New techniques for requirements management using Rational RequisitePro, Part 1: Using architectural methods to analyze, manage, and trace business requirements

Kumar Mani
February 06, 2007

Discover new techniques to capture, manage, and trace architectural requirements in this series from Kumar Mani. The techniques are based in architecture theory and are applicable to all IT projects. If you're an IT architect or project manager who faces complex requirements in enterprise or integration projects, you can use these techniques to manage the project and help to ensure that it is delivered on time. This article explores an implementation that uses the IBM® Rational® toolset, but it can be replicated just as easily using other products.

View more content in this series

Introduction

This article describes new architectural methods and techniques that focus on requirements engineering, the science of capturing, analyzing, and managing requirements throughout a project's life cycle. Although this article uses the Rational toolset to illustrate the techniques, it is not intended as a tutorial for these products. The goal is for you to use the underlying architectural techniques and apply them to your projects. IT architects familiar with Rational will, of course, be able to replicate these techniques readily in their projects.

Requirements are important because they form the centerpiece for developing architectures. Figure 1 shows The Open Group Architecture Framework Architecture Development Method (TOGAF ADM) and its eight phases.
Requirements, as manifestations of a need, opportunity, or deficiency, drive the other phases throughout the process. Yet requirements management is often overlooked or not addressed adequately in projects for many different reasons. IT architects often ask:

- "I've been using Rational products for modeling and development, but how can I link them to the requirements?"
- "We already have XYZ tool with comparable features, so what's better here?"

IT specialists would like their technical requirements clearly stated so they can start coding. Project managers and customers are not clear about the benefits of tools like Rational RequisitePro® and their applicability with project management tools such as Rational Portfolio Manager. (Some projects use the entire Rational toolset except RequisitePro!) These are valid questions that are addressed in this article.

Tools such as Rational, when used properly, can provide significant benefits. Some project managers use tools (such as RequisitePro) as repositories for requirements; that is, the requirements are imported as is into the tool and reports are published from it. Project managers are left wondering why they don't see any tangible benefits.

This article aims to bridge the gap between the product literature and the process literature. The Rational tool documentation is excellent in describing its features and capabilities; the process literature abounds with methods and best practices. However, IT architects still face a gap in how to maximize the tools' benefits. In this article you learn about business requirements, use cases, and basic traceability, followed by system-level requirements, componentization, and traceability during testing.
Empire Systems Corporation

To demonstrate how to best use architectural requirements techniques, this series of articles will refer to specific, hypothetical examples from Empire Systems Corporation (ESC), a fictional manufacturer and supplier of personal computers, laptops, computer parts, and related hardware, such as Web cameras and microphones. ESC already has a Web presence and has Web enabled many of its applications. Now the company's leaders want to move the enterprise to the next level by streamlining processes, automating business capabilities, adopting an enterprise architecture, and ultimately, improving revenue and profitability.

Business requirements

Successful IT projects begin with good business requirements. Technical literature is filled with methods, techniques, and best practices for requirements engineering. However, the discussion about requirements, particularly business requirements, is often vague, rambling, and, in general, difficult to grasp. This lack of clarity affects how the business requirements are interpreted, and subsequently mapped, to technical requirements. Let's recap a few basic principles of requirements engineering.

Focus on business

Business requirements should focus on actual requirements of the business and be distinguished from other business concepts (such as vision, mission, goal, or objectives). A common error is to state a business goal (for example, "to achieve 20% cost reduction") as a business requirement.

Be SMART

Requirements must **Specific**, **Measurable**, **Actionable**, **Realistic** and **Timebound**. This is more challenging than it looks. It can be done easily, though, by asking the following questions:

- Why does it need to be done this way?
- What result will you get or be able to do if this problem is solved?
- What process, specifically, suffers from this drawback?
- What benefit (as quantifiable as possible) is achieved by amending the process?
- What can be done to change things?
  
  This is usually hard to answer, and stakeholders tend to drift back to the problem statement. The requirements engineer can start with some *what if* suggestions to get some possibilities flowing. It is important, though, to back off as soon as the stakeholders get back on track.

- When, pertaining to timeframe, should this be achieved with respect to context and business environment?
  
  This is not intended to be a launch pad for project delivery timelines; rather, it is a starting point for reasoning about the **business** context. For example, an online card merchant may want to target the holidays (their business context) to debut their new Web site.

Identify and clarify scope

What is **in** and what is **out**? What is to be performed or delivered, and by what time?
Identify the "subject" and "verb"

This is most often overlooked and is the primary cause of scope creep. If there is more than one verb, then consider stating it as a distinct requirement. This improves clarity and promotes better understanding among the stakeholders.

Keep to what, not how

Business requirements always state what is to be done and should avoid any hint toward how it could be done. For example, an enterprise may want subjective, integrated views into their business data. The focus, then, should be on what they want to do, rather than on what they need to get (for example, data warehouse).

Requirements example

ESC has several Web-enabled applications. Because each application has evolved independently, customers face issues like having to sign into each application independently and working with different views for each business function, causing complaints. In response, the ESC business team has framed the following requirement:

**BR264:** Customers should be able to access all ESC applications through a consistent interface and sign in only once. ESCWeb (the primary commerce Web site) will have the same look and feel for all customers. This will reduce user error and improve the user experience. ESCWeb will eventually also support mobile and remote customers.

Clearly, this requirement can be improved. By applying the principles, we can restate it as follows.

**BR264:** In ESC5.0, customers will be able to sign-in once for all ESC applications, including ESCWeb, ESCOrderStatus, ESCVendor and ESCSupport.

**BR265:** In ESC5.0, all ESC applications will implement ESC Web Standard 273-1 and 273-2, which will lead to 10% fewer user errors.

Technique 1. Identify business requirements in RequisitePro

In this technique for recording the requirement in RequisitePro, the key is to retain the discipline used in capturing and clarifying the requirement. This technique involves three steps:

1. Define a **Requirement Type** for business requirements. All business requirements will acquire this type. **BUS** is a customary prefix.
2. While defining the Requirement Type, use the **Requirement Must Contain** option to specify a delimiter (such as "|"). The subject and verb can be placed on either side of the delimiter. This is not strictly enforced; the RequisitePro manual has more details. The idea is to gently induce discipline in identifying the subject-verb into requirements.
3. Create a package for business requirements at the top level. You see how this helps traceability in a moment.

Technique 1 is now complete. You've captured a clear and concise business requirement and are ready to move to the next phase. **Figure 2** shows this technique.
Use cases

Technique 2 (Connecting use cases to requirements) covers the next phase of the project. As soon as the business requirements are defined, the business architect or analyst develops use cases to elucidate them. Especially on large or complex projects, business analysts often run into two problems:

- Requirements reside in documents (unless you use Technique 1) and use cases live in models, such as Rational Rose® Enterprise (Rose).
- This disconnect becomes difficult to manage as the number of requirements (and use cases) grows.

How are business requirements connected to use cases? Use cases capture the user view; they specify the actions of the user and the responses of the system. But business requirements go beyond serving as the origin for use cases. When specified as functional requirements, they drive the content of use cases. When specified as qualities or constraints, they become attributes (or alternate flows) of use cases. We should be able to record the natural connection between a business requirement and a use case and do meaningful things with it, such as trace or analysis.

There is a wealth of material on modeling use cases. The articles in Resources are a good starting point. Reviewing use cases is outside the scope of this article, but we will introduce a naming convention that applies to the implementation of the technique.

Naming principle for use cases

Why does the name of a use case matter? Recall that we introduced the subject-verb structure to define requirements. The notion is extended here. The subject is represented by actors in the use case model. Our use case naming principle states that use cases should be named with the present tense or the gerund (ing) form of the verb. In some cases, it helps to include the name of the object modified by the verb.
Technique 2. Connect use cases to requirements

Now it's time to define use cases and connect them to the requirements. The goals of this technique are to maintain clarity in defining use cases, and to keep them connected (traced) to the requirements, using the following steps.

1. If your RequisitePro project does not contain a package for use cases or a requirement type for use cases, then create it.
2. Associate the Rose model with the RequisitePro project using the Associate Model to RequisitePro project dialog box. Ensure that the Default Requirement Type is set to Use Case.
3. Now you can begin creating your models in Rose. For each use case, first create a RequisitePro requirement of type Use Case with the name compliant with the naming principle, as shown in Figure 3. Enter this name in the Requirement Text.

Figure 3. Use case naming principle

4. Create the use case in Rose. Right-click the use case to get the Associate Requirement to Use Case dialog. Select the Use Case in RequisitePro and click OK. This will bring up the Resolve Use Case Name dialog, which is important because it connects use cases to requirements using the verb, as shown in Figure 4.
5. You will now see the Requirement Properties dialog. Select the Traceability tab, and trace the use case back to the BUS requirement.

This technique is now completed. You started by naming the use case in RequisitePro, which enabled you to define, elaborate, and trace the use case entirely within Rose. This highlights a big advantage in Rational: the integration between requirements, modeling, and design.

**Traceability**

Why is traceability important? **Traceability** is the ability to follow the life of a requirement from its origin through its various incarnations, both forward and backward. As enterprises evolve, so do the systems (and their requirements) that support them. Traceability is the critical link that connects the past, present, and future of requirements. Traceability reports provide the data upon which various project analyses, such as cost, coverage, and impact can be performed.

Traceability is hard to achieve in practice. The main problem is that traceability is viewed as an ancillary aspect of requirements engineering. The requirements are defined in documents and the models are built in Rose. Traceability is recorded, often manually, in spreadsheets, after the modeling work is done. Maintaining the spreadsheet is cumbersome and prone to error. But what hurts the project more is that the spreadsheet itself serves as the traceability report and thus escapes scrutiny.

Our next technique solves this problem. The first part of the technique is to induce discipline in tracing requirements when they are created. (We demonstrated this in the previous technique with use cases.) The idea is to maintain discipline throughout the requirements cycle and into test. The second part is to generate coverage and dangler reports. The dangler report is the first level of impact analysis. The central idea is that these reports, generated automatically, call for scrutiny during reviews.

Coverage, as the name implies, ensures that every requirement at one level is covered (or further specified) at the next level. Every use case must be covered by a test case, for example. Anglers
are requirements that have crept in, unintentionally, at one level with no precedent at the previous level. It is important to catch both of them early in the cycle because a warning at this stage is much easier to tackle than a bug report after the system is delivered. It is equally important to automate these functions.

**Technique 3. Trace coverage and danglers**

This technique shows how to build Coverage and Dangler views to find gaps between the business requirements and the use cases. These views fit in the standard RequisitePro view framework. Use the following steps.

1. From **Coverage** in the business requirements package, select a new **View**, where the **View Type** is a traceability matrix, the **Row Requirement Type** is BUS, and the **Column Requirement Type** is UC. Note how we reuse the BUS package from **Technique 1**.
2. In the **View Properties** box, create a **Query** named Business Requirements Coverage on the row requirements. Now **Add** the **Traced-from** attribute to the **Query**.
3. This will bring up the Query Requirements dialog. As shown in **Figure 5**, select **Not Traced** and check **Direct links only**.

**Figure 5. Trace coverage**

4. Now refresh the **View**. You will see the business requirements that have no use cases associated with them. You can create a full report **View** by omitting steps 2 and 3.
5. **Dangler** views are created by reversing the **Query** criteria in this technique. We start with the column requirement type (use cases) and select **Traced-to** BUS requirements. This view displays all the dangling use cases.

This technique is now completed. You traced use cases back to the originating business requirement and eliminated spurious use cases and incomplete business requirements. Your project can now move safely into the solution realm.

**Summary**

In this article, you investigated the fundamentals of requirements engineering and introduced three new techniques for managing architectural requirements. It reviewed the basic principles of
requirements and described three techniques for managing business requirements, use cases, and traceability.

We are, however, still in the problem domain. Stay tuned for Part 2, which will enter the solution domain and the various stages of solution development (from an architectural perspective) and explore new techniques on managing solution delivery.

© Copyright IBM Corporation 2007
Trademark
(www.ibm.com/developerworks/ibm/trademarks/)