Requirements Traceability

prepared by

Michael Morckos
ID : 20363329
Electrical and Computer Engineering
April 4, 2011
1.0 Introduction

In software engineering, a requirement is a singular documented need of what a particular software system should be or perform. Requirements constitute the most important part of the software engineering process of a system since they are the statements that identify necessary attributes, capabilities, characteristics, and qualities of the system in order for it to have value and utility to a user [1]. A properly stated and formatted requirement is of vital importance to the development process of the software system since it ensures an efficient, timely and cost effective engineering and development process. Some of the attributes of a good requirement are.

- Unitary. The requirement addresses one thing specifically.
- Complete. The requirement is fully stated with no missing information.
- Consistent. The requirement does not contradict any other requirement and is fully consistent with all authoritative external documentation.
- Current. The requirement has not been made obsolete by the passage of time.
- Feasible. The requirement can be implemented within the constraints of the project.
- Unambiguous. The requirement is concisely stated.
- Traceable. The requirement is authoritatively documented and meets all or part of a business need as stated by stakeholders.

Requirements traceability is an important sub-discipline of requirements management within software development and systems engineering which is concerned with documenting the life of a requirement and providing bi-directional traceability between various associated requirements and other artifacts in the development process. Tracing a requirement means identifying all parts of the software product that were conceived by it and the ability to trace back from the products to the requirements [2].

This report provides a brief overview of requirements traceability. The second section illustrates some of the definitions of requirements traceability and the motivations for having traceability in the software development process. The third section provides a detailed literature about the methods of requirements traceability and how it is implemented into the software development process. The fourth section provides a brief overview of the benefits and feasibility of requirements traceability. The fifth and final section concludes the report.
2.0 Requirements Traceability

2.1 Definition

According to [3], traceability as a general term is the “ability to chronologically interrelate the uniquely identifiable entities in a way that matters.” The word chronology here reflects the use of the term in the context of tracking an object from its source of creation till it reaches its destination. The IEEE Standard Glossary of Software Engineering Terminology defines traceability as “the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another.” [4]

In software engineering, different sources define requirements traceability from different points of view. According to [1], requirements traceability is “the ability to describe and follow the life of a requirement, in both forward and backward directions.” In addition, [5] states that requirements traceability is “the ability to define, capture and follow the traces left by requirements on other elements of the software development process and the trace left by those elements on requirements.” Both [1] and [5] give conceptual definitions for requirements traceability. While [1] emphasized tracking of requirements through all the phases of development, [5] states that traceability involves other development artifacts as well, by emphasizing the relationship between requirements and the other artifacts such as specification statements, designs, models and developed components. A practical definition is supplied by [6], which states that requirements traceability is an intensive iterative process which involves labeling each requirement in a unique and persistent manner so they can be unambiguously referred to throughout the development cycle.

Requirements traceability is a vital task in the design and development of software systems which allows tracing the requirement from the instance it is conceived until it is implemented into running code as a specification. Requirements traceability is one of the characteristics of excellent requirements specifications [6].
2.2 Motivation for tracing requirements

The motivation for tracing requirements comes from the rapidly growing complexity of software systems and the constant need for rapid evolution and upgrade. Based on a real life situation mentioned in [6], missing a requirement in the final product can be both time and money consuming, especially if the requirement is for a safety-critical functionality of the system or if the customer is not satisfied with the final system. Basically, requirements tracing provides a way to demonstrate compliance with a contract, specification, or regulation. At an advanced level, requirements tracing can improve the quality of the products, reduce maintenance costs, and facilitate reuse. Requirements traceability is intended to ensure continued alignment between stakeholder requirements and system evolution [7]. The main purpose of requirements traceability is to facilitate the following.

- Understanding the software under development and its artifacts.
- Ability to manage changes effectively.
- Maintaining consistency between the software and the environment in which the product is operating [2].
3.0 Methods of Requirements Traceability

3.1 Traceability links

Traceability links are used to track the relationship between each unique requirement and its source. For instance, a requirement might trace from a business need, a stakeholder, a business rule, an external interface specification, an industry standard or regulation, or from some other source. In addition, traceability links are also used to track the relationship between each unique requirement and the work products to which that requirement is allocated. For example, a single requirement might trace to one or more architectural elements, detail design elements, objects/classes, code units, tests, user documentation, and/or even to people or manual processes that implement the requirement [8]. Good traceability practices allow for bi-directional traceability. According to [6 7 9], and as illustrated in Figure 1, there are four types of traceability links that constitute bi-directional traceability.

![Figure 1: Bi-directional traceability links.](image)

- **Forward to requirements.** Maps requirements source/stakeholder needs to the requirements, which can help to directly track down requirements affected by potential changes in sources or needs. This also ensures that requirements will enforce all stated needs.

- **Backward from requirements.** Helps to identify the origin of each requirement and verify that the system meets the needs of the stakeholders.

- **Forward from requirements.** As requirements develop and evolve into products, a product can be traced from its requirements. Forward traceability ensures proper direction of the evolving product and indicates the completeness of the subsequent implementation. For example, if a requirement cannot be traced forward to one or more products then the product requirements specification is incomplete and the resulting product may not meet the needs of the business [8].
• **Backward to requirements.** This link traces specific product elements *backward to requirements*. Backward traceability can justify the need and existence of that component and verify that the requirements have been kept current with the design, code, and tests [8]. Moreover, it helps to verify that no “gold plating” has been done, which is the adding of features for which requirements do not exist.

Bi-directional traceability links give the ability to analyze the impact of changes where all products are affected by a change in requirements and all requirements are affected by a change or defect in products. Moreover, they provide continuous assessment of the current status of the requirements and products by identifying missing requirements.

In addition, Figure 2 summarized many kinds of direct traceability relationships that can be defined in a project. A project may not implement all kinds of traceability relationships. However, the choice of traceability relationships is a major contributor to success and efficient maintainability of the system under development.

![Figure 2: Some possible traceability relationships [6].](image-url)
3.2 Traceability matrices

A traceability matrix is a document, usually in the form of a table, that correlates any two baselined documents that require a many to many relationship to determine the completeness of the relationship.

In software engineering the Requirements Traceability Matrix (RTM) is a classical tool to help ensure that the project’s scope, requirements, and deliverables remain “as is” when compared to the baseline [10]. Thus, it “traces” the deliverables by establishing a thread for each requirement, from the project’s initiation to the final implementation. A traceability matrix summarizes in a table form the traceability from original identified stakeholder needs to their associated product requirements and then on to other work product elements. In a traceability matrix, each requirements source, each requirement and each work product must have a unique identifier that can be used as a reference.

<table>
<thead>
<tr>
<th>User Requirement</th>
<th>Functional Requirement</th>
<th>Design Element</th>
<th>Code Module</th>
<th>Test Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>UC-28</td>
<td>catalog.query.sort</td>
<td>Class catalog</td>
<td>catalog.sort()</td>
<td>search.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>search.9</td>
</tr>
<tr>
<td>UC-29</td>
<td>catalog.query.import</td>
<td>Class catalog</td>
<td>catalog.import()</td>
<td>search.12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>catalog.validate()</td>
<td>search.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>search.14</td>
</tr>
</tbody>
</table>

Table 1: Portion of a single traceability matrix [6].

Table 1 illustrates a portion of a traceability matrix. Each functional requirement is linked backward to a specific use case and linked forward to one or more design, code and test modules. As the project gets bigger and more complex, more columns are added to extend the links to other work products. Including more traceability details takes more work, but it points the precise locations of the related software elements, which can save time during change impact analysis and maintenance.

An important feature of traceability matrices is that more information are added as the work gets done, not as it gets planned. For instance, the “catalog.import()” in Table 1 is added only when the code in that function has been written, tested and integrated into the code base of the project. This type of traceability matrices can accommodates one-to-one, one-to-many, or many-to-many relationships between system elements by having several items in a single table cell.
It’s impossible to perform requirements tracing manually for large and complex projects. Table 2 illustrates a two-way traceability which can be easily managed by automated traceability tools, unlike Table 1. In the table, each cell at the intersection of two linked components is marked to indicate the connection. Different symbols can be used in the cells to explicitly indicate “traced-to” and “traced-from” or other relationships. In addition, some tools automatically flag a link as suspect (visually using a red flag or a diagonal red line) whenever the entity on either end of the link is modified. For instance, as shown in Table 2, a change in UC 2.1 will trigger the flags, indicating that requirements 1.1.3 and 1.1.4 need to be inspected.

<table>
<thead>
<tr>
<th>Requirement Identifier</th>
<th>Reqs Tested</th>
<th>REQ UC 1.1</th>
<th>REQ UC 1.2</th>
<th>REQ UC 1.2.1</th>
<th>REQ UC 2.1</th>
<th>..</th>
<th>REQ UC 4.3.2</th>
<th>..</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Cases</td>
<td>321</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>..</td>
<td>4</td>
<td>..</td>
</tr>
<tr>
<td>Tested Implicitly</td>
<td>77</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.1</td>
<td>1</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.2</td>
<td>2</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.3</td>
<td>2</td>
<td>x</td>
<td>x</td>
<td>?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.4</td>
<td>1</td>
<td>x</td>
<td>?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>..</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.2.1</td>
<td>2</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>..</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Portion of a two-way traceability matrix.

3.3 Traceability and non-functional requirements

Non-functional requirements such as performance goals and quality attributes don’t always trace directly into code [5]. For instance, a portability requirement could restrict the language features that the programmer uses but might not result in specific code segments that enable portability. Another example is integrity requirements for user authentication, which lead to derived functional requirements that are implemented through passwords or biometrics functionality. In such cases, functional requirements are traced backward to their parent non-functional requirements and forward to products as usual.
3.4 Requirements traceability procedure

Requirements traceability for a project can be implemented sequentially as follows [6].

- Depending on the project, it is important to define the required relationships from the possibilities illustrated in Figure 2. As mentioned earlier, the choice of relationships is crucial to the project’s success and maintainability.

- Identifying the parts of the product to maintain traceability information for. This parts could be critical core functionalities and/or parts that are expected to undergo the most maintenance and evolution over the product’s life.

- Choosing the type of traceability matrix to use.

- Defining the tagging conventions that will be used to uniquely identify all requirements and system elements so that they can be linked together homogeneously.

- Identify the key individuals who will supply each type of link information and the personnel who will coordinate the traceability activities and manage the data.

- Educating the team about the concepts and importance of requirements tracing. Enriching the sense of importance and responsibility among the team members and stressing the need for ongoing creation of detailed traceability data.

- Auditing the traceability information periodically to make sure it is being kept current.
4.0 Advantages and Necessity of Requirements Traceability

4.1 Advantages of requirements traceability

There are many benefits to establishing requirement tracing within a software development lifecycle. Coverage analysis is easier to execute, since all requirements are traced to higher-level sources, it can be verified that all requirements are satisfied [7]. 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. Moreover, change management is improved, as when a requirement is changed, the entire “trace” can been reviewed for the impact to the application. [12] states that all of these benefits ultimately result in a shorter development cycle and reduced costs.

Regarding the maintenance of a system, [13] states that requirements traceability is a prerequisite for effective system maintenance and consistent change integration. Not implementing traceability can have negative effects. It could lead 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.

The benefits of implementing requirements traceability can be summarized as follows.

- Certification. Traceability information can be used in product certification to demonstrate that all requirements were implemented.
- Tracking. Recording traceability data during development allows for an accurate record of the implementation status of planned functionalities.
- Maintenance. Accurate traceability information facilitates making changes correctly and completely during maintenance, thus improving productivity.
- Re-engineering. Traceability information can be vital in case of re-using requirements from an old system into a new system.
- Reuse. Traceability information facilitates reusing product components by identifying components of related requirements and designs.
- Risk reduction. Documenting the component interconnections reduces the risk if a key team member with essential knowledge about the system leaves the project.
- Testing. In the testing phase, links between tests, requirements, and actual code can be used to track and identify components producing errors or unexpected behaviors. This can eliminate redundancy and save time.
4.2 Is requirements traceability always useful

Despite the many benefits of requirements traceability, many project manager’s nowadays do not see a really necessity for traceability due to a number of reasons. Obvious disadvantages 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 [12]. However, perceiving this as a disadvantage is relative. Some people see that traceability is worth while if the information captured is used in the future to save time and effort. Moreover, according to [11], “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.”

Therefore, when the traceability process is established, a certain amount of management change 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 pitfall of traceability is the tendency to not know when to stop tracing. Some tracing may go “so deep” that the effort is counter productive. However, setting a clear plan for the traceability process and having the ability to scale back if the project timeline is being cut can counter this drawback.
5.0 Summary

This report was a brief overview on requirements traceability, one of the milestones of requirements management in the software development process. Conceptual and practical definitions for requirements traceability were illustrated as well as the motivation for using traceability in projects. Moreover, the report demonstrated the methods and tools of traceability such as bi-directional traceability links and the traceability matrix. Finally, the report briefly stated the procedure for implementing traceability in software projects and the advantages and necessity of requirements traceability.
References


