Building and Applying Requirements Models

Indira Kuruganti and Dennis M. Mancl
{indira,mancl}@lucent.com

Abstract

Software requirements are traditionally documented in a relatively unstructured text requirements document. A better approach is to use modern tools to create a requirements model. A requirements model structures the information to facilitate management of requirements churn and flow-through of information to design and test artifacts. In this paper we describe the requirements model and explain how using such as model can lead to interval reduction and improved product quality.

Introduction

The traditional advice for writing software requirements is to create a large document or table containing detailed atomic requirements statements. Each statement is supposed to be as independent as possible from the other requirements and each statement should describe some behavior that the system under development must support.

A more modern approach for writing software requirements is to create a requirements model based on frequently executed scenarios. Several modern software development processes (such as the IBM Rational Unified Process [4] and usage-centered design [3]) ask the requirements writer to write descriptions of common usage scenarios.

However, scenarios are not sufficient to define a complete set of requirements for most complex software systems, such as telecommunications systems. The scenarios and use cases often need to be qualified by specific constraints and additional requirements need to be imposed by business processes and operational contexts.

Therefore, in this paper we propose a four-dimensional meta-model for requirements information that can be used to structure requirements for complex systems. The requirements information in each of the four dimensions is formatted to directly support the creation of other downstream software development artifacts such as design models and test cases. The requirements model is also structured to provide a mechanism for understanding the impact of changes to the requirements as the needs of the stakeholders evolve during the product's life cycle.

A Requirements Meta-Model

In the requirements meta-model, the structure of the requirements documentation is not a simple document. It is a set of four different sets of requirements information:

- *A use-case model* – A description of the most important behavior in the system. The use-case model for a system is composed of a set of use cases, a set of actor descriptions for the actors that participate in the use cases, and some optional diagrams. Each use case contains scenarios, i.e., descriptions of what happens when a user tries to use the system.

- *Business rules* – A set of conditions, policies, and conventions that are determined by the business domain in which the system will be used. Business rules are usually expressed as restrictions on the form and function of the system, e.g., the hardware to be used, computational formulas or algorithms, communications standards, and priorities.
- *Operational profiles* – A model that gives the details of the kinds of operations that will be simultaneously performed by the system in specified time intervals—in particular, the busiest hour of the day, the overnight maintenance interval, and the initial setup and configuration of a new system [5].
- *Architectural requirements* – A set of requirements statements that specify and quantify some of the desired system characteristics [1]. These requirements include statements about how dependable, serviceable, usable, and changeable the system will be.

When using this meta-model, each requirement will be one of four types of requirements object: a use case, a business rule, an operational profile, or an architectural requirement.

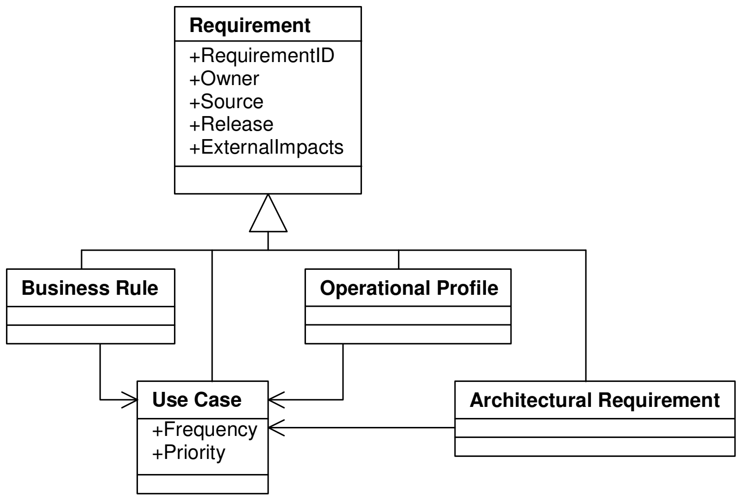

Figure 1: Basic requirements meta-model

Each kind of requirements object has some common attributes, so each of these attributes should be entered into the requirements model for any kind of requirement:

- *Requirement ID.* This is an identifier that uniquely identifies the requirement, so it can be referred to in other requirements, in architecture documents, in design models, and in test programs.
- *Owner.* This is the name of the person in the requirements-writing organization who is responsible for the requirement, especially for answering questions and resolving problems related to the requirement.
- *Source.* This is the person or document outside of the requirements-writing organization that originally supplied information about the need expressed in the requirement, e.g., a customer interviewed by the requirements writer or a standards document used by the requirements writer.

- *Release.* This is the name or number of the product release for which this requirement applies.
- *A set of names of other systems impacted by the requirement.* This is the list of external systems that might be affected by this requirement.

Some of the requirements objects need to have extra attributes, beyond the common set of requirements attributes. For example, each use case has two extra attributes:

- *Frequency.* This is the average or maximum number of times the use case will be executed in a given time interval.
- *Priority.* This is the level of importance of this set of scenarios to the stakeholders.

The requirements model also contains links between use cases and the other three kinds of requirements objects. These links define the behavioral context for a business rule, give a list of which use cases are used in an operational profile, and specify the use cases that are critical to satisfying an architectural constraint. These links may be traceability links or dependency links depending on whether the requirements objects have a parent-child relationship or are simply dependent on each other.

In summary, the description of the requirements meta-model is in terms of requirements objects, attributes of each kind of requirements object, and the possible links between requirements objects. This simple meta-model makes it possible to use computer-aided software engineering (CASE) tools, such as Telelogic DOORS, to store and manage a sophisticated requirements model.

Benefits of Adopting the Requirements Meta-Model

There are three main benefits of adopting this requirements modeling approach:

1. Requirements models make it easier to do “artifact flow-through.” The machine parseability and transferability of the information in the requirements meta-model enables the use of requirements information to jump-start the artifacts built by developers and testers. For example, some of the design-level use cases and sequence diagrams in the design model and test scenarios embedded in test cases can be initialized directly from the use cases in the requirements use-case model.
2. Once requirements information is structured in the meta-model format, it is easier to parse and run consistency checks on this information. This, in turn, reduces the number of errors in requirements that are passed on to downstream development processes. For instance, inconsistencies in the capacity and workload requirements on the system can be detected through consistency checks on the operational profiles during requirements engineering long before the system test stage when they are traditionally discovered.
3. Requirements changes are easier to make when the needs of customers change and evolve. Once requirements are structured in the meta-model format, it is easy to develop an actionable set of policies for linking requirements with design elements and test cases and using these links to understand the impact of requirements changes. For example, project team members can link architectural requirements to specific class diagrams and related test cases. They can use these links to assess the impact of proposed changes to

the architectural requirements on the existing design and test plan for the system before committing to these changes.

Therefore, by arranging the requirements artifacts in an organized model, errors in requirements can be detected early in the development process, downstream design and test activities can be jump-started earlier, and the changes to the requirements can be analyzed and applied more efficiently. In fact, if the modeling environment supports direct links between requirements, design, and testing information, the project team will have an opportunity to navigate these links to understand the end-to-end impact of proposed requirements changes before committing to these changes.

Implementing Requirements Modeling

Traditionally, requirements writers have produced text-based requirements documentation. By making a shift to model-based requirements and by using CASE tools to manage these models, software development projects can achieve significant improvements in development time interval and product quality.

There are many good choices for CASE tools for capturing requirements information. The scenarios in the use cases can be modeled in Unified Modeling Language (UML) using IBM Rational Rose or Microsoft Visio. Text-based requirements can be written using any number of word-processing tools. A project team might choose to store and manage the overall requirements model using a requirements repository tool, such as Telelogic DOORS or IBM Rational RequisitePro, or they might choose to adapt a relational database system or a change management tool.

The decision to move to a model-based approach for requirements documentation will impact many people in a software development organization, so it is best to attempt the transition in incremental steps. Depending on the extent of previous experience within the team with model-based development and CASE tools, project teams can choose one of three implementation levels:

- **Level 1. Use text-based templates for all parts of the requirements model.** Use traditional word-processing software to describe each requirements object and put all of the text into a requirements document. Adhere to well-defined best practices and rules for writing requirements objects and clearly distinguish these from other explanatory text. Annotate the textual requirements with visual diagrams, where possible, to increase the comprehensibility of the requirements document.

  There are significant benefits to organizing the requirements information as described in the meta-model, even without using tools for each element of the requirements. Just using document templates that cover all the requirements dimensions in the meta-model can help structure the collection and documentation of requirements and ensure greater coverage of often-neglected topics, such as business-related constraints and features needed for maintainability of the system. Adherence to rules for writing requirements can result in requirements documents whose text can be parsed to jump-start design and test artifacts.

- **Level 2. Put all requirements objects in a requirements management tool.** Some of the requirements objects will be defined using text-based templates and entered into the model; other requirements objects will be created using visual modeling tools. The requirements management tool will be used to control changes to the requirements model.

By creating visual models of parts of the requirements, requirements writers can increase the comprehensibility of the requirements. Further, by establishing a database of requirements, project teams can facilitate a systematic approach to requirements reuse. In addition, team members will have the ability to define links between different requirements and, hence, better understand the full extent of changes to requirements before committing to them.

- **Level 3. Use tool-based automation to perform flow-through of requirements information.** At this level, design models, test cases, and other software development artifacts are generated directly by tools that read the information in the requirements management tool.

  Project teams can leverage significant interval reduction and product quality improvements by adopting this approach. Improvement comes from the elimination or reduction of existing manual steps performed by architects/designers, developers, and testers (who are translating the requirements information into the format of their own modeling environment) before starting work on their own artifacts. Further, by automating the establishment of traceability links between requirements, design, and test artifacts, team members can more effectively understand the end-to-end impact of requirements and respond to changes in these requirements.

Experiences with a Lucent Project

We recently worked with a Lucent software development project to achieve significant improvements in development interval and quality through introduction of tools and software engineering practices. The tools introduction was expected to impact the work of a total of fifty systems engineers, designers, and testers. To this end, the project team chose a specific set of tools:

- DOORS for requirements management,
- Rational Rose for design modeling, and
- AUTOPLEX® test management system (TMS) for test management.

This decision was based on many factors, including prior experience and alignment with other projects in their business unit. Further, in order to contain the disruption to existing release schedules, deployment occurred in three phases:

1. A pilot phase to evaluate the potential for success with this approach,
2. A limited deployment in selected early-adopter product releases, and
3. A full rollout involving all releases for the six products owned by the department.

The first two phases of the deployment resulted in the creation of several DOORS-based requirements models containing use cases and text requirements using the process outlined below. During these two phases, tools were developed and tested to implement the transfer of DOORS requirements information into the Rational Rose use-case view and into AUTOPLEX®

TMS test scenarios. Early results from the requirements models built in the first two phases of deployment indicate that the requirements modeling approach will successfully feed requirements information directly into Rational Rose and AUTOPLEX® TMS. The automation will produce savings in the design and test development interval from reuse of the requirements scenarios, plus a reduction in product defects from more complete testing. As a result, the project management recently initiated the final full rollout phase.

To achieve success, the project team needed assistance to adopt certain elements of all three implementation levels of the requirements meta-model. We supported the project by performing the following tasks:

- Identifying specific best practices (elements of each implementation level) to be adopted (after examining their current processes and constraints),
- Developing tool-based templates and software for ensuring effective tool-usage and for automatically transferring information from one tool to another, and
- Delivering training to systems engineers, designers, testers, and their managers on using the tools, templates, and software to realize these best practices.

Additional details are provided in the following sections.

Implementing the Meta-Model in DOORS

Systems engineers needed to adopt the Telelogic DOORS tool as a requirements repository and to enable automated flow-through from requirements to design and test artifacts. Therefore, our team created a DOORS template for a requirements object model and a set of rules for writing structured use-case-based requirements that are machine parseable. In particular, each use case is defined in DOORS by creating a hierarchical set of DOORS objects containing the use-case title, preconditions, the steps of the main success scenario, the steps of the alternative success and failure scenarios, frequency information, and technology variations. The format is similar to the style of use cases in [2]. We also built a tool to check each use case to make sure that it met formatting rules that would make it machine parseable. The DOORS template also included a section for non-use-case requirements to hold text-based requirements such as the business rules.

Jump-Starting Design in Rational Rose

The designers had some exposure to Rational Rose, but we provided guidance on effectively using this tool and the requirements information to jump-start their design models. Therefore, our team delivered training on UML and object-oriented design practices using Rational Rose. In addition, our software automatically transfers each actor, use case, and scenario in the DOORS requirements model into the use-case view of a Rational Rose model. Because an intermediate Extensible Markup Language representation is used to effect this transformation, this approach can easily be extended to accommodate another design tool or another requirements management tool.

As a result of this work, designers have the functional requirements information available within Rational Rose, which keeps them from having to constantly refer to a requirements document. This functional requirements information is automatically available in a more formal and less ambiguous form—with use-case diagrams and UML sequence diagrams—rather than in text.

Furthermore, designers can create links from their design models to related requirements representations inside the Rational Rose model and produce reports on requirements coverage. In addition, developers are guided in the direction of "scenario-based design," i.e., they create within Rational Rose more elaborate design-level scenarios, which describe the responsibilities of the design-level classes that are employed to implement the use cases.

Jump-Starting Test Artifacts in AUTOPLEX® TMS

The testers were already using AUTOPLEX® TMS for test management. However, we assisted them in jump-starting test cases from requirements and changing their work practices to identify errors in requirements while still writing test cases. Therefore, we developed software to transform the structured-text version of the use case into a set of candidate test scenarios that the tester can choose to import into AUTOPLEX® TMS. The tool also creates a set of "one-line tests" for other (non-use-case) requirements for user-driven import into AUTOPLEX® TMS. The subset of tests selected by the test engineers is imported directly into the test tool. We also provided training and recommendations on how testers can use this tool to iteratively generate test cases and provide feedback to systems engineering on faults and gaps in requirements while these are still being engineered.

Test engineers thus avoid many errors in transferring scenarios because each step of the test scenario comes directly from a requirements-level use case. In addition, the tool presents implicit scenarios that a tester might not ordinarily create from a normal text-based requirements document. This can significantly improve the quality of test coverage. The test engineer still needs to elaborate on the test scenario (by including details on user interface commands and test environment) to convert it into a real system test, but the process of importing test scenarios reduces the likelihood that a significant scenario will be overlooked in the test development process.

Conclusions and Next Steps

We have described a meta-model for requirements information, and we advocate a model-based approach to requirements engineering. Such a model captures not only the requirements objects but also the relationship between these requirements objects.

Our experiences show that adopting such an approach can lead to interval reduction through jump-starting design and testing artifacts. In addition, the management of the requirements model using a repository-based CASE tool can aid projects in managing requirements churn.

We were, however, unable to implement all aspects of the requirements meta-model in our first project. Due to the need to focus on tangible results in the short term, we were unable to implement tool-based artifact flow of a complete requirements model. Instead, we focused on use cases and business rules. Since we believe that architectural requirements and operational profiles are crucial for ensuring product maintainability and customer satisfaction over the long term, we would like to extend our work to address tool-based representation and automated transfer to design and test requirements information in these dimensions. In addition, we would like to broaden our work on requirements traceability in design and test artifacts by automating the establishment of cross-functional traceability links and automating notifications of changes in requirements and their impact to stakeholders. This will enable projects to speedily and effectively respond to requirements churn.

Trademarks

References

[1] L. Chung, E. Yu, B. Nixon, and J. Mylopoulous, Non-Functional Requirements in Software Engineering, Kluwer Academic Publishers, New York,1999.

[2] A. Cockburn, Writing Effective Use Cases, Addison-Wesley, Reading, MA, 2000.

[3] L. Constantine and L. Lockwood, Software For Use, Addison-Wesley, Reading, MA, 1999.

[4] P. Kruchten, The Rational Unified Process: An Introduction, second edition, Addison-Wesley, Reading, MA, 2000.

[5] J. Musa, Software Reliability Engineering, McGraw-Hill, New York, 1998.

Abbreviations, Acronyms, and Terms

CASE— computer-aided software engineering
ID—identification
TMS—test management system
UML— Unified Modeling Language
XML— Extensible Markup Language

This paper appeared in the Fall 2003 issue of Bell Labs Technical Journal (vol. 8, no. 3)