effort spent on link checking

throughout the traceability data. Because of this, it is easy for manually created traceability data to become out of sync with the current set of requirements, design, code, and test cases. In the author's own experience, approximately 20% of the entries in a manually created traceability artifact were found to be at least partially out-of-date when subjected to review six months after its initial creation.

Manual traceability methods are also prone to errors which are not easy to catch. Errors can arise from simple typographic mistakes, from inadvertently overlooking a portion of the traceability data such as an individual requirement, or from carelessness by the individual capturing the traceability data. Because traceability artifacts for large projects are often hundreds or even thousands of pages in length, such errors are difficult to detect when depending on manual methods for error checking. In the author's own experience, over 200 requirements were found to be missing from a supposedly up-to-date manually created traceability artifact when a new traceability artifact was generated using an automated traceability method.

Because of these disadvantages, manual traceability methods are not suitable for anything other than small software projects. Young (2006) stated "in my judgment, an automated requirements tool is required for any project except tiny ones." Similarly, Ramesh (1998) found that traceability is error-prone, time-consuming, and impossible to maintain without the use of automated tools. Therefore, why would nearly 50% of aviation software companies use manual traceability methods? Is it because they are all developing tiny projects? In the somewhat humorous words of one DER the author has worked with, "There are no

small aviation software projects." In 1994, Gotel and Finkelstein found that manual traceability methods were preferred in industry due to shortcomings in available traceability tools. It is apparent that this problem still exists today because manual traceability methods are still preferred by a significant percentage of aviation software organizations.

3.2.2 Telelogic's DOORS

Telelogic's DOORS provides a moderately popular alternative to manual traceability methods in the aviation software industry. DOORS is a requirements management system sold by Telelogic that claims to provide full support for traceability. The author was able to obtain a fully-functional trial version of DOORS 8.1 to test with the aviation software project mentioned in the previous section. The findings of that test are discussed here.

The user interface for displaying requirements in DOORS is similar in appearance to that of a word processor. This makes DOORS ideal for storing documentation elements such as requirements, design, and verification data. Creation of new requirements within the DOORS system is a relatively straightforward task. It is also possible to import existing requirements and other artifacts into the DOORS system. The importation process can be customized to a degree using the proprietary DXL scripting language that is supported by DOORS. Traceability links between elements stored in DOORS are created manually. Link creation is a reasonably simple, albeit tedious, task.

Traceability information in DOORS is more resistant to project changes than manually created traceability data. If an element in a chain of traceability links changes, the links to that item will be highlighted as suspect by DOORS. Such links will remain suspect until a user manually updates them or confirms that they are still valid. However, there is no mechanism to force users to update or confirm suspect links to prevent them from appearing in generated traceability matrices. Cleland-Huang et al. (2003) found this to be a general problem with currently available traceability tools.

Errors in traceability data are also less likely in DOORS. Since all of the project requirements are stored within DOORS, it is not possible for these elements to be inadvertently missed when traceability data is created. Instead, if a requirement is missing traceability information, it will appear in the generated traceability data without any links. In theory, these untraced requirements could slip by, but it is likely that they would be caught in a review.

Unfortunately, DOORS is far from ideal as a traceability solution. A major concern with DOORS is its cost. Upon inquiry to Telelogic, the author was quoted a price of $4,000.00 per license plus a 20% yearly maintenance fee. Compare this price to the $299.99 currently charged by Microsoft for the non-upgrade business version of the Windows Vista operating system (Windows Marketplace 2008). Obviously, licensing DOORS gets prohibitively expensive very quickly.

Cost is not the only concern with DOORS. Converting to DOORS from using manual traceability methods is a daunting task. Although DOORS supports

importing requirements from existing documents, there are problems with this feature. DOORS requires everything it stores to be tagged with a unique requirement ID. This means that items such as document section headings, notes, and other non-requirement data all gets treated like a requirement when it is imported into DOORS. It is possible to filter these noise items out of traceability data generated by DOORS, but to do so requires a lot of manual effort to identify them and to let DOORS know that they are not requirements. Even after this is done, the fact that everything must be tagged with a unique requirements ID can make it difficult to determine what is actually a requirement.

Importing existing requirements into DOORS also virtually guarantees that significant rework on the requirements and traceability information will be required. Because DOORS uses its own requirements tagging method, all requirements imported into DOORS automatically are given a unique ID by DOORS. This means that previous methods of identifying requirements immediately become obsolete. It is also necessary to recreate any existing traceability data by manually creating links inside of DOORS.

It is possible to reduce the amount of manual work required when converting to DOORS through use of DXL scripts within DOORS. However, any automation would require a working knowledge of the proprietary DXL scripting language which would require that time be spent learning it. Even with DXL, it is not possible to automate everything. Therefore, a conversion to using DOORS would almost

certainly require at least one individual with a solid knowledge of the system to be dedicated to the DOORS conversion process.

Because of these factors, DOORS is much more appealing when a system is built from the ground up using DOORS. Even then DOORS has shortcomings. A major limitation of DOORS is its ability to only store and interact with document-style artifacts. This is fine for items such as requirements, design, and test cases, but what about source code? The author was unable to find a feasible way to integrate source code into the DOORS system. Because of this, traceability data generated by DOORS lacked source code information. This appears to be an intentional design decision by Telelogic because even on the DOORS website (Telelogic 2007), example DOORS traceability data lacks source code information as illustrated in Figure 3-3.

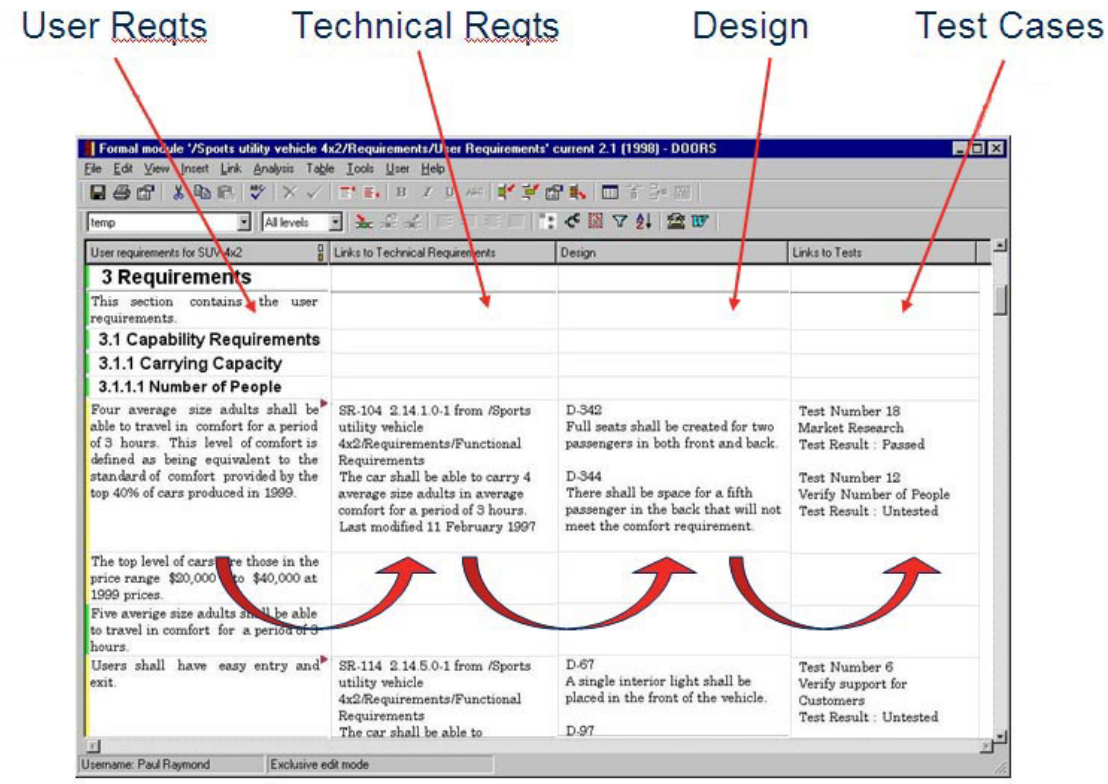

Figure 3-3. Example Traceability Data Generated by DOORS (Telelogic 2007).

The failure of DOORS to include source code in its generated traceability data means that automatically generated traceability artifacts must be manually updated to include source code information in order to meet governmental requirements for aviation software projects. This limitation greatly reduces the utility of having automatically generated traceability information. Unfortunately, this is a common problem among commercial traceability tools because they tend to be decoupled from the development environment (Naslavsky et al. 2005).

DOORS also has some technical limitations. DOORS was initially developed in the early 1990s, and its age shows throughout the user interface. Certain common

user input methods such as using a mouse wheel for scrolling are not supported. Many activities are not intuitive and require several more steps than should be necessary. Even after completing all of the DOORS tutorials, the author still had to consult the DOORS help system in order to determine how to perform many simple activities which could easily have been made more intuitive. This makes it clear that significant training would be required for employees to utilize DOORS effectively. The author also experienced occasional program crashes while creating traceability links within DOORS.

In spite of DOORS' problems, it is likely that using DOORS for traceability would save time and effort compared to using manual traceability methods. The question is, does it save enough time and effort to be worth the high cost? The answer has to be determined by organizations individually, which probably explains why there is nearly an even split between aviation software companies that use DOORS and those that use manual traceability methods.

3.2.3 Other Methods

A small number of aviation software companies use methods other than DOORS or manual methods to create traceability data. A few companies use IBM's Requisite Pro and even fewer use other commercial tools. Because the number of companies using DOORS vastly outnumbers the companies using other commercial traceability tools, it is reasonable to assume that DOORS is best suited for use in the aviation

software industry. For this reason, in-depth testing of other commercial traceability tools was not performed.

It is likely that the reason for the small market penetration of Requisite Pro in the aviation software industry is due to its focus on object-oriented software development (IBM Software 2007), which has been historically shunned in the aviation industry (FAA 2001). The main reason for this is because it is difficult to meet the demands of DO-178B using an object-oriented software architecture.

It is also interesting to note that several companies chose to develop their own proprietary traceability tools. Ramesh and Jarke (2001) discovered that the development of in-house traceability tools was typically initiated because users were dissatisfied with currently available tools. This reinforces the premise that quality traceability tools adequate for the needs of the aviation software industry are not available. Clearly, there is a need for traceability tools that improve upon the foundation laid down by DOORS.

Chapter 4

A Solution for the Traceability Tool Problem

The lack of quality tools for implementing traceability is not an insurmountable problem. The solution is simply the creation of traceability tools usable for software projects that do not share the limitations of currently available tools and that are available for a reasonable cost. To accomplish this, a proposal for a new traceability tool that improves upon the capabilities provided by existing traceability tools is presented in this chapter.

4.1 A Proposal for a Database-Based Approach to Traceability

The DOORS approach to implementing traceability has a lot of merit, but its failure to integrate source code and its high cost are significant drawbacks. The plan for a database-based approach for a traceability tool came from the idea of creating a traceability tool that builds upon the features provided by DOORS without including its limitations.

The main idea behind the database-based approach to traceability is to use a database to store all traceability information and to include a mechanism supporting the generation of a complete traceability artifact. Identifiers for each traceability

element would need to be stored within the database, but the elements themselves could be maintained outside of the database to reduce the impact on existing project artifacts.

Identifiers for requirements and other project artifacts would need to be imported into the database for it to be used with an already existing project. This would require special code to be written in order to parse the existing requirements documents and other project artifacts. After the initial set of records containing identifiers in the database was created, it could be kept up-to-date by regular usage of the importation features of the tool. Depending on the needs of the project, this process could occur automatically at periodic intervals or it could require human intervention to trigger the updates.

Traceability would be maintained through the use of link fields for each requirement record. These fields would specify other requirements, design, source code modules, and test cases that each requirement traces to. Filling out the link fields would be where the human interaction in this traceability method would take place. Depending on the format of the project, portions of the link creation could be automated. For example, test cases typically identify the requirements and source code that they test. The database tool could parse this information from test cases and use it to create links between the test case and the requirements and source code identified in the test case. The database would then be capable of generating a complete traceability artifact based on the stored identifiers and the link fields for each element.

It is expected that the database would make use of referential integrity to ensure that all links stored within the database are valid. If a requirement or other data element is deleted, the database would be able to detect and flag any traceability links that become invalid. Flagged links would need to be corrected to satisfy the constraints of referential integrity, thereby ensuring that any invalid links are corrected before the traceability artifact can be generated. Similarly, the database would be able to detect and prevent any attempts to create links between invalid project elements using referential integrity.

The main goal behind the database-based approach for a traceability tool is to create a traceability method that adds to the traceability feature set of a tool like DOORS at a fraction of the cost. The tool would run on a common database platform such as Microsoft Access or MySQL. The cost of these platforms is considerably less than the cost of a tool such as DOORS. Although this proposed tool would not include as many requirements management features as DOORS, its focus on traceability would help ensure that it is a better traceability tool for the price.

4.2 Prototyping the Database-Based Approach to Traceability

Developing a prototype for the proposed database-based traceability approach required three main activities: identifying the necessary traceability data, designing the database, and creating a software wrapper around the database to provide the user interface, automation, and error-checking capabilities.

4.2.1 Identifying the Necessary Traceability Data

The first step towards creating an improved traceability tool was to identify the data that needed to be traced. Because the database-based traceability tool was expected to be used for a project in the aviation software industry, it needed to be able to meet the governmental traceability mandates for aviation software projects specified by DO-178B (RTCA 1992). These mandates include the following:

- Traceability between system requirements and software design data
- Traceability between system requirements and software requirements
- Traceability between software requirements and source code
- Traceability between software requirements and test cases
- Traceability between source code and test cases

To fulfill these requirements, the traceability tool needed to track links for all of the mandated traceability data.

4.2.2 Designing the Database

The next step was to design a database capable of storing traceability information for the identified traceability elements. The database design began with an entity-relationship diagram relating the entities that needed to be traced to fulfill the FAA traceability mandates. For other applications, the database design could easily be adapted to include other traceability information by customizing the elements

included in the entity-relationship diagram. The resulting entity-relationship diagram is shown in Figure 4-1.

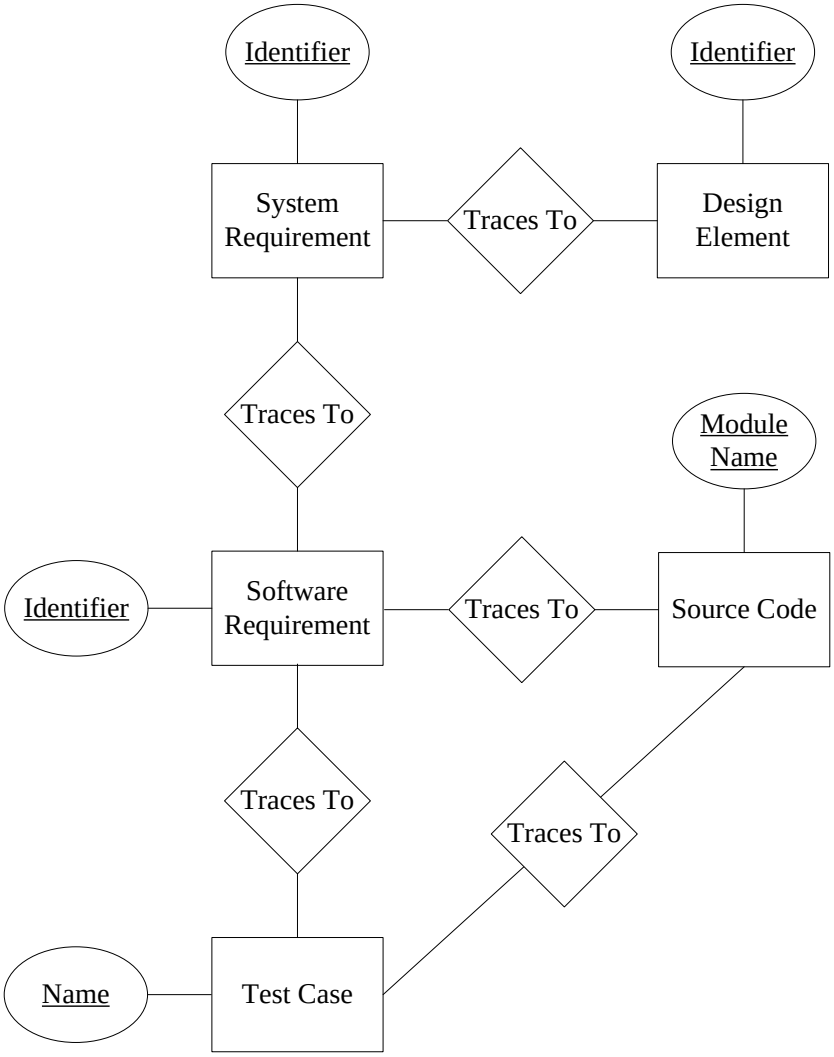

Figure 4-1. Entity-Relationship Diagram for the Database.

The entity-relationship diagram for the database led to the initial simplistic idea for a database relation shown in Figure 4-2.

TRACE

System Requirement ID	Design Element	Software Requirement ID	Source Code Module	Test Case

Figure 4-2. Simplistic Idea for Database Relation.

If the database was implemented using the relation shown in Figure 4-2, it would include a lot of redundant information. Redundancy would be a problem because multiple design elements can trace to a single system requirement, multiple software requirements can trace to a single system requirement, multiple source code modules can trace to a single software requirement, multiple test cases can trace to a single software requirement, and multiple test cases can trace to a single source code module. Using the relation shown in Figure 4-2 would result in many tuples being required to catalogue the traceability data for a single element. Not only would this database design be wasteful in terms of space, but it also would not be able to make use of referential integrity to perform integrity checking on the data.

The initial simplistic database design was normalized into Boyce-Codd Normal Form (BCNF) to address the problems with the initial design. BCNF provides protection from redundancy and logical anomalies as well as providing the opportunity to utilize referential integrity for data integrity checking. The normalized database design is shown in Figure 4-3.

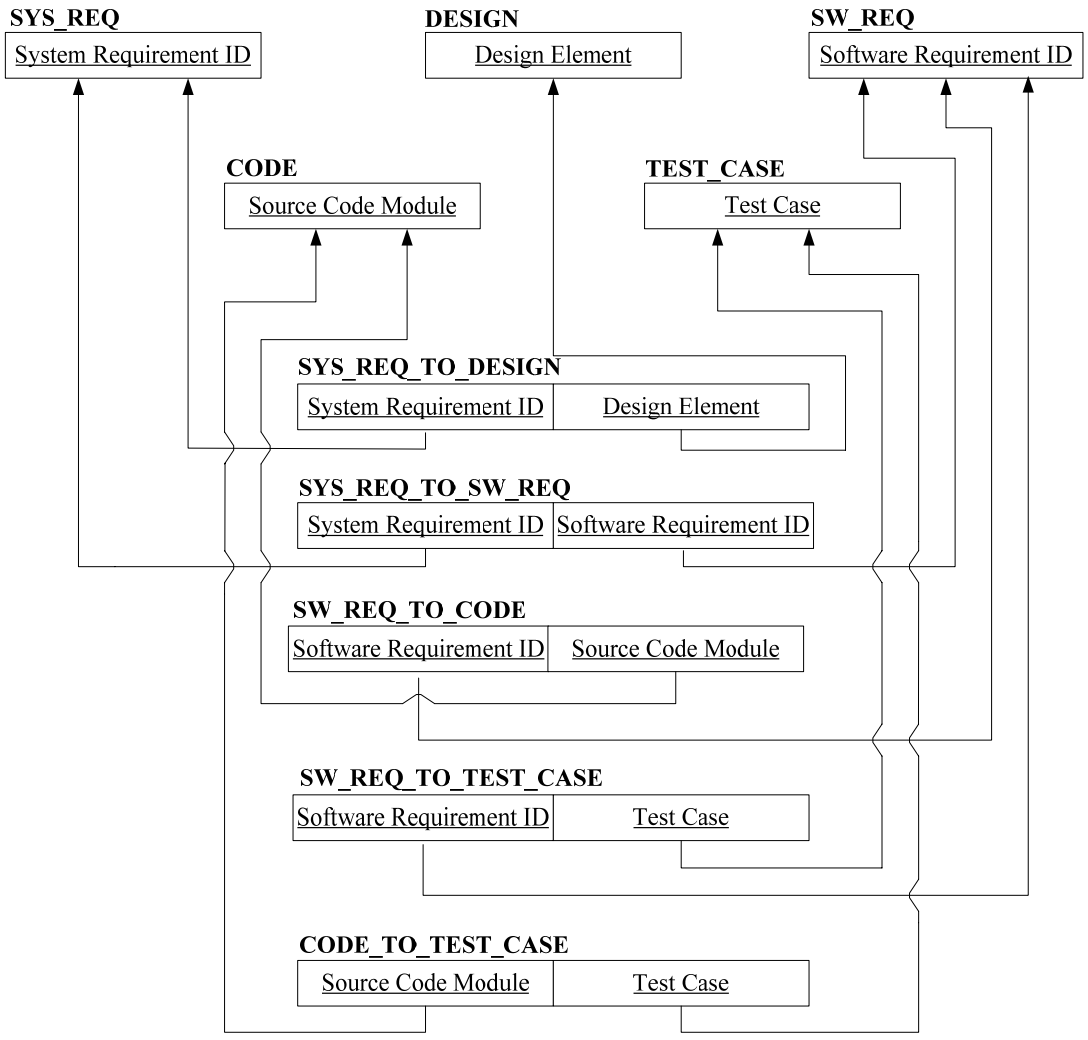

Figure 4-3. Normalized Database Design.

4.2.3 Creating the Software Wrapper for the Database

After the database design was complete, software mechanisms for automatically populating the database relations needed to be written. Custom code was written to automatically populate the system requirements, design data, software requirements, source code, and test cases relations in the database. This was a straight-forward task

involving writing code to parse the requirements and design documents for requirements and design identifiers and to store them in the appropriate relations in the database. This aspect of the tool was made extensible by allowing the format of the requirements and design identifiers to be configurable using regular expressions. For the source code and test cases, the importation software was set-up to simply read the directories where all of the source code and test cases for the project were stored and to enter the name of each source code module and test case into the database.

The next task was to allow for importation of existing traceability links to populate the traceability link relations in the database. Code was written to parse an existing traceability artifact to automatically populate the link relations for all existing traceability information. This was made extensible to a degree by allowing the format of the traceability artifact to be configurable. However, the traceability artifact is expected to be in the format of a traceability matrix because it would be difficult to import traceability data stored using any other method. Although this could be viewed as a limitation, it is unlikely to be a major issue because most manually created traceability data is in the form of a traceability matrix, and most existing traceability tools support the generation of traceability data in the form of a traceability matrix.

After the initial importation of existing traceability links, new links would need to be recorded by entering them into the database. This is where the human interaction in the traceability process occurs. Requiring human interaction to create traceability links is a reasonable decision because it is impossible to completely

remove human interaction from the traceability process (Hayes & Dekhtyar 2005) and because the reason for adding a traceability element is nearly always known by the person adding that element.

Test cases are an exception to this procedure because test cases already typically identify the requirements and source code that they test. Therefore, traceability links involving test cases can be automatically populated by code written to parse each test case for the requirement identifiers and source code modules that they identify. This leaves only the traces between requirements, design elements, and source code as items requiring human interaction.

Validity of the traceability links is enforced through referential integrity. Only links between valid elements are allowed because the use of referential integrity disallows the ability to create links to non-existent items. Each attribute in the link relations in the database is a foreign key that references the key attribute in the relation maintaining data for that particular traceability element. This reduces the potential for human error through typographical mistakes.

To make the database tool more robust, it was desired to include functionality to detect any missing traceability data. One way to do this would be to make the single attribute relations in the database foreign keys referencing the corresponding attribute(s) in the link relations. The downside to this approach is that every time new data is imported into the database, all of the traces for the new data would need to be entered at the same time to satisfy the referential integrity constraints. This is not necessarily desirable as it may be the case that different people are responsible for

importing the data and entering the traceability information. Therefore, instead of making the single attribute relations foreign keys, a reporting feature was included to detect and report any missing traceability information to the user.

It is unlikely that many users would utilize the database tool if it required them to interact with the database directly using queries, so it was important to develop a user-friendly front-end for the database. Therefore, a custom menu was created to appear when the database tool is started. Buttons on the menu make traceability tasks as simple as possible. There are buttons to update the system requirements, design data, software requirements, source code, and test cases relations in the database. There are also buttons to automate the test case traces and to manually enter traces between requirements, design, and source code elements. A button for detecting missing traceability links is also included as well as a button for generating a complete traceability artifact in a traditional traceability matrix format. Initially a button was available that provided access to a database view presenting a complete picture of the traceability data for the project. However, after performing a case study which involved testing the tool with an actual software project (described in Chapter 5), this view was replaced with the ability to generate a standalone traceability artifact. This makes viewing the traceability data easy for those who are unwilling to analyze the data using the tool's interface. The custom user interface created for the database tool is shown in Figure 4-4.

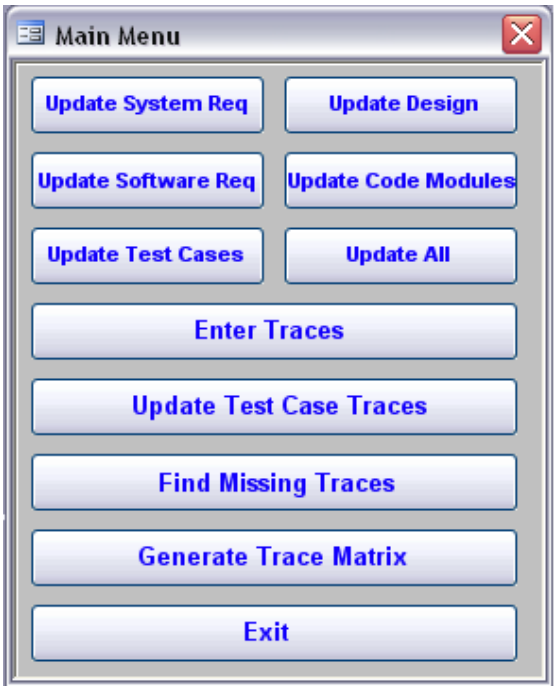

Figure 4-4. Database Tool User Interface.

When the user selects the "Enter Traces" option, they are presented with an interface screen that allows the user to create links between requirements, design elements, and source code modules. First, the user selects the requirement identifier of the requirement to create a traceability link for. This can be done by typing the requirement identifier into a text box, or the requirement can be selected using a drop-down list that is populated with all of the requirement identifiers stored in the database. If the user manually types a requirement identifier, the tool will ensure that the requirement identifier exists in the database. If it does not exist, the user will not be allowed to create a link. This eliminates the possibility of the user creating a link to a non-existent requirement.

Once the requirement identifier to create a traceability link for is selected, the user can create a link to another requirement, a design element, or a source code module by either typing the identifier of the requirement, design element, or source code module into a text box or by selecting the requirement identifier, design element, or source code module using drop-down lists for each element type that are populated with all of the requirement identifiers, design elements, and source code modules that are stored in the database. If the user manually enters the identifier of a requirement, design element, or source code module, the tool will ensure that the item exists in the database. If it does not exist, the user will not be allowed to create a link to eliminate the possibility of the user creating a link to a non-existent item. Traceability links are not saved until the user presses the Save button to give the user the opportunity to verify that the each entered traceability link is correct.

Since the custom front-end user interface for the tool abstracts the database from the user, it would be possible to use any database to store the traceability data. The prototype of the database tool was implemented and tested with two different databases: Microsoft Access and MySQL.

Both databases prototyped with the tool offer their own advantages and disadvantages. The Microsoft Access version of the tool uses a database that is commonly available with other office software products that requires no maintenance or special expertise to keep up. All database information is stored in a single database file that could easily be used with a configuration management system for version

control. The downside to using a Microsoft Access database is that only one user can modify the database at a time since it is contained in a single file.

The MySQL database offers an advantage in that MySQL is freely available and does not require licenses to use. In addition, MySQL provides support for multiple users to modify the database at the same time. The disadvantages of using MySQL are that MySQL is not a commonly-known tool to the average office worker; therefore, using MySQL potentially introduces the need for a database administrator to be responsible for database maintenance and backup functionality. Another disadvantage of MySQL is that it is more difficult to implement version control since a MySQL database cannot easily be stored within a configuration management system.

Chapter 5

A Practical Case Study

This chapter describes a case study performed using the database-based approach to implementing traceability that was presented in Chapter 4. Details about the process of using the prototype of the database-based tool on an actual software project in the aviation software industry are presented. Metrics and qualitative results from this case study are presented in Chapter 6.

5.1 Software Project Background

The software project used for the case study described in this chapter is an iterative, incremental project where versioned builds of the software are delivered periodically. Each succeeding build of the software is based upon the previous build, but it adds significant new functionality. All of the waterfall model software lifecycle activities are repeated for each build. The project is used in the aviation industry and is therefore subject to the FAA governmental mandates specified by DO-178B. The initial build of the project took place in 2002, and the project has continued to grow in size and complexity since that date. Today, the development team for the project

includes 45 software engineers, and the project easily meets Bennatan's (2006) definition for a large software project.

5.1.1 Initial Traceability Implementation

Little thought was given to traceability prior to the completion of the first build of the software project. Only after the realization occurred that a traceability artifact was necessary for the software to obtain approval from the FAA did traceability activities begin. Unfortunately, the lack of planning for traceability meant that it was difficult to implement, making it a time-consuming activity that provided little benefit to the project apart from meeting governmental mandates.

Traceability information was recorded in a traceability matrix contained in a single spreadsheet shared among the software engineers working on the project. This was not a very efficient mechanism because all of the traceability data was gathered manually, and it needed to be entered into the spreadsheet manually by each software engineer. Having multiple engineers work in parallel was a challenge because only one person could enter data into the spreadsheet at a time. Multiple individuals could work in parallel using a temporary copy of the spreadsheet on their own computer, but there was no foolproof method to ensure that work was not duplicated, and merging each person's changes into the spreadsheet was a time-consuming and potentially error-prone process.

The information recorded in the spreadsheet traced system requirements to software design data, system requirements to software requirements, software

requirements to source code, software requirements to test cases, and source code to test cases in order to meet the traceability mandates in DO-178B. The source of this information was simply special knowledge either recollected or researched by specific engineers working on the traceability artifact since most of the information had not been previously documented. This meant that finding traceability data for items that none of the engineers had a clear recollection of was difficult and time-consuming.

Overall, the creation of the traceability artifact required input from 23 software engineers and took five weeks to create. When the initial version of the traceability matrix was subjected to a review, it took another full day to correct all of the problems found during that review. In the end, the lack of forethought regarding traceability meant that the initial delivery of the software was delayed by nearly six weeks after the software build itself was complete.

The significant delays introduced by the creation of the traceability artifact after the completion of the first software build made it obvious that better methods were necessary for implementing traceability in the future. However, since the project already had a foundation in place, it was desired that any changes to the traceability method have little impact on the already existing project artifacts. Since all of the project's documentation, including requirements, design data, and test cases, was based around Microsoft Word documents, the possibility of converting the project to using alternative methods of documentation, such as a database or a commercial tool such as Telelogic's DOORS was ruled out.

5.2 Database-Based Traceability Tool Case Study

5.2.1 Preparation for Use of the Database Tool

Before the database-based traceability tool could be used for the project, a decision had to be made regarding whether to use Microsoft Access or MySQL as the database back-end for the tool. Meetings were held with the project stakeholders, and the strengths and weaknesses of each database were discussed. A prototype of both versions of the tool was provided to the stakeholders to assist with the decision-making process.

The version of the tool based on Microsoft Access was chosen for use with the project for two reasons. First of all, the engineers working on the project already had Microsoft Access installed on their computers as part of the standard Microsoft Office suite used by the team. Secondly, there were concerns about demonstrating configuration management if a MySQL database was used since the database itself could not easily be stored within the configuration management system used for the project. This was not a problem for the Microsoft Access database, as the Access database file could easily be stored within the existing configuration management system. The only downside to using Microsoft Access was that it would only allow one engineer to use the tool at a time since the database would have to be checked out from the configuration management system and later checked back in when the modifications were complete. This was not considered to be a major issue for the project because other project artifacts had the same limitation.

The next step was to configure a few parameters for the software project to customize the tool for the project. This included specifying the format of the requirement and design identifiers so the tool could identify them as well as pointing the tool to the requirements and design documents and the directories containing the project's source code, test cases, and previously existing traceability artifact to allow for importation of data. The format of the existing traceability matrix artifact was also configured to allow for importation of the previously captured traceability links for the project. Once this information was configured, it was easy to import the necessary traceability data for the project using the buttons included in the tool's user interface.

5.2.2 Using the Database Tool

After the database tool was configured for the project, the first challenge was getting the engineers working on the project to use the tool. In spite of the time spent on the user interface attempting to make it as easy to use as possible, many of the team members on the project were reluctant to start using the tool initially. This seemed to be a psychological barrier due to the fact that few of the members of the team were comfortable working directly with a database.

Training was scheduled for all of the team members to demonstrate how to use the tool and to present the perceived benefits offered by the tool, namely, reduced human interaction in the traceability process resulting in time savings and fewer

errors in the traceability data output. After going through training, the engineers seemed more receptive to using the tool.

After the database tool was presented to the team, it was introduced as a replacement to the manually created traceability matrix which had been used to document traceability information for the project in the past. The conversion occurred right after the release of a build of the software so that it would not cause a disruption right in the middle of a software release. From that point on, the engineers working on the project used the database tool to record traceability information for the project. Instead of waiting until the end of the software release to document traceability links, use of the tool to capture traceability links for project elements when they were created was added to the process of adding new elements to the project. Because the FAA requirements for aviation software mandated by DO-178B necessitate reviews for all project elements, this was easily accomplished by including checks for appropriate traceability in the review forms for each project element.

At this point, the database-based traceability tool was fully integrated into the process of capturing traceability information for the project. However, before the tool could be used to demonstrate compliance with the FAA requirements for traceability for aviation software projects, it needed to be reviewed and accepted by the project's designated engineering representatives (DERs).

5.2.3 Reviewing the New Traceability Method

The next step was to get the new traceability method reviewed and approved by the project's DERs. DERs perform review work on aviation projects to ensure that they are in compliance with FAA standards such as DO-178B. Before the new traceability method could be used to take credit for compliance with governmental traceability regulations, it had to be approved by the DERs.

At first, DER acceptance was a major roadblock. Even though the initial version of the database tool included a view that provided a complete picture of the project's traceability data, the DERs refused to accept the traceability view within the database as proof of compliance with the required traceability mandates. They were unwilling to look at data within the tool's user interface, citing their general unfamiliarity with databases and calling it a non-standard way to demonstrate traceability compliance.

The DERs' response forced part of the database tool back onto the drawing board. If it could not be used to demonstrate compliance with traceability mandates, the tool would lose much of its value for the project. This led to the idea of having the database tool output traceability data in the form of a traceability matrix because the DERs had accepted manually created traceability data in the form of a traceability matrix in the past. An example of the format of the output traceability matrix from the database-based tool is shown in Table 5-1.

Table 5-1. Example Traceability Matrix Output from the Database-Based Tool.

System Requirement	Design Element	Software Requirement	Code Module	Test Case
005-00150-80#00505	005-00150-60#01225	005-00150-85#00112	IOP_air_data_intf.c	tc_103.doc
005-00150-80#00506	005-00150-60#00562	005-00150-85#00234	cdp_fld_airspeed.c	tc_125.doc

When traceability information was output from the database-based traceability tool and presented to the DERs in the form of a standalone traceability matrix, they had no problem with the results. In fact, for the first time in the history of the project, the DERs did not have any non-compliance comments about the traceability data. Instead, they focused their comments on the format of the output data, demanding that the output be presented in nicely formatted tables. This meant that a large amount of time had to be spent on custom code for outputting the traceability data to ensure that the output was presented well. This was a tedious task, but once it was accomplished, the DERs accepted the results. At that point, the database tool received the DERs' stamp of approval for use for capturing and reporting traceability information for the project.

5.3 Current State of the Database-Based Traceability Tool

All subsequent releases of the software for the project that was used for the case study have continued to utilize the database tool for traceability activities because the tool was deemed to be a major success. In addition, numerous other projects within the company have expressed interest in the tool, and several additional projects have

begun the process of converting to use the prototyped traceability tool based on the success of the initial case study. Detailed quantitative metrics from the case study and qualitative evaluation criteria for the database tool are presented in Chapter 6.

Chapter 6

Evaluation and Analysis

This chapter evaluates the results of the case study performed using the prototyped database-based traceability tool in order to determine if the proposed approach for implementing traceability is a viable alternative to existing methods. Quantitative metrics from the case study are presented, and a cost comparison with alternative traceability methods is provided. A qualitative analysis of the strengths and weaknesses of the database-based tool for implementing traceability is also performed.

6.1 Quantitative Metrics

6.1.1 Comparison with Past Project Results Using Manual Methods

This section quantitatively compares the results of using the database-based traceability tool with the manual traceability methods used on the project in the past. Bar graphs are used to detail the number of man-hours required for activities such as preparation for use, time spent while working on a software release, and time spent at the end of a software release for each method. These results are reasonable to

compare because, for each method, the results were collected using software releases that added similar amounts of functionality to the system. The number of errors found after the initial release of the traceability data for each method is also compared.

Figure 6-1 shows the amount of development and preparation time required to be able to use each traceability method. Figure 6-2 shows the amount of time spent on traceability activities while working on a software release, and Figure 6-3 shows the amount of time spent preparing the traceability artifact and going through the review process at the end of a software release. Figure 6-4 shows the number of errors that were later detected in the traceability artifact after it had been released.

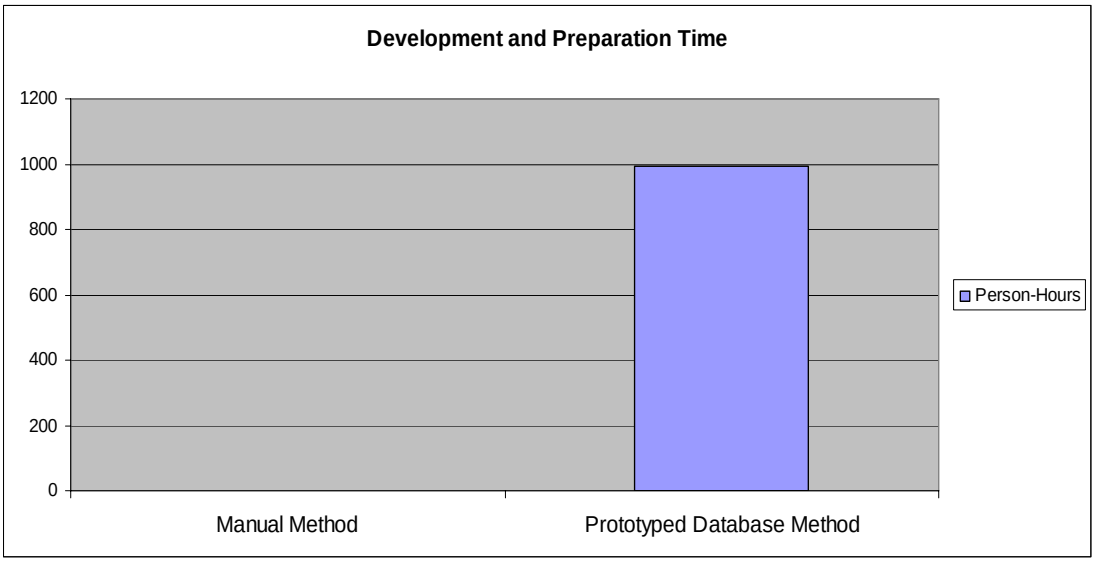

Figure 6-1. Development Time Required for Traceability Methods.

As shown in Figure 6-1, the database method of implementing traceability required significantly more development and preparation time than the manual

method. This is because the database tool required a significant amount of complex custom code to be written for the automation, error-checking, and data output capabilities.

By comparison, manual traceability methods require very little preparation time. The creation of a spreadsheet or a word processor document with tables to record the data is sufficient. However, the extra development time required for the automated database traceability method pays off later through improved quality of the results (see Figure 6-4) and time saved later on in the process (see Figure 6-3). Because the development time is a one-time cost, it can be viewed as an up-front sacrifice resulting in faster, higher quality results later. In addition, if the tool was used for other projects, the development time would not need to be repeated for each project; thereby making it a start-up cost only.

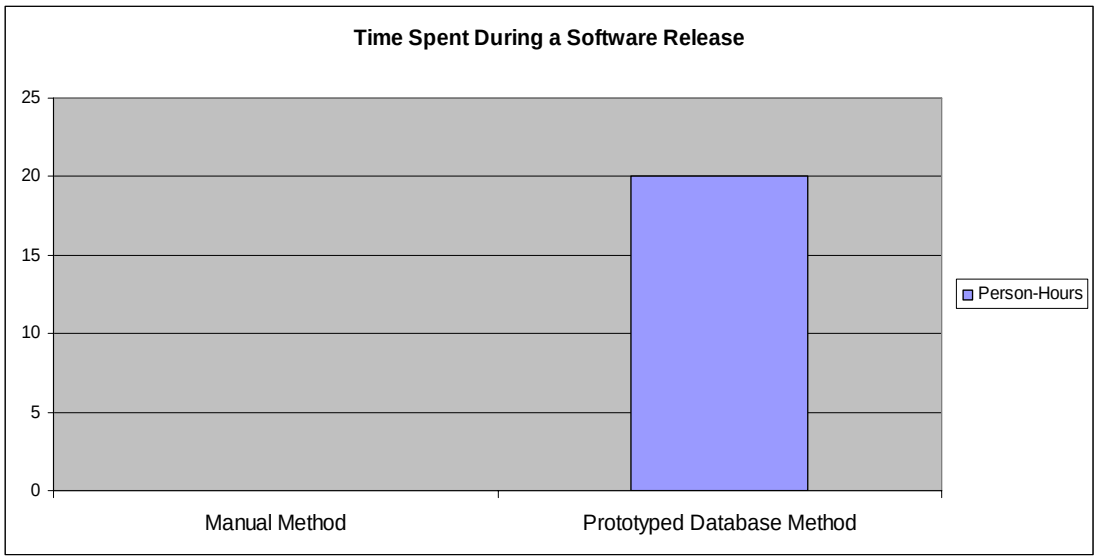

Figure 6-2. Time Spent on Traceability Activities During a Software Release.

As shown by Figure 6-2, the use of the database traceability method did require more time than manual methods while working on a software release due to the need to create traceability links as elements were added to the project. However, the extra amount of time required for the automated database method was a small price to pay for the time savings later as shown in Figure 6-3 and better quality of the results as shown in Figure 6-4.

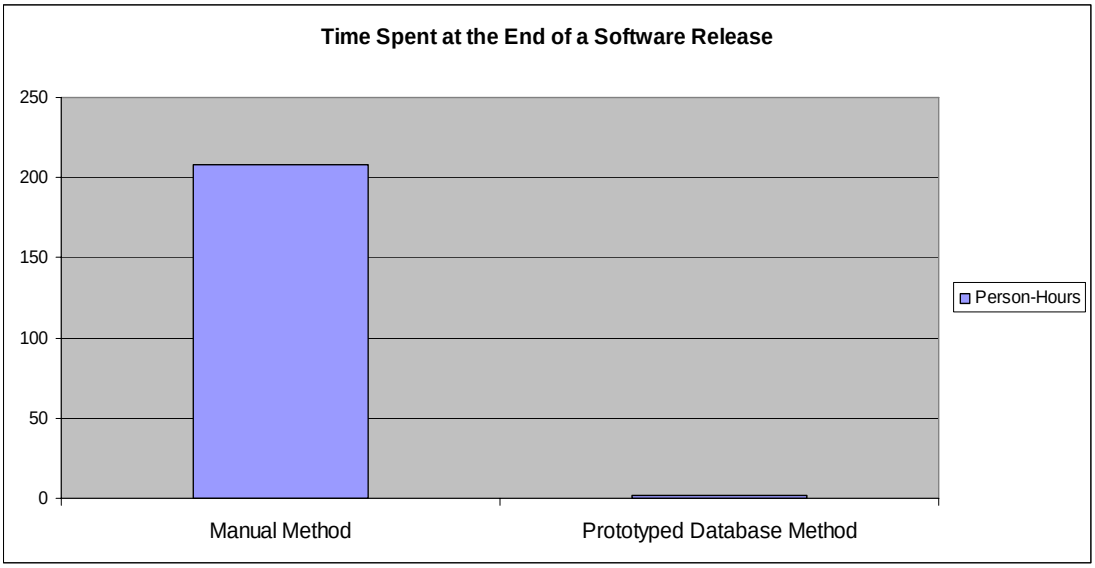

Figure 6-3. Time Spent on Traceability Activities at the End of a Software Release.

Figure 6-3 clearly shows that the payoff for using the automated database traceability method comes at the end of a software release. Although some time is still required to generate the data and have the traceability information reviewed, the total time required is insignificant compared to the amount of time required to gather traceability data manually. In fact, it would be virtually impossible to reduce the amount of time required for traceability activities at the end of a software release

because of the need for reviews. Nearly all of the time required at the end of the software release for the database tool was spent on reviews.

The significant time savings at the end of a software release provided by the database-based tool is important because it meant that the software could be released to market approximately 4.5 weeks sooner than it could in the past when manual traceability methods were used. An earlier time to market results in additional sales which means higher profits are realized. Pinning an exact dollar figure on the impact of releasing the software to market 4.5 weeks sooner is nearly impossible due to differing contractual obligations and other factors which vary per software release. However, the past history of the software project used for the case study described in Chapter 5 shows that approximately five new sales occur in the first 4.5 weeks after each software release for each aircraft program that takes the new software, and on average ten programs take each software release. The translates into approximately 50 extra sales if the software is released 4.5 weeks earlier, which results in a potential increase of $4,000,000.00 in gross profits at an average sales price of $80,000.00.

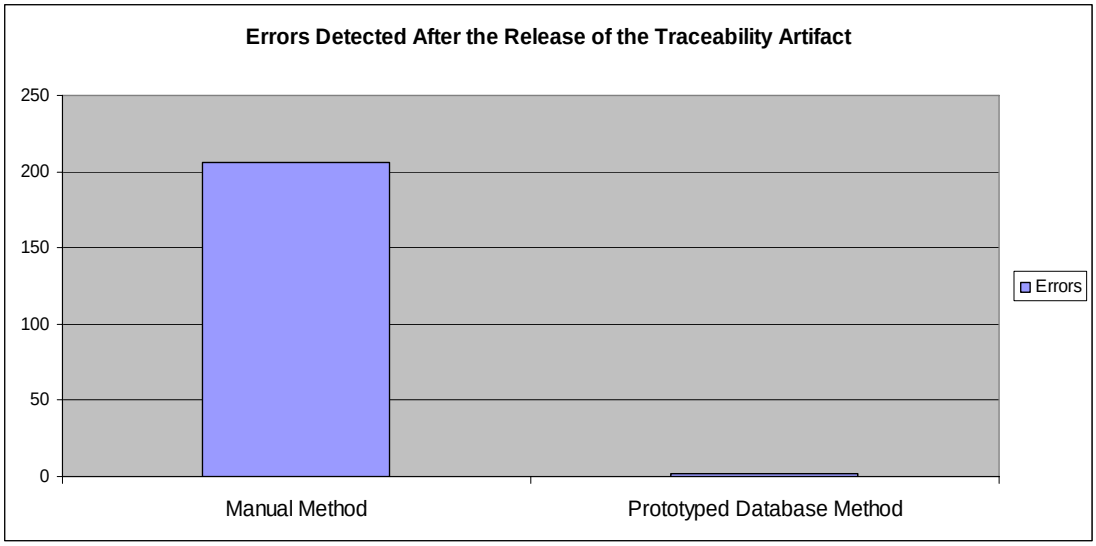

Figure 6-4. Number of Errors Detected in the Traceability Data.

As shown in Figure 6-4, using the database method of implementing traceability greatly reduced the number of errors that were later detected in the released traceability artifact. Due to the robust error-checking features built into the database tool, only two errors were found after the release of the traceability data generated by the tool. These errors were human errors where incorrect links between requirements were manually entered into the database. The reason that so many errors were detected in the results from the manual method was because many requirements were overlooked in the manually created traceability matrix due to human error.

Fewer errors in the traceability results is significant because not only does it prevent the possibility of errors propagating later, but it also reduces the potential for errors to be uncovered during an FAA audit. The last time that errors were uncovered during an FAA audit on the project used for the case study performed in Chapter 5

resulted in two extra months of effort on the next software release to correct the errors and to put additional processes in place to prevent similar errors in the future. Those extra two months of effort cost $562,500.00 for staff salaries and resulted in a potential loss of $8,000,000.00 in gross profit on sales.

6.1.2 Cost Comparison with Traceability Alternatives

This section compares the cost of using the database-based traceability tool with the cost of using other traceability alternatives including manual methods and Telelogic's DOORS. Figure 6-5 compares the start-up costs for each traceability method and Figure 6-6 compares the cost of using each method for each software release.

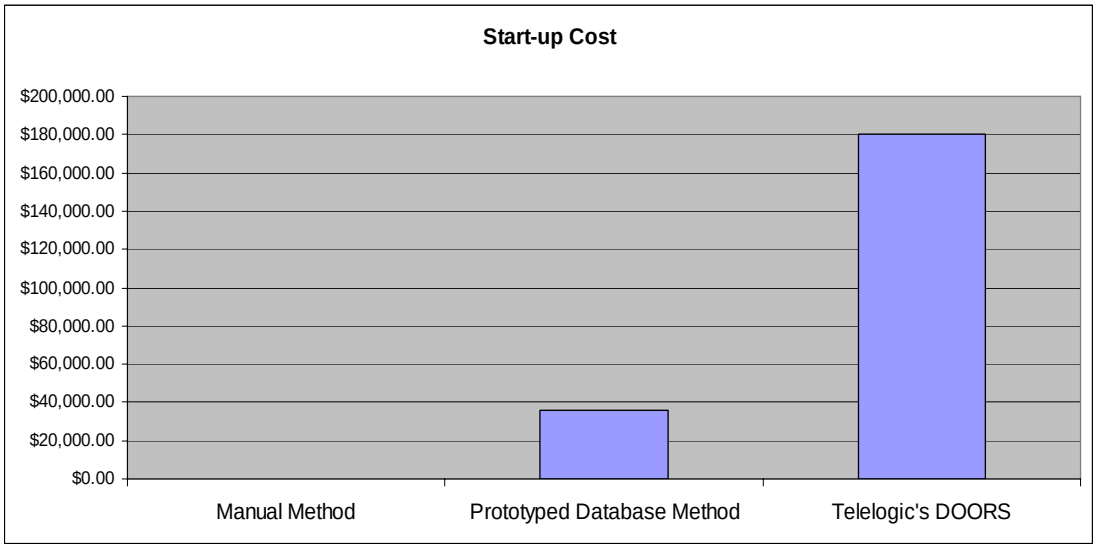

Figure 6-5. Start-up Cost Comparison.

The development and other necessary start-up efforts required for using the prototype for the database-based tool (including DER reviews) required approximately 995 man-hours of effort. Assuming an average salary of $75,000.00,

this translates into a start-up cost of approximately $35,877.40. If Telelogic's DOORS had been selected for use on the project, the licensing cost for the 45 software engineers assigned to the project would have been $180,000.00 in addition to a $36,000.00 yearly maintenance fee. As additional engineers were added to the project, the cost for licenses and maintenance would only increase as additional licenses would need to be purchased for each new person added to the team. Converting to DOORS would also incur a signification start-up cost in addition to the licensing fees because it would require both time and resources to convert the project over to the DOORS system. Manual methods require very little in terms of start-up costs because they can make use of a simple spreadsheet or table in a document. However, manual methods become more costly after a project is started due to the amount of time required to use them. This is shown in Figure 6-6.

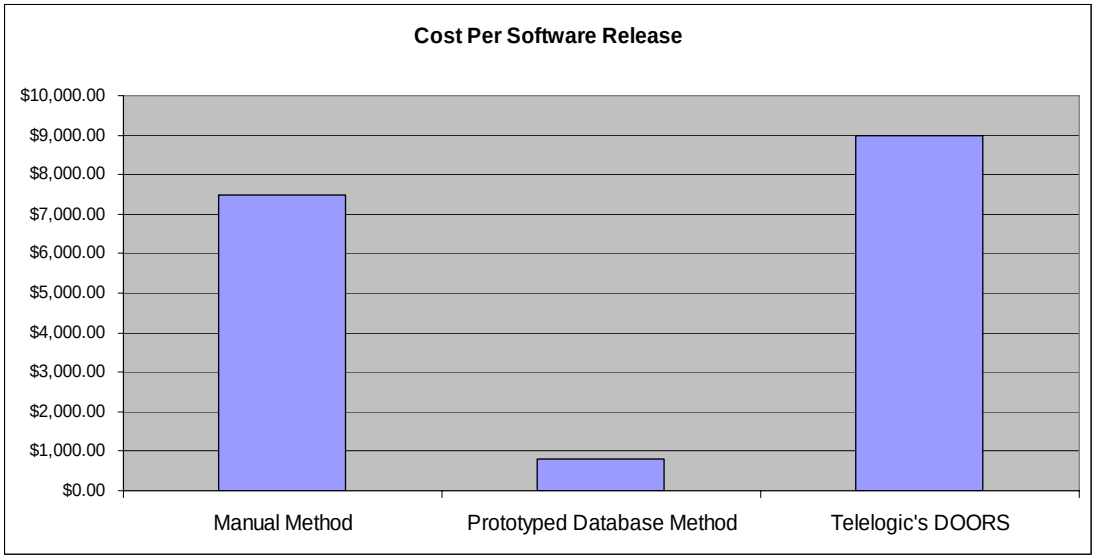

Figure 6-6. Cost Comparison per Software Release.

The high cost of using manual traceability methods is clearly shown in Figure 6-6. Due to the large amount of time and effort required to implement traceability manually for each software release, manual methods incurred a cost of $7,500.00 per software release. In comparison, the prototyped database method only cost $793.27 because most traceability tasks were automated and did not require significant human interaction. The cost estimate of $9,000.00 for Telelogic's DOORS came from dividing the yearly maintenance fee of $36,000.00 by the average number of software releases per year (four) for the software project. In practice, the actual costs would be higher because time would need to be spent on traceability activities within the DOORS system for each software release.

Overall, use of the database-based tool for traceability is favorable in terms of cost in comparison to both Telelogic's DOORS and manual methods. Because implementing traceability using manual methods required 186 additional man-hours of work per software release, this translates into an extra cost of approximately $6,706.73 per software release. At that rate, only six software releases would be required to completely offset the initial development cost of the database-based tool. Because the software project used for the case study described in Chapter 5 averages four software releases per year, the initial cost of development for the database-based tool would be offset in only 1.5 years. In addition, the estimate of the extra cost for using manual methods is a very conservative one, as neither the potential for extra sales resulting from releasing the product to market sooner nor the benefits from the higher quality results provided by the database-based tool were taken into account. If

the tool were used for additional projects, the overall costs would be even lower because the initial development costs could be spread among multiple projects.

Use of the database-based tool is also favorable in terms of cost when compared to using Telelogic's DOORS. The initial costs for developing the database-based tool were $144,122.60 less than licensing Telelogic's DOORS, and the cost per software release was $8,206.73 less because the database tool did not have yearly maintenance fees. This is a conservative estimate as the cost per software release for Telelogic's DOORS does not include the cost of the time that would need to be spent on traceability activities using the DOORS interface because this data was not available for the project for which the case study was performed.

6.2 Qualitative Analysis

6.2.1 Database Tool Strengths

The biggest strengths of the database tool are the amount of automation it introduces to the traceability process and the facilities for preventing and detecting traceability errors that are included. With the database tool, most of the aspects of generating traceability information are automated; human interaction is only required for creating links between requirements, design data, and source code. Everything else can be automated using the buttons included in the user interface.

The error checking facilities for traceability links included in the database tool are a major benefit. The use of referential integrity for the traceability links means that it is impossible to introduce links to non-existent data. Such incorrect links were

a common occurrence with the manual traceability method due to typographical errors. Similarly, the importation tools for the database prevent the possibility of failing to include existing items or having non-existent items in the base relations in the database. Use of the importation tools on a regular basis makes it possible to prevent stale items from being stored in the database as well as automatically adding new elements. In addition, the user interface includes an option for checking for missing traces. This allows for the identification of areas where tracing needs to be completed as well as making it easy to identify the creation of unspecified features which are indicated by design elements or source code that do not trace to requirements. Such features, known as gold-plating or feature creep, are a drain on both time and resources and should be avoided (Muvuti & Lungu 2004).

The only errors that the database tool cannot account for are manually created incorrect links between existing project elements. If checking traceability links is included in the project's review process, it should be difficult for such links to slip through. This is demonstrated by the very low number of errors detected in the traceability data generated from the database tool in the case study. This is not surprising because it is a well-documented fact that as the amount of human interaction in the traceability process is reduced, the number of errors in the resulting traceability data is also reduced (Hayes & Dekhtyar 2005).

The database tool also had a low impact on existing project artifacts, as it was capable of importing data from them without requiring changes. This was important because it allowed other project development activities to continue in parallel with the

development of the database-based automated traceability tool. If a major impact on project documentation had been required, a significant amount of delay would have been incurred because the project DERs would have needed to review the changes to the artifacts in addition to the new tool to ensure that they were acceptable. The low impact on existing project artifacts is a strength that is not shared by commercially available traceability tools such as Telelogic's DOORS (2007). Similarly, the ability of the database-based tool to easily integrate source code into the traceability information is a major benefit that is not provided by Telelogic's DOORS (2007).

6.2.2 Potential Areas of Improvement

The database tool for automating traceability does have some room for improvement in certain areas. Usability is one such area. Although much time and effort was spent trying to make the database tool as user friendly as possible, many software engineers who were not experienced with databases were initially reluctant to try it. Similarly, the tool was initially viewed with suspicion by the DERs who performed reviews for the project. They refused to use the tool's interface to view the traceability data and to check for traceability errors. Instead, they demanded that the tool output traceability data in a traditional traceability matrix format which required a lot of time and effort to be spent writing code to allow the database to output nicely formatted traceability matrices for the DERs to review.

One way to improve the usability of the tool would be to add user documentation and to include context-sensitive help features. Since the tool was only

developed as a prototype, time was not spent developing significant user help features as these topics were covered in a training session with the expected users of the tool. However, such features have become more important as additional projects have expressed interest in the tool. Therefore, the development of user documentation and help features is considered to be important future work on the tool.

Human error has the potential to introduce incorrect links in the traceability information when the traceability links between requirements, design data, and source code are created. It is impossible to completely remove human interaction from the traceability process (Hayes & Dekhtyar 2005); however, if the links were made even easier to create, the potential for human error could be reduced. One idea for making the link creation process easier is to include contextual information along with the requirement, design, and source code module identifiers that are stored within the database. This would reduce the potential for human error because it would make the items being linked more apparent without having to refer to external resources such as a requirements document.

The reliance of the tool on an underlying database such as Microsoft Access or MySQL introduces tradeoffs. It is possible that neither database may be completely ideal for a project. For example, use of an Access database may introduce complications for allowing multiple simultaneous users. Use of a MySQL database may complicate configuration management and may require a database administrator. If the tool did not rely on an external database, it is possible that the best features of the currently supported external databases could be combined in a database contained

within the tool itself. However, development of an internal database for the tool was considered to be outside the scope of this research.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

Traceability offers many benefits to software projects, and it has been identified as being critical for their success (Young 2006). Unfortunately, many organizations struggle to understand and implement traceability which means that these benefits can go unrealized. Many methodologies exist for implementing traceability; however, each existing methodology has important weaknesses that hinder the implementation of traceability. Some of these methods require a significant amount of manual work to create and maintain. Commercial tools exist that attempt to automate some aspects of the traceability process, but they are expensive and have their own set of limitations. Methodologies for automating traceability have been proposed in the academic world, but tool support for these methods is lacking in industry. Because of this, quality tool support for traceability activities in the software engineering field has remained an open problem.

For this reason, this thesis has explored a streamlined, cost-effective method of automating traceability activities using a database-based tool. The proposed method was described in detail, prototyped, and tested in a case study using an actual

software project. The experimental results of the case study were presented in Chapter 6, and the results serve to demonstrate the viability of the proposed method for implementing traceability for software projects. Not only did the new method save time in comparison to manual methods of implementing traceability, but the resulting output also contained far fewer errors. The new method also did not share in the usual weaknesses of commercial traceability tools in that it was significantly lower in cost, and it was able to include important traceability information such as source code that is lacking from popular commercial tools such as Telelogic's DOORS (2007). A comparison of pertinent information for the new tool is provided for manual methods in Table 7-1 and for Telelogic's DOORS in Table 7-2.

Table 7-1. Comparison of the Database-Based Tool with Manual Methods.

Method	Start-Up Costs	Cost Per Software Release	Number of Errors Detected in the Results
Manual Method	$36.06	$7,500.00	206
Database Tool	$35,877.40	$793.27	2

Table 7-2. Comparison of the Database-Based Tool with Telelogic's DOORS.

Method	Start-Up Costs	Yearly Maintenance Fees	Source Code Included in Results?
Telelogic's DOORS	$180,000.00*	$36,000.00	No
Database Tool	$35,877.40	$0.00	Yes

*This figure only includes licensing fees, and does not take into account the cost of converting the project over to the DOORS system, which is likely to be significant.

7.2 Summary of Contributions

This thesis proposed a streamlined, cost-effective database-based method for implementing and automating traceability activities for software projects. The proposed method was described, prototyped, and tested in a case study using an actual software project. Metrics were collected from the case study, and the results demonstrated that use of the new traceability approach resulted in time savings as well as fewer errors in the resulting traceability output in comparison with manual methods. The proposed traceability tool was considerably more cost-effective to develop and use than either manual traceability methods or established commercial traceability tools such as Telelogic's DOORS. A qualitative analysis of the new traceability tool was also performed. The strengths and weaknesses of the approach were described and analyzed. The quantitative and qualitative analysis demonstrated that the new approach to traceability provides significant improvements over both manual methods of implementing traceability and existing commercial traceability tools such as Telelogic's DOORS.

7.3 Future Work

This research focused on developing a cost-effective alternative method for implementing and automating traceability activities for software projects. Although the proposed method was prototyped and tested in a case study, effort was not spent on developing a viable commercial product that could easily be deployed throughout the software engineering industry. In the future, it would be beneficial to extend upon

the work presented in this thesis to make the proposed traceability method more easily portable among software projects. This would also facilitate testing of the tool with other software projects.

Additionally, since the proposed traceability tool was only developed as a prototype, potential improvements to the user interface were identified and noted as areas that could be improved in the prototyped version of the tool. It would be useful to spend time refining the user interface to make the tool easier to use, especially in the area of link creation. Similarly, it would be helpful to spend time creating user documentation and context-sensitive online user assistance for the tool to improve its usability.

It would also be beneficial to consider an internal implementation of the database used with the tool instead of relying on an external database such as Microsoft Access or MySQL. Each external database has its own strengths and weaknesses, and it may be possible to realize the strengths of each external database in an internal database without incorporating their weaknesses.

Bibliography

Bennatan, E. (2006), *Catastrophe Disentanglement: Getting Software Projects Back on Track,* Addison Wesley Professional

Boehm, B. (2003). "Value Based Software Engineering", *ACM SIGSOFT Software Engineering Notes,* 28(2).

Bowen, J., O'Grady, P. & Smith L. (1990), "A Constraint Programming Language for Life-Cycle Engineering", *Artificial Intelligence in Engineering,* 5(4), pp. 206-220.

Chrissis, M., Konrad, M. & Shrum, S. (2003), *CMMI : Guidelines for Process Integration and Product Improvement*. Addison-Wesley Professional.

Clarke, S., Harrison, W., Ossher, H. & Tarr, P. (1999), "Subject-Oriented Design: Towards Improved Alignment of Requirements, Design, and Code", *Proceedings of the 1999 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications,* pp. 325-329, Dallas, TX.

Cleland-Huang, J., Chang, C. & Christensen, M. (2003), "Event-Based Traceability for Managing Evolutionary Change", *IEEE Transactions on Software Engineering,* 29(9), pp. 796-810.

Cleland-Huang, J., Chang, C. & Ge, Y. (2002), "Supporting Event Based Traceability through High-Level Recognition of Change Events", *Proceedings of the 26th Annual International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment,* pp. 595-602, Oxford, England.

Compuware Corporation (2004), "Requirements Traceability for Quality Management", Compuware Whitepaper. http://www.softwarebusinessonline.com/images/WhitePaper_Compuware.pdf.

Davis, A. (1990), "The Analysis and Specification of Systems and Software Requirements", *Systems and Software Engineering*, IEEE Computer Society Press, pp. 119-144.

Domges, R. & Pohl, K. (1998), "Adapting Traceability Environments to Project Specific Needs," *Communications of the ACM*, 41(12), pp. 55-62.

Egyed, A. (2001), "A Scenario-Driven Approach to Traceability," *23rd International Conference on Software Engineering*, pp. 123-132, Toronto, Ontario, Canada.

Egyed A. & Grunbacher P. (2002), "Automating Requirements Traceability: Beyond the Record and Replay Paradigm", *Proceedings of the 17th IEEE International Conference on Automated Software Engineering*, pp. 163-171, Edinburgh, United Kingdom.

Egyed A. & Grunbacher P. (2004), "Identifying Requirements Conflicts and Cooperation: How Quality Attributes and Automated Traceability Can Help", *IEEE Software*, 21(6), pp. 50-58.

Evans, M. (1989), *The Software Factory*, John Wiley and Sons.

Federal Aviation Administration (2001), *Commercial Off-The-Shelf (COTS) Avionics Software Study*, Report No. DOT/FAA/AR-01/26.

Finkelstein, A. & Dowell, J. (1996), "A Comedy of Errors: The London Ambulance Service Case Study," *Proceedings of the Eighth International Workshop on Software Specification and Design*, pp. 2-5, Schloss Velen, Germany.

Gills, M. (2005), "Software Testing and Traceability", University of Latvia. http://www3.acadlib.lv/greydoc/Gilla_disertacija/MGills_ang.doc.

Gotel, O. & Finkelstein, A. (1994), "An Analysis of the Requirements Traceability Problem", *Proceedings of the First International Conference on Requirements Engineering*, pp. 94-101, Colorado Springs, CO.

Harker, S., Eason, K. & Dobson, J. (1993), "The Change and Evolution of Requirements as a Challenge to the Practice of Software Engineering", *Proceedings of the IEEE International Symposium on Requirements Engineering*, pp. 266-272, San Diego, CA.

Hayes, J. & Dekhtyar A. (2005), "Humans in the Traceability Loop: Can't Live With 'Em, Can't Live Without 'Em", *Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering*, pp. 20-23, Long Beach, CA.

Hayes, J., Dekhtyar, A. & Osborne, J. (2003), "Improving Requirements Tracing via Information Retrieval", *Proceedings of the 11th IEEE International Requirements Engineering Conference*, pp. 138-147, Monterey, CA.

Heindl, M. and Biffl, S. (2005), "A Case Study on Value-Based Requirements Tracing", *Proceedings of the 10th European Software Engineering Conference*, pp. 60-69, Lisbon, Portugal.

IBM Rational Software (2007), "IBM Rational RequisitePro", ftp://ftp.software.ibm.com/software/rational/web/datasheets/version6/reqpro.pdf.

IBM Software (2007), "Rational Requisite Pro", http://www-306.ibm.com/software/awdtools/reqpro/

Institute of Electrical and Electronics Engineers (IEEE) (1995), *Software Development*, J-STD-016 standard.

Interactive Development Environments (1991), *Software Through Pictures: Products and Services Overview*, IDE, Inc.

International Council on Systems Engineering (2008), "INCOSE Requirements Management Tools Survey", http://www.paper-review.com/tools/rms/read.php.

International Organization for Standardization (ISO) (2000), *Quality Management Standard*, ISO 9000:2000 standard.

Jarke, M. (1998), "Requirements Tracing", *Communications of the ACM*, 41(12), pp. 32-36.

Karlsson, J. (1996), "Software Requirements Prioritizing", *Proceedings of the 2nd International Conference on Requirements Engineering*, pp. 110-116, Colorado Springs, CO.

Lefering, M. (1993), "An Incremental Integration Tool Between Requirements Engineering and Programming in the Large", *Proceedings of the IEEE International Symposium on Requirements Engineering*, pp. 273-276, San Diego, CA.

Leffingwell, D. (1997), "Calculating Your Return on Investment from More Effective Requirements Management," *American Programmer*, 10(4), pp. 13-16.

Lempia, D. & Miller, S. (2006), *Requirements Engineering Management*, presented at the 2006 National Software and Complex Electronic Hardware Standardization Conference, Atlanta, GA.

Levelson, L. & Turner, C. (1993), "An Investigation of the Therac-25 Accidents," *IEEE Computer*, 26(7), pp. 18-41.

Li, W., Vaughn, R. & Saiedian, H. (2002), "Pre-Requirements Traceability", *Encyclopedia of Software Engineering*, Marciniak, J. (editor), vol. 61, Wiley, New York, NY.

Macfarlane, I. & Reilly, I. (1995), "Requirements Traceability in an Integrated Development Environment", *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, pp. 116-123. York, England.

Matthias, J. (1998), "Requirements Tracing", *Communications of the ACM*, 41(12), pp. 32-26.

Munson, E. & Nguyen, T. (2005), "Concordance, Conformance, Versions, and Traceability", *Proceedings of the Third International Workshop on Traceability in Emerging Forms of Software Engineering*, pp. 62-66, Long Beach, CA.

Muvuti, F. & Lungu M. (2004), "Service Oriented Architecture for a Software Traceability System", Technical Report CS04-14-00, Department of Computer Science, University of Cape Town.

Naslavsky, L., Alspaugh, T., Richardson, D. & Ziv, H. (2005), "Using Scenarios to Support Traceability", *Proceedings of the Third International Workshop on Traceability in Emerging Forms of Software Engineering*, pp. 25-30, Long Beach, CA.

Nuseibeh, B. (1997), "Ariane 5: Who Dunnit?," *IEEE Computer*, 14(3), pp. 15-16.

Palmer, J. (1997), "Traceability", *Software Requirements Engineering*, Thayer, H. & Dorfman, M. (editors), IEEE Computer Society Press, New York, NY.

Paulk, M., Curtis, B., Chrissis, M. & Weber, C., "Capability Maturity Model for Software", Version 1.1. Technical Report, Software Engineering Institute, CMU-SEI-93-TR-024.

Radio Technical Commission for Aeronautics (RTCA) (1992), *Software Considerations in Airborne Systems and Equipment Certification*, DO-178B standard.

Ramesh, B. (1998), "Factors Influencing Requirements Traceability Practice", *Communications of the ACM,* 41(12), pp. 37-44.

Ramesh, B. & Edwards, M. (1993), "Issues in the Development of a Model of Requirements Traceability", *Proceedings of the 1st International Symposium on Requirements Engineering,* pp. 256-259, San Diego, CA.

Ramesh, B. & Jarke, M. (2001), "Toward Reference Models for Requirements Traceability", *IEEE Transactions on Software Engineering,* 27(1), pp. 58-93.

Ramesh, B., Powers, T., Stubbs, C. & Edwards, M. (1995), "Implementing Requirements Traceability: A Case Study", *Proceedings of the Second IEEE International Symposium on Requirements Engineering,* pp. 89-95. York, England.

Smithers, T., Tang, M. & Tomes, N. (1991), "The Maintenance of Design History in AI-Based Design", *Tools and Techniques for Maintaining Traceability During Design,* IEE Colloquium, Computing and Control Division, Professional Group C1 (Software Engineering), Digest Number: 1991/180, pp. 8/1-8/3.

Song, X., Hasling, B., Mangla, G. & Sherman, B. (1998), "Lessons Learned from Building a Web-Based Requirements Tracing System", *Proceedings of the Third International Conference on Requirements Engineering: Putting Requirements Engineering to Practice,* pp. 41-50, Colorado Springs, CO.

Spanoudakis, G., Zisman, A., Perez-Minana, E. & Krause, P., (2004), "Rule-Based Generation of Requirements Traceability Relations", *Journal of Systems and Software,* 72(2), pp. 105-127.

The Standish Group (1994 & 2006), *The Chaos Report,* http://www1.standishgroup.com/.

Swartout, W. & Balzer, R. (1982), "On the Inevitable Intertwining of Specification and Implementation," *Communications of the ACM,* 25(7), pp. 438-440.

Telelogic (2007), "Telelogic DOORS – Requirements Management for Advanced Systems and Software Development", http://www.telelogic.com/products/doors/doors/index.cfm.

U.S. Department of Defense (U.S. DoD) (1988), *Military Standard: Defense System Software Development,* DOD-STD-2167A.

U.S. Department of Defense (U.S. DoD) (1994), *Military Standard: Software Development and Documentation,* MIL-STD-498.

Watkins, R. & Neal, M. (1994), "Why and How of Requirements Tracing", *IEEE Software,* 11(7), pp. 104-106.

Wiegers, K. (2003), *Software Requirements,* Second Edition, Microsoft Press, Redmond, WA.

Windows Marketplace (2008), "Windows Marketplace: Product Details for Microsoft Windows Vista Business", http://www.windowsmarketplace.com/

Young, R. (2006), "Twelve Requirement Basics for Project Success", *CrossTalk, The Journal of Defense Software Engineering*, 19(12), pp. 4-8.

Zemont, G. (2005), "Towards Value-Based Requirements Traceability", DePaul University. http://facweb.cs.depaul.edu/research/TechReports/TR05-011.pdf.