Requirements Traceability and the Effect on the System Development Lifecycle (SDLC)

by

Glenn A. Stout
(STOUT)

Table of Contents

Overview

This paper will discuss the traceability between different types of requirements and other project artifacts, and how the SDLC benefits from this practice.

The author of this research paper is a senior functional specialist at The Revere Group, located in Deerfield, IL. During the year 2000, the consulting firm was engaged at ABC Company, with the task of establishing a software development methodology (SDLC), where the author served as requirements, testing, and deployment manager.

A key process that was put in place was tracing requirements from Use Cases to Test Cases, which allowed team members to see where a low level test case tracked to a high level requirement. This is valuable as it allows decision makers to know what high level requirement is unfulfilled should a test case fail, among other benefits.

This paper will discuss how requirements traceability is beneficial to a software development project, as well as discuss various requirement types and how one type of requirement flows from higher-level types of requirements. The paper is part academic study and part case study.

In summary, this document defines traceability and related terms, and it will discuss traditional requirements gathering methodology to show what the current shortfalls are without employing traceability. The next sections discuss several approaches to enable a traceability process in a software development project, discuss benefits of traceability, and list some of the software tools that assist project managers manage traceability in their projects. A case study within this research paper will discuss the actual project referenced above and how the traceability made it more successful, by showing a map of different types of requirements, how they relate to each other through traceability, and the benefits of adopting this particular methodology.

Purpose

Based on the author’s experience, when requirements gathering is discussed in general, the topic of the “requirements document” is often a big part of the discussion. The result of the requirements gathering process is generally a long document that is difficult for anyone to read or glean anything from, and is even more difficult to design and code from. Also, when changes are needed, it is rare that a process is in place to determine what other requirements may be affected by a particular change, which causes rework.

It is for these generally known reasons, and others that will be discussed later in this research paper, that requirements tracing is a required process that needs to be implemented, at some level, in every software development project.

Traceability

Traceability of requirements for a software project is a common concept, that when used, can offer considerable benefits for a development project (Leon, 2000, p. 54). These benefits will be explored further, later in this document. First, however, there are several different ways to define traceability, which will be discussed in the next section.

Traceability and Traceability Item Defined

To define traceability, the term "traceability item" first needs to be defined. This term is defined by Spence and Probasco (1998) as "Any textual, or model item, which needs to be explicitly traced from another textual, or model item, in order to keep track of the dependencies between them" (p. 1). A general definition of a traceability item would be a "project artifact." Spence and Probasco (1998), further define the term "requirements traceability" as an explicit tracing of requirements to other requirements, models, test requirements, and other traceability items such as design and user documentation.

It is also defined by Gotel and Finkelstein (1994) as the ability to describe and follow the life of a requirement, in both a forward and backward direction.

This leads to the general definition of traceability, which would be defined as the concept of "a connection" between traceability items or project artifacts.

Traceability Links

There are four types of traceability links that Matthias Jarke (1998) defines, with regard to requirements and other traceability items. Two of them are more heavily in the scope of the SDLC, the other two will be briefly mentioned, as they are outside of the actual development sphere, but do have an impact. The detailed discussion below is the author's paraphrase of two types cited, and determine added discussion from the author's experience.

Forward from Requirements – Components that are built from requirements have those requirements assigned to them. This is also key in the case where there is a change in a high-level requirement, the lower-level requirements that were derived or traced from that requirement need to be re-evaluated as well to ensure that the change at the top is enforced throughout the system (Jarke, 1998)

To trace requirements in this way also allows designers to package requirements into blocks that target a particular module or section of the system that needs to be built, based on that block of requirements. The testing team can also create test cases that directly relate to the requirements, and when this is complete, a person can trace a high-level requirement down to the actual test case, and verify that the test will properly validate the requirement. If traceability were not employed, this verification would be very difficult.

Backward to Requirements – A verification point, where design elements that have been built can be traced back to a requirement. By going backwards, team members can also verify that no "gold plating" has been done, which is the adding of features for which requirements do not exist (Jarke, 1998).

Starting from the low level requirement and tracing backwards also assists unit testers with their testing, so they can see what part their module plays in the big picture of the system, based on requirements. While testing, if a defect is found, and the low level requirement is known as the failure point, it can be determined what high-level requirement would then not be satisfied if it were not repaired. With this known, and educated decision can be made whether or not to correct the defect, depending on the priority of the high-level requirement.

Forward to Requirements – Maps stakeholder needs to the requirements, so that when there is a change to the vision of the system, all requirements in that chain can be looked at for impact (Jarke, 1998).

This is important regarding the situation where a "what if" analysis is needed, to determine costs and scope of a high-level requirement changing. If one does change, there is enough information to determine the order of magnitude of the overall change needed to the system.

Backward from Requirements – Allows the verification that the system meets the needs of the user community (Jarke 1998). Simply put, this allows us to look at the requirements, and determine whether or not the user needs have been met.

In summary, "Traceability has the most value in a methodology when the links are bi-directional" (Ecklund, Delcambre & Freiling, 1996, p. 353). This is important so that when changes are made to the system, the impacts are easily identified. Linking "downstream" allows team members to see the scope of a change. Tracing "upstream" allows the team to see why a particular decision was made, based on the high-level requirements (Ecklund et al, 1996).

Traditional Requirements Gathering

Now that traceability and some related terms are defined, we can use these terms in the discussions below. To show that requirements traceability is needed, it first must be shown that the practice is not used widely. Then, by offering research and examples, we must show that by implementing requirements traceability, especially by employing use cases, we can improve the process.

In General

A literature survey yielded the following information pertaining to current uses of traceability of requirements:

Marco Leon (2000), mentions that traceability usage is rare, by noting, "It is a very valuable but seldom used technique in today's development processes. Traceability analysis is rarer still in the internet development industry, where it is even more essential" (p. 54). Scott Ambler (1999), states "It's rare to find a software project team that can honestly claim full requirements traceability throughout a project, especially if the team uses object-orientated technology" (p. 48).

From these two quotes, the author's experience, and from other research gathered, the statement can be made that requirements traceability is not used as often as it should be, and should be used more if software quality is a goal of the development project team.

Even when traceability is put in progress, it is not always done for the right reasons, or is done in a way where it does not offer a long-term benefit. In a survey conducted to determine how projects were using traceability, the people surveyed ultimately fell into one of two categories; low-end users and high-end users. Low-end users see traceability as a mandate, and where high-end users see it as a way to build knowledge. The findings were that "Low-end users see traceability mostly as a costly defense against criticism and liability lawsuits." (Domges & Pohl, 1998, p. 33). That does not mean that they did not successfully employ traceability, because "Companies with life-critical business, such as aerospace industries, pharmaceutical and medical engineering companies, have long been forced to document each and every activity at a detailed level" (Domges & Pohl, 1998, p. 33). These companies can be low-end or high-end users. However, it was also found that "High-end users see traceability as an investment in corporate knowledge asset management within and beyond information systems engineering"(Domges & Pohl, 1998, p. 34). These are two varying ways to look at the process of traceability of requirements.

Scott Ambler (1999) notes that there are two questionable motivations to establish traceability, one is that it would satisfy a contract obligation, and the other echoes the quote cited earlier about protection from lawsuits.

Even the Department of Defense uses traceability of requirements, but after a review of the regulation, it seems to only mention that traceability needs to be implemented from the requirements to the design. It does not implicitly mention that requirements should trace to each other (Department of Defense, 1988), which is an important part of traceability.

Engaging in the Practice of Tracing Requirements

To instill the practice of requirements tracing within an SDLC, it must be understood that this is a process that needs to be managed, as Jarke (1998) states: "Tracing then is a sub process of evolutionary system development that supplies and exploits these traces" (p. 34). Analysts and designers are the key people involved with tracing requirements, the former responsible for the tracing from the stakeholder needs to the requirements model, and to the test model, where the latter is responsible for tracing requirements to deliverables (Ambler, 1999).

Since this is a new process, and like other processes, it needs to be customized depending on the project needs. According to Domges and Pohl (1998), "If requirements traceability is not customized it can lead to an unwieldy mass of unstructured and unusable data that will hardly ever be used" (p. 54). If the project is very large, with thousands of requirements, then tracing down to the lowest degree may not be in the best interest of the project, unless building a medical device where someone's life depends on it, for example. If with that same magnitude of requirements the team is building a video game, some requirements tracing would still be necessary to produce quality software, but perhaps not down to the absolute lowest level. Project managers need to use their best judgment when establishing the traceability process.

Traceability also is important when the system changes, as we find that: "Requirements tracing is emerging as an effective bridge that aligns system evolution with changing stakeholder needs" (Jarke, 1998, p. 32). From this it can be noted that if the high-level requirements change, we know what lower level objects within the system need to change.

Software Requirements Specification (SRS)

In order to discuss how using requirements traceability is more beneficial than not tracing requirements, we must discuss the traditional software gathering methodology of the Software Requirements Specification (SRS). We must also explore the question whether of how easy it is to trace requirements using this methodology.

The SRS is generally a document-centric approach to requirements gathering where the deliverable is a main requirements document. Karl Wiegers (1999), a respected leader in requirements study, states “As the ultimate repository for the product requirements, the SRS must be comprehensive: *all* requirements should be included” (p. 149).

Wiegers (1999) offers the methodology for tracing requirements of creating a traceability matrix (several to choose from), to show how one requirement “traces-to” or is “traced-from” another requirement. This practice is not specifically mentioned to be used with use cases (another requirements gathering methodology) or with an SRS, it is stated as a general practice, and does not state that it is easier or harder to implement requirements traceability using an SRS or a use case. Therefore, although it is possible to employ requirements traceability using the SRS, research and the author’s experience indicate that it is easier to establish requirements traceability when employing use cases in general, which are discussed in the next section.

Use Case Methodology

The methodology for gathering requirements that employs use cases as opposed to the traditional SRS, lends itself to easier requirements traceability. The use case allows for traceability to be an easy addition to the requirements gathering process, instead of an additional extra process that happens at the end of the requirements gathering process.

Use cases have been studied for requirements traceability. According to Kulak and Guiney (2000), “Use cases can provide requirements traceability effectively through the lifecycle because they are a building block for system design, units of work, construction iterations, test cases and delivery stages” (p. 59). This is a key reason to study the benefits of employing use cases for requirements traceability.

A Rational Software whitepaper entitled “Traceablity Studies for Managing Requirements with Use Cases” by Ian Spence and Leslee Probasco (1998), is a wealth of information pertaining to the topic at hand. Rational, a self-proclaimed leader in the software development methodology business, is offering this document as a “best practice” for those who would employ traceability. Rational calls their development lifecycle the Rational Unified Process (RUP) (Kruchten, 1999). They offer four main strategies to employ traceability, and the one that is recommended is the one that will be discussed here, which is the “Features Drive the Use Case Model.”

In this strategy, features are documented in what Rational calls a “Vision Document,” which is a document that captures the stakeholder views on what the system should do. If there are requirements, such as non-functional requirements, they are captured in what Rational calls a “Supplementary Specification Document,” which describes all requirements that are not in the vision document. Spence and Probasco (1998) summarize this approach by noting, “In this case the Use-Case Model acts as the main statement of the functional requirements” (p. 5). Other documents mentioned above, along with the glossary, round out the document set.

The view of the strategy, summarized here, has the Stakeholder requests traced to the Vision document, as well as the Use Case Model and Supplementary specifications. The Use Case Model and the Supplementary specifications are directly traced from the Vision Document. From the Use Case Model and the Supplementary specifications, the Design Model, the Test Model, and the End-User documentation and Training Materials are traced. Figure 1 in the figures section at the end of this document shows how this logically lays out (Spence & Probasco, 1998).

Benefits of Traceability

There are many benefits to establishing requirement tracing within a software development lifecycle. Marco Leon (2000) offers several, some of which in summary are:

Coverage Analysis is easier to execute, since all requirements are traced to higher-level requirements, it can be verified that all requirements are satisfied. A Better Design is the result of requirements tracing, as "best" designs are possible when complete information is available to the architect. Fewer Code Reworks is another benefit offered, as tracing enables the team to catch potential problems earlier in the process. Change Management is improved, as when a requirement is changed, the entire "trace" can been reviewed for the impact to the application. Mr. Leon (2000) states that all of these benefits ultimately result in a shorter development cycle and more on time launches.

Along the lines of change management, if use cases are employed, (Ecklund, Delcambre & Freiling, 1996) state that "Traceability allows us to gauge the scope of a change with respect to any level of a system's evolving design by following the traceability links forward from affected use cases to the level we are examining. Understanding the scope of a change on any level enables us to make judicious decisions about how to change the design at that level" (p. 352).

When regarding the maintenance of a system, Domges and Pohl (1998) state that requirements traceability is a prerequisite for effective system maintenance and consistent change integration. Also stated in the same article is the negative effect to a project if tracing is not done. It leads to a decrease in system quality, causes revisions, and therefore, increases project costs and time. It results in a loss of knowledge if key individuals leave the project, could lead to wrong decisions, misunderstandings, and miscommunication.

Disadvantages to Tracing Requirements

According to current rhetoric, as it has been established throughout the course of this research paper, project managers in today's development world do not feel that requirements tracing is not an absolute necessity. It could be that there are perceived negatives that are associated with requirements tracing, and real negatives associated with the practice.

Obvious negatives are that it takes extra time and effort during the project to keep up with the tracing of requirements, as well as increased maintenance effort, especially in an evolving system that requires numerous changes for future releases. Whether this is a real or perceived negative is up to the project team, as the effort to perform traceability is not negative if the information captured is used in the future to save time and effort. At the time of performing the action of tracing requirements, some may feel that the effort is a waste of time.

This subject was mentioned in an article by Jarke (1998), where it is mentioned "Establishing and maintaining requirements traceability is an expensive and politically sensitive endeavor. Developers are not exactly known for their love of documentation. Traceability should come as a side effect of their daily productive work rather than imposing additional bureaucracy" (p. 35). This is one way to offset this particular perceived negative to tracing. Therefore, when the traceability process is established, a certain amount of change management needs to be employed, especially with regard to developers, who need to be convinced that the minor daily effort of requirements tracing will have payoffs in the future.

A real negative is the tendency to not know when to stop tracing. Some tracing may go “so deep” that the effort is counter productive. Leon (2000) states, “If you have been diligent about managing the project and are still pressed for time, then it may not be prudent to conduct a full traceability analysis. You certainly don’t want the tracing process to be the cause of a delayed launch!” (p. 55). Setting a clear plan for the traceability process, and keep in mind what may need to be scaled back if the project timeline is being cut can offset this negative.

Tools

There are many software tools that are available to allow users to trace requirements throughout the system development lifecycle. See APPENDIX A for a list of the requirements traceability tools, according to the International Council on Systems Engineering. The website does not rate the tools, or give any significant details pertaining to them, but they are listed, complete with the website where additional details can be found. The author is personally familiar with the tool RequisitePro, a tool by Rational, Inc. Features of this tool include the ability to create a customized requirement types, and trace requirements to each other by type, or in hierarchical sequence. There are reports to print out the requirements in sequence, and to show the traceability from the highest-level requirement, down to the lowest level test case. It was by using this tool that enabled the author to initially understand the concept of requirements traceability, and eventually led to the interest and creation of this research paper.

Case Study

In this case study, where the author, a consultant for ABC company, was assigned to a client site in the role of requirements manager. That experience was the basis for the traceability model that will be presented next. This model is depicted in Figure 2 at the end of this document.

The Rational strategy defined earlier in this document, even when defined at a lower level, in the author’s opinion, still leaves some “real-world” details out. For example, all requirements are called “software requirements” in this high-level view. In this case study, requirements types are drilled down to a lower level of granularity, to offer a variant on the Rational approach. It is similar to the Rational approach, with common requirements types used throughout. The difference, the author believes, is how this is laid out and how the requirements flow together.

Requirements Types

In this approach, there are several categories of requirements, these are: Stakeholder, High-Level Business, Application, Data, Security, Test (Quality), Supplementary, and Change Management. Some of these categories have a lower level of granularity associated with them, which will be discussed in the sections below.

Stakeholder Requirements

These usually come from the CEO of the company, or the body of the Board of Directors. These are very high-level. They usually focus on a major “problem” that they want solved. These can also include high-level expectations for: Performance (“instantaneous”), Availability (“24x7x365”) and Security (“totally secure”).

High-Level Business Requirements

High-level requirements will fulfill Stakeholder Requirements. They are traced directly from them. For example, if a Stakeholder request is "to allow customers to directly interface with our company," a High-Level Business requirement may be to "build website to allow external business interface." High-Level business requirements trace to Application Requirements, discussed next.

Application Requirements

Application requirements drive the development of the application. They fall into the following categories; Use Case Requirements, Business Requirements/Rules, Functional Requirements, Technical Requirements, User Interface Requirements, "Look and Feel" Requirements and Navigation Requirements. There are some very fine lines between these types, and the ultimate strategy may be to combine these categories as needed.

Use Case Requirements

These start in the form of Use-Case Documents, but can be broken down into individual requirements. The use case describes the basic flow of the system, alternate paths through the system. Business Rules, Navigation and User Interface requirements are all derived from the use case, and are traced from it.

Business Requirements/Rules

A Business Requirement or Rule is something that is specific to the business. These are sometimes listed as "non-functional" requirements, as they do not have any function to them. The rules exist whether or not a system exists. An example of this would be, for a transportation firm,"Trucks only can travel 55 mph," and another would be "We accept no bid less than $100,000." This is also likely a list of data that is required to be captured, with the data including the data items needed, such as Name, address, etc. If there is a business need for data to be in a certain format, or minimum or maximum length for a text field, for example, that too is captured here. Any edit, validation, or default value that serves a business purpose is also in this category. This list of business rules is often captured in a business catalog. These are all should be traced from the use case.

Functional Requirements

These are driven and traced from the Business Requirements and Rules. They describe how a business rule is carried out, or how data is captured. These are usually in the format of "system shall do this or do that."

Technical Requirements

These are derived during the low level design process, and sometimes during development. They are driven from and traced from the Functional Requirements. They are not "user" features of the system, but describe the low-level required actions of the system, that only a designer or developer would know. For example, "When transaction is being processed, and an error of type XYZ occurs, a rollback of the database transaction is required." They can also include details of an error message, and error handling requirements.

User Interface Requirements

These are driven from Functional and Use Case Requirements, are traced from them both, depending on where they were derived from. They include items such as screen layout, tab flow, mouse and keyboard use, what controls to use for what functions (e.g. radio button, pull-down list), and other "ease of use" issues.

"Look and Feel" Requirements
These are driven from Functional and Business Requirements, and are traced from them both, depending on where they were derived from. These can be categorized as the "non-functional" User Interface type requirements. They deal with how fields look, regarding font, font size, colors, graphics, etc.

Navigation Requirements
These are driven and traced from the Use Case, as the Use Case lists the flow of the system, and the Navigation Requirements depict how that flow will take place. They are usually presented in a storyboard format, and should show the screen flow of each use case, and every alternate flow. Additionally, they should state what happens to the data or transaction for each step, for example, what happens to the data entered on a previous screen if the user decides to "go back?" They include the various ways to get to all screens, and an application screen map should be one of the artifacts derived in this category of requirements.

Based on the above Application Requirements, the Data Requirements are derived, and are traced from the Application Requirements.

Data Requirements
Based on the data needed in the Business Rules, and taking other requirements types into consideration, the database platform, database tables, columns, data types, size and other database attributes can be created.

Security Requirements
These are traced from wherever they need to be traced from, as they encompass all requirements that have do directly with the application (see Figure 2). They mainly will come from Business Rules.

Test (Quality) Requirements/Test Cases
The Test Cases are traced from all types of requirements, and are shown in Figure 2 as the all-encompassing box that all requirements are inside of. This shows that all requirements can have a test case traced to it. If that test case fails during testing, the team should be able to see what higher-level requirement would not be fulfilled.

Supplementary Requirements
Traced from Data and Application requirements, these refer to the technology infrastructure and operations type of requirements, such as hardware and network requirements. There are many Operations Requirements that must also be captured, in the categories of, Administration Requirements, Maintenance Requirements, Disaster/Recovery Requirements, Skill Set of personnel Requirements, Change Management Requirements, User Documentation Requirements, Training Requirements, and Demo/Presentation Requirements. Other non-functional requirements that do not fall into any other category would fall into this "catch-all" category.

Case Study Notes

In the actual project development cycle of the project discussed, all of the above requirements were not addressed, as this was the first time that some of the above types of requirements were even documented at the client site. The ones that were identified at the time, were entered and traced, and test procedures were written directly from the test cases identified, based on the requirements. The analysis of the coverage of the test procedures indicated that the coverage was very good for most things, but defects were being found during the navigation of the application. It was determined that navigation requirements were not captured, and this led the development team to interview users on a case by case basis, to determine what the navigation requirements were, and it was directly programmed as the users stated. The same was for certain user interface requirements as well.

It was only after a review of the project took place were some additional requirement types identified, such as navigation requirements, and categorized by the author for his own use for future projects.

Case Study Summary

The results of this case study show a few results that are worthy of review. Most importantly, when requirements were explicit, and traced to test cases from use cases, the testing coverage was excellent for those requirements, and the programming was well above average. However, in the case of the navigation of the application, it performed poorly. This is because navigation requirements were not captured, and therefore not traced. This happened despite the fact that use cases were used in this SDLC, and the RUP was loosely followed. However, as previously stated, the RUP is fairly vague in some places, and specific requirements types is a place where there is some ambiguity. If the requirements team had been looking to document navigation requirements, they would have been documented, and the project would have been more successful.

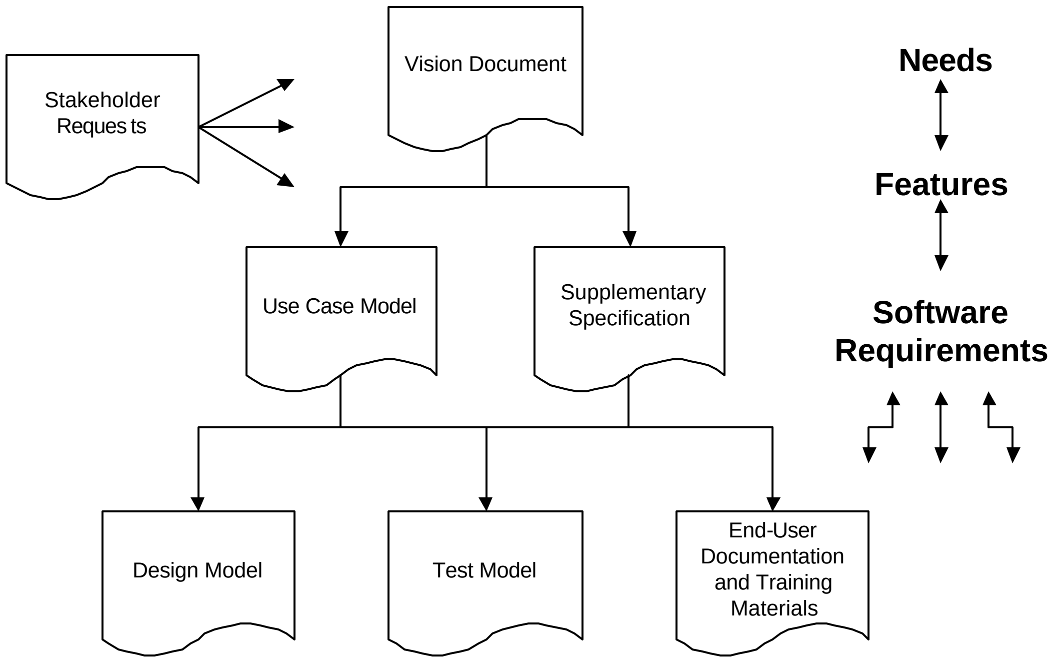
Therefore, the adoption of the practice of identifying all requirement types possible, and then determining whether or not the project needs to collect that type of requirement, would be a good baseline to begin the requirements gathering process. After that, all requirements can be collected, derived from each other, and traced from Use Cases to Test Cases. The above two processes, executed in concert, would be, in the author’s opinion, a beneficial practice.

Summary

Requirements tracing is not a new concept. According to the research presented in this research paper, it is a process that is needed, but not done as often as it should be, for many reasons stemming from budget concerns, to ease of executing the actual process. It has been shown throughout the course of this research paper that the practice of employing Use Cases as a requirements gathering methodology, and having requirements flow from them down to test cases, while maintaining traceability throughout, will result in many benefits to a SDLC.

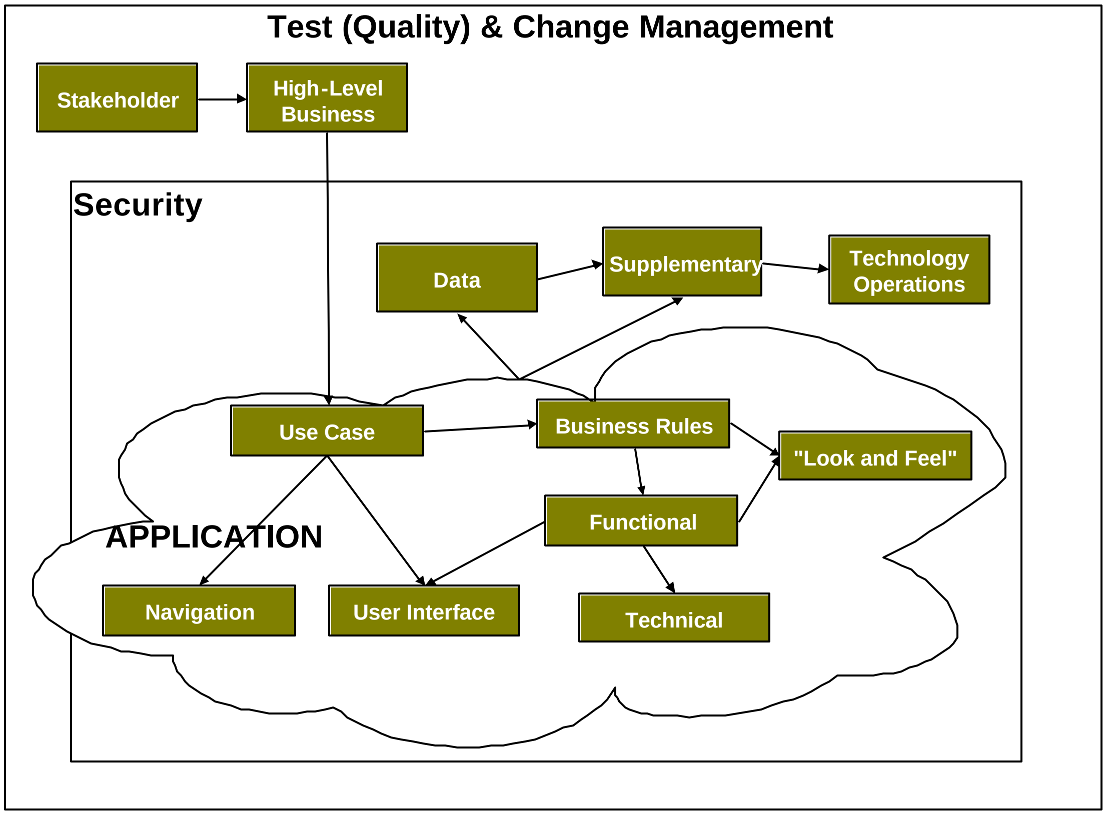
Figures:

<u>Figure 1: Default Rational Unified Process Traceability Strategy</u>



Source: (Spence & Probasco, 1998, p. 6)

Figure 2: Author's Approach to Requirements Traceability (Case Study)



Appendix A: SE Tools Taxonomy - Requirements Traceability Tools

This table was found at the website:

http://www.incose.org/tools/tooltax/reqtrace_tools.html

Some of the tools that are listed on the website have been removed from the table, due to that they were no longer available. The "Last Updated," "Hardware Supported" and "Operating Systems" columns were also removed for space considerations.

Description: Requirement Traceability Tools enable the engineer to link requirements to their source, to changes in requirements, and to modeling elements that satisfy the requirements. They provide traceability among the successive documents that are used to review the system development.

Tool Name	Vendor	Description	Uniform Resource Locator
AnalystStudio	Rational Software	Tool Suite. Includes RequisitePro, Rose, SoDA and ClearCase	http://www.rational.com/
Caliber-RM	Technology Builders, Inc (TBI)	Requirements traceability tool	http://www.tbi.com/products/caliber.html
CORE	Vitech Corporation	Full life-cycle systems engineering CASE tool. It supports the systems engineering paradigm from the earliest days of concept development and proposal development, requirements management, behavior modeling, system design and verification process.	http://www.vtcorp.com
CRADLE/REQ	3SL (Structured Software Systems)	Requirements Management tool capable of storing within its database, graphs, spreadsheets, tables, diagrams and any other information as part of a requirement.	http://www.threesl.com
DOORS	Telelogic (was QSS)	Requirements traceability tool	http://www.telelogic.se/
DOORS/ERS	Telelogic (was QSS)	Enterprise Requirements traceability tool suite	http://www.telelogic.se/
DOORSrequireIT	Telelogic (was QSS)	Requirements trace tool that is integrated with Microsoft Word. Data can be merged with DOORS databases	http://www.telelogic.se/
GMARC	Computer Systems Architects (CSA)	Generic Modeling Approach to Requirements Capture (GMARC). Toolset will also generate quality metrics for a specification enabling formal proof that use of the GMARC has improved the requirement set.	http://www.messages.co.uk/hp/csa
icCONCEPT	Integrated Chipware	Requirements traceability tool. Replaces RTM	http://www.chipware.com
IrqA	TCP Sistemas e Ingenieria	Integral Requisite Analyzer. A requirements management tool, but also a requirements analysis environment, that	http://www.tcpsi.es

Tool Name	Vendor	Description	Uniform Resource Locator
		includes facilities to support problem domain modeling and automatic domain analysis.	
ITraceSE	ITrace Systems	Requirements traceability tool	http://www.itracese.com
RDT	IGATECH Systems Pty Limited	Requirements traceability tool	http://www.igatech.com/rdt/index.html
RequisitePro	Rational Software	Requirements traceability tool. Also part of AnalystStudio	http://www.rational.com/products/reqpro/docs/datasheet.html
RIMS	Sygenex Incorporated	Requirements and Information Mamagement System (RIMS).	http://www.sygenex.com/
RTM	Integrated Chipware	Requirements traceability software. See icCONCEPT product.	http://www.chipware.com/rtm
RTS	Electronic Warfare Associates, Inc.	Requirements Traceability Systems (RTS). Complete foundation for tracking the requirements of a software/hardware project through the accompanying documentation and source code. This includes tracking the development and testing status of requirements	http://www.ewa.com/rts_overview.html
SLATE	SDRC SSG	System Level Automation Tool for Engineers (SLATE) is used to capture, organize, build and document system-level designs from raw concepts through structural partitioning. Interfaces to Office 97, Project and CASE tools.	http://www.tdtech.com
Systems Engineer	Blue Spruce	Requirements trace tool	http://www.bluespruce.net/bluespruce
Tofs	Tofs AB	Tool For Systems. Assists you in realizing and managing not only software, but also the manual (human) and hardware (electronic, hydraulic, mechanic, etc) parts of a system, which complete the system's missions together with the software.	http://www.toolforsystems.com
Tracer	RBD, Inc.	Requirements traceability tool	http://www.revbiz.com/
Vital Link	Compliance Automation Inc.	Requirements traceability tool	http://www.complianceautomation.com/
XTie-RT	Teledyne Brown Engineering	Requirements traceability tool	http://www.tbe.com

Source: (International Council on Systems Engineering, April, 2001).

Reference List

Ambler, Scott (1999, April). Trace Your Design. *Software Development*, 48-54.

Department of Defense (1988, February). *DOD-STD-2167A Defense System Software Development.* Retrieved May 28, 2001, from the World Wide Web: http://jcs.mil/htdocs/teinfo/directives/soft/ds2167a.html

Domges, Ralf & Pohl, Klaus (1998, December). Adapting traceability environments to project-specific needs. *Commun. ACM 41*, *12*, 54-62.

Ecklund , Earl F., Delcambre, Lois M. L. and Freiling, Michael J. (1996). Change cases: use cases that identify future requirements; *Proceedings of the eleventh annual conference on Object-oriented programming systems, languages, and applications*, 342–358.

Gotel, O. & Finkelstein, A.W. (1994). An analysis of the requirements traceability problem. *In Proceedings of the Int'l. Conference Requirements Engineering, IEEE Computer Society Press*, 94–102. Colorado Springs, CO.

International Council on Systems Engineering (April, 2001). *SE Tools Taxonomy – Requirements Traceability Tools.* Retrieved May 27, 2001, from the World Wide Web: http://www.incose.org/tools/tooltax/reqtrace_tools.html

Jarke, Matthias (1998, December). Requirements tracing. *Commun. ACM 41*, *12*, 32-36.

Kruchten, Philippe (1999). *The Rational Unified Process*. Reading, Massachusetts: Addison-Wesley.

Kulak, Daryk & Guiney, Eamonn (2000). *Use Cases: Requirements in Context*. New York, New York: ACM Press.

Leon, Marco (2000, September). Staying on Track. *Intelligent Enterprise*, 54-57.

Spence, Ian & Probasco, Leslee (1998). *Traceability Studies for Managing Requirements with Use Cases.* Retrieved April 17, 2001 from the World Wide Web: http://www.rational.com/products/whitepapers/022701.jsp

Wiegers, Karl (1999). Software Requirements. Redmond, Washington: Microsoft Press.