Importance of Requirements Analysis & Traceability to Improve Software Quality and Reduce Cost and Risk

Author: Manju M. Kapoor aka Manju Mehta, Jet Propulsion Laboratory/Caltech, Pasadena, CA

Abstract

Requirements engineering plays a critical early role in producing quality software. Natural Language (NL) is still the most common and easiest way to express software requirements. Requirements must be testable, measurable, and defined to a level of detail sufficient for System Design Review. Analysts employ methods such as stakeholder interviews, use cases, and prototyping to elicit requirements from the customer. The objective of these methods is to ensure that a rightly interpreted requirement is defined and developed from the stakeholder’s point of view. With the interdependency of requirements and complexity of software systems, requirements analysis and traceability can be crucial to improving the verification process. Requirements engineering when correctly carried out helps reduce risk as well as cost, and improves the overall quality of software.

The goal of this paper is to emphasize the importance of developing complete and unambiguous requirements early in the project cycle (prior to Preliminary Design Phase). Having a complete set of requirements early in the project cycle allows sufficient time to generate a traceability matrix. Requirements traceability and analysis are the key elements in improving verification and validation process, and thus overall software quality. Traceability can be most beneficial when the system changes. If changes are made to high-level requirements it implies that low-level requirements need to be modified. Traceability ensures that requirements are appropriately and efficiently verified at various levels whereas analysis ensures that a rightly interpreted set of requirements is produced.

In practice, there are numerous tools and techniques for analyzing and managing such requirements. Proper use of tools such as Dynamic Object-Oriented Requirements System (DOORS) and use-case specifications can greatly help improve the analysis and verification process. Use cases are powerful tools for capturing the functional requirements of a system. Certain tools, for example MagicDraw, can even execute the use-case specifications when they are expressed in SysML and UML. Executing use-cases can help drive out conceptual, logical and semantic defects early in the process and can thereby lower costs, reduce time-taken and increase quality. Additionally, these tools allow test teams to effectively communicate findings to the appropriate personnel in a timely manner.

Effective use of various tools will be illustrated and relevant experience with NASA flight project/s will be discussed here. With the help of effective use cases and requirements trace matrix tools such as DOORS, software development engineers can better deliver systems on time, at reduced cost, and with fewer defects.

1. Introduction

Requirements analysis in systems and software engineering encompasses the tasks that go into determining the conditions or needs to meet a new or modified product. It takes into account the possibly conflicting requirements of the various stakeholders. Requirements engineering is a sub-discipline of systems engineering and is mostly concerned with determining the functions, goals and constraints of software and hardware systems.
The task of eliciting requirements needs communicating with customers and users to determine what their expectations are. One can call it requirements gathering. It also requires the identification of stakeholders because stakeholders are not limited to one organization.

The purpose of analysis is to enquire into the clarity, completeness, contradictions, and ambiguities of the requirements. It also includes determining ways to resolve the issues that result from analysis. The entire process is long and arduous and includes psychological skills as well. The system requirements must be clearly traceable to stakeholder needs.

In industries, the primary methods to identify ambiguities and inconsistencies are reviews and inspections. However, humans have difficulties identifying ambiguities and tend to overlook inconsistencies in NL Software Requirements Specifications (SRS). More modern techniques include use cases and prototyping. A combination of methods can be employed to establish the precise requirements of the stakeholders. Prototyping means mock-up of a product that allows users to visualize an application that has not been constructed, yet. It helps users get an idea of what the system will look like without waiting for the system to be built.

Use cases are descriptions of a system’s behavior from the user’s point of view. It is a technique to document the potential requirements of a new system or modified system. Use cases typically are written in end user’s language avoiding complex technical terminology. They can be co-authored by systems engineers and stakeholders. A good use case must contain a description of all of the desired permutations and combinations in which the users can work with the system or software. It is important to note that use cases do not provide any internal workings of a system, nor do they explain how a system will be implemented. They simply show how a user plans to perform a task and interacts with the system.

Use cases can also be written in SysML and UML and can be executed using IBM Rational Rhapsody or MagicDraw. IBM Rational Rhapsody is a model-driven development environment for systems engineering, software development, and testing and it is based on UML and SysML. It assists in finding defects early through continual testing – to aid in reducing costs. MagicDraw is a CASE tool developed by No Magic, Inc. MagicDraw has Plug-ins that support:

- Using SysML for systems engineering
- Executing use cases (written in SysML) by using appropriate plug-ins
- Working with DOORS for requirements management
- Working with IBM Rational RequisitePro (requirements management tool)

In practice, above-mentioned tools and techniques enable systems engineers to develop complete and unambiguous set of requirements early in the project cycle. Additionally requirement traceability can be accomplished by using tools such as DOORS. A combination of these tools and techniques can help save cost and ultimately improve the quality of software.

### 2. Improving Software Quality While Reducing Cost

#### 2.1 Traditional methods to approach systems engineering

Involvement of stakeholders, users, and test personnel early in the requirement generation process can help improve the software quality and reduce cost simultaneously. It is cost effective to hold ambiguity reviews and have a non-domain-expert look for ambiguous requirements (in addition to prototyping and use cases). Ensure that requirements are revised periodically to eliminate
ambiguities and improve contents and preciseness. Systems and software engineers must work in
concert with requirement authors and perform “triage” on the ambiguities. Requirements Triage
[2] is the process of determining which requirements a product should satisfy given the time and
resources. As an example:

- Major issues involving architecture and business policies must be resolved immediately. The
development of software can’t continue until these types of issues are resolved.

- Significant issues such as impacting single component can be showstoppers in limited areas and
must be resolved in a timely manner.

- Minor issues that have narrow or localized impact must be addressed, however those issues do
not stop the development process.

2.2 Model-based Systems Engineering Approach

Use Case Modeling and use of Dynamic Object Oriented Requirements System DOORS can
significantly reduce time spent to trace and manage requirements. The SysML requirements
modeling serves as a bridge between traditional text-based requirements and the modeling
environment. The requirements can be imported from a requirements management tool such as
DOORS or created directly into the modeling tool. SysML is a modeling language that supports
analysis, design, verification and validation of complex systems. The language is intended to help
architect systems and specify its components and is relatively new.

Execution of use cases written in SysML and UML is an emerging technique and is extremely
powerful one, if utilized properly. Note that unified modeling language (UML) is a standardized
general purpose modeling language. IBM Rational Rhapsody tool has been successfully exercised
and is being utilized by some organizations. However, it is not a commonly used technique.
MagicDraw (MD) is also very useful in creating use case models, diagrams etcetera. However,
execution of uses cases using MD is still in works and requires plug-ins. These tools are
discussed in details in sections 4 and 5.

2.3 Reduced Cost and Risk

If developers and test teams are provided with an unambiguous and complete set of requirements,
it would significantly reduce the development and test time as well as improve the software
quality. Reduced development and test time is translated into significant cost reduction for any
software project.

Studies have shown [1] that cost for a defect found early in the software development process is
$25 - $50 (per criticality one or two). If a defect is found during system integration and test
phase, cost can be $750 to $3000.00 (SEI). However, if a defect is found in operations or
productions, cost can be several thousand dollars. If something is ambiguous in the specs it will
most likely result in a defect (s) in the code. Therefore, developers must be provided with a set of
unambiguous and traceable requirements to reduce cost and risk.

3. Importance of Generating Requirements Early in the Project Cycle

Requirements have always been acknowledged as the backbone of any software system. With the
increasing demands for accelerated software delivery time and reduced budgets, it is vital that
requirements are fully understood and specified early in the project cycle. We all have heard of
the problems and challenges that we face with bad requirements. There are horror stories about
the things that go wrong, the cost overruns, the lost opportunities, the slips in schedules.
From my personal experience, I have seen schedule slips and cost overruns due to extended delays in developing a traceable, complete, and unambiguous set of requirements. Early requirement analysis and specification is one of the most important and difficult phases of the software development process for any system. This phase demands critical interactions with users and stakeholders. Any miscommunication at this point may lead to expensive errors and cost overruns during later development stages of the project.

Proper use of DOORS and other tools such as MagicDraw (used only for SysML) can help eliminate ambiguities and thus save time and cost. Efforts are being made within our organization to utilize tools such as MagicDraw to design Use Case Diagrams (UCD) and to learn more about Requirements Modeling using SysML.

4. Effective use of Dynamic Object Oriented Requirements System (DOORS) to Trace Requirements

There are challenges in implementing traceability between requirements, code and tests. These types of matrices are not always maintained in practice. It is primarily caused by failure to update matrices with changing requirements. Additionally, ambiguities in requirements can be a hindrance to maintaining the traces. On-going communication between testers and requirement owners is suggested in order to overcome the deficiencies of documenting changes in trace metrics. It is widely stated that the testing personnel should take part in requirement reviews. Setting traceability policy is highly recommended and failure to trace tests to requirements is one of the several ways to damage a project. Additionally, tools such as DOORS, FileMaker Pro, and Rational IBM RequisitePro are available to trace and manage requirements.

Many large companies including aerospace industry often use DOORS for managing and tracing requirements. Quality Systems and Software (QSS), a UK COMPANY, originally created DOORS, however this market-leading tool was acquired by Telelogic several years ago. It is essentially an object-oriented database that runs on PC or Unix. A Remote Desktop Client (RDC) is needed for Macintosh. It is often used in capturing, identifying, managing and tracing the necessary and sufficient set of requirements.

Tracing requirements among various levels is one aspect of traceability and can be managed by DOORS, if used appropriately. The other aspect of tracing is “linking test cases to requirements” which a test engineer does. Most developers and test engineers admit that if applicable requirements are not traced to one or more test cases, it leaves some gaps in test coverage and may cause risk. Traceability benefits in several ways e.g.:

- Improved test coverage and automation
- Improved efficiency of change management because change in requirements can be traced to appropriate test case/s.

DOORS tool is highly effective in linking requirements so that satisfiers (children) can be traced to parents and vice versa. Link is a connection between requirements indicating some sort of relationship within the same or different modules. Parent is a linked requirement usually at a higher level. A sub-allocation is called a “satisfier” [5] (or child). A requirement at higher level (N-1) will usually have one or more satisfier at lower level (N+1). Links are usually bottom up and tracing can be up or down. Below is an example of linking a parent to its satisfies:
In Figure 1, arrows indicate link(s); arrow pointing to red box is out-link to parent; arrow pointing to green box is in-link from satisfier.

Definition of traceability is that a requirement at level N will have one or more parent(s) at level N-1, and one or more satisfiers at level N+1. If the requirement on one end of a link changes, a suspect link flag is generated at the other end. The suspect link flag is cleared by the custodian after review and disposition [5].

5. Importance of Use Cases

5.1 What is a Use Case?

A use case is a sequence/scenario of steps that describes an interaction between a system and its users. It describes a mechanism in which users interact with the system and can be in textual format or a diagram. A Use Case Model (UCM) includes a textual diagram and textual descriptions. Unfortunately, there are no industry standards in existence to capture UCM, therefore there are many styles that can be used for the use case specification. Typically, users do not have to be humans, they could be other systems. A scenario can describe successful or unsuccessful interactions between a user and a system. An example of a scenario for weather forecasting system is as follows:

The user specifies a timeframe to retrieve weather data information for planet earth. Additionally, the user specifies a unit of measurement to display the retrieved results.

In response to above request, the following occurs:

- Request starts with getting weather information for specified time
- The system fetches weather information for specified time
- The weather information such as temperature, pressure and humidity is converted into the specified unit of measurement and sent to the user

5.2 Relationship between Use Cases and Requirements

Use Case Modeling is a technique that can help capture and communicate functional requirements for software system development and test phase, if used properly. A requirement describes a capability to which a system must conform and can be stated in a contract, specification or other formal document. Requirements can be functional or non-functional. Use cases can easily capture largest subset of functional requirements and describe the behavior of the system as it interacts with its users. However, use cases may not be all that you need to capture the requirements of a system. Early end user involvement and frequent use of requirements management tools play an important role in tracing and capturing software requirements.

Use cases can be written in SysML using either one of the following [4]:

- Activity diagrams: are similar to a functional flow diagram and are used to model behavior in terms of the flow of inputs, outputs, and control.
- Parametric diagrams: are used to create systems of equations that can be used to constrain the properties of blocks.
- State Machine Diagrams: are used to describe the state-dependent of a block throughout its life cycle in terms of its states and transitions between them.

Figure 2 –Example of Partial Use Case Diagram (UCD) for a flight project at NASA/JPL
In Figure-2, <<include>> relationship between use cases allows a base use case to include the functionality of an included use case as part of its functionality. The included use case is always performed when the base use case is performed. In addition to UCD, this relationship should be captured in the use case specification or the textual format.

5.3 Use Case Modeling

In general, the basic building blocks of a use case model are users & actors, the use cases, the relationship between use cases and actors, and the diagram in which they appear. Actors and users are typically people or other systems that are external to the system and usually control the system. They impose requirements on what the system being built must do.

A use case model is a complete model of a system’s environment and its intended function. It can also serve as a thread that unifies development phase and is input to activities in test and analysis phase. Use cases can be modeled with interactions, activities or state diagrams as mentioned in section 3. If use cases are expressed in SysML or UML, engineers can execute them using MagicDraw or IBM Rational Rhapsody to help drive out any requirements defects early in the process. If using MagicDraw, certain plug-ins are required before executing use cases.

6. Summary and conclusions

There is a lot of interest within our organization to find ways to improve and trace requirements with proper use of tools. One of the most common problems in gathering, analyzing and tracing requirements is lack of customer involvement, together with lack of resources. Experience on JPL/NASA flight projects has shown early user and customer involvement to be one of the most effective ways of improving requirements quality and thus reducing test and development time. In my opinion, early user involvement is still not done as often as it should be in view of its potential effectiveness. Therefore, it should form a vital part of an effective approach in improving requirements quality and thus reducing test and development time. During the software development and test phases, it can be very advantageous to arrange meetings between software developers, test personnel and users to elicit and understand the requirements. In real life, it is not easy for developers and test personnel themselves to have contact with users and understand their needs. It is an issue worth addressing early in the project cycle so that necessary adjustments can be made and resources can be allocated.

On the other hand, it is expressed that Model-based Systems Engineering (MBSE) approach will become standard practice in a similar way that it has with other engineering disciplines. Given the intense interest in moving towards model-based system’s engineering (MBSE) approach as opposed to document-based approach. In general model-based approach is becoming more prevalent in systems engineering, the goal is to eventually model functional and non-functional requirements with the help of Use Case Diagrams. By executing use cases written in SysML or UML and using DOORS as requirements management tool, engineers can eliminate defects and inconsistencies early in the project cycle and can thereby reduce time taken to analyze requirements. The tools and techniques mentioned in this paper (DOORS, IBM Rational Rhapsody, MagicDraw, SysML) along with early customer involvement can assist in improving software quality and thus reducing cost to a significant extent.
7. Acknowledgements

The author thanks Robert G. Elliott at Jet Propulsion Laboratory for helpful suggestions.

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

1. Bender RBT Inc.
2. The Art of Requirements Triage by A. M. Davis
3. Use Case Modeling by Kurt Bittner & Ian Spence
4. A Practical Guide to SysML by Sanford Friedenthal, Alan Moore, Rick Steiner
5. How to Track and Manage Requirements in DOORS, Jet Propulsion Laboratory, Pasadena, CA