

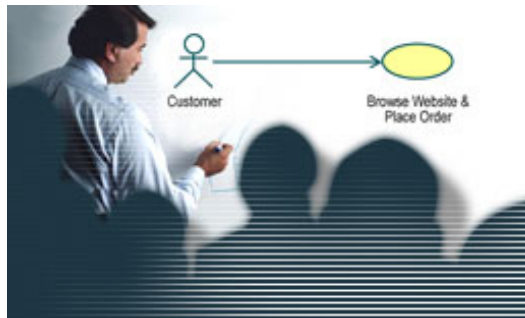
Adopting use cases

Part I: Understanding types of use cases and artifacts

by **Pan-Wei Ng**

Software Engineering Specialist
Rational Software Singapore
IBM Software Group

The use-case technique is an increasingly popular approach to capturing requirements and driving system development. Among the challenges new adopters of this technique face are how to introduce the technique into an organization and how to determine when use cases are complete. Typically, they must address these challenges under real project pressures. The goal of this article series is to outline principles that will help them overcome these challenges. In Part I, we will examine different types of use cases and artifacts, and talk briefly about how to introduce use-case techniques to a team that is unfamiliar with them.



In this series, we will make our discussions concrete through a hypothetical case study in which an aspiring analyst and other members of her CATLYST project team begin a new journey with use cases. Those of you who read ["The True Story of a New RUP Manager"](#) in the January 2003 issue of The Rational Edge will recognize the players on our fictional team. Part II will trace the execution of the project and highlight how the principles were applied.

Smith, a veteran project manager who has just successfully delivered the project REALITY-J, is sitting in his cubicle when one of the more senior team members, Harriet, raps gently on the steel frame of his doorway. Harriet has been assigned to another project as an analyst and needs help in gathering the project's requirements, particularly using use cases. Knowing that Smith has had practical experience with use cases, she has come to him for advice.

- ▶ [subscribe](#)
- ▶ [contact us](#)
- ▶ [submit an article](#)
- ▶ [rational.com](#)
- ▶ [issue contents](#)
- ▶ [archives](#)
- ▶ [mission statement](#)
- ▶ [editorial staff](#)

"We're about to start a new project called CATALYST," she explained. "Its goal is to provide value-added services for an international hotel chain and to solve their billing problems. Our team has different levels of exposure to use-case techniques. Can you give me some tips on how to proceed?" Harriet asked.

Requirements leadership is critical

"Is there a lead analyst on your team?" Smith asked.

"What do you mean by lead analyst?" Harriet replied.

"If your team members have different ideas about how to conduct the requirements management process and different ways of documenting and organizing the requirements, who will have the final say?" Smith asked.

"I don't know. I don't think *I'm* functioning as a lead analyst. I have only seen use cases applied when I was on REALITY-J, but you wrote most of them. I was just a consumer of your use cases. We have currently three working members on our team: Roland, Helen, and myself. We'll all be assuming the analyst role at project start, then subsequently taking team leader roles. Roland said he has some experience with use cases. Helen has no experience with use cases, but she is actively reading on the subject, and so am I. Simon is our project manager. I guess, if there are any differences among us, he would be the one to decide," Harriet replied.

"Would Simon be actively involved in the requirements gathering -- identifying use cases, outlining them, and so forth?" Smith asked.

"I don't think so. He's likely to be quite busy with other aspects of the project and is also fairly new to use cases. We'll probably do the requirements, and he'll likely be the one to come up with the project schedule, and so on," said Harriet.

"So, Simon won't have time to do the work of a lead analyst," Smith said.

"This is a problem. Without an effective lead analyst on your team, everyone is at risk. The first thing your team must do is identify a lead analyst," Smith admonished. He pointed out that the IBM Rational Unified Process[®] or RUP[®] makes an explicit distinction between two roles involved in the requirements gathering process and showed Harriet the following descriptions in RUP:

- **System analyst.** The system analyst role leads and coordinates requirements elicitation and use-case modeling by outlining the system's functionality and delimiting the system; for example, establishing what actors and use cases exist, and how they interact.
- **Requirements specifier.** The requirements specifier role details the specification of a part of the system's functionality by describing the requirements aspect of one or several use cases and other supporting software requirements. The requirements specifier may also be responsible for a use-case package, and maintains the integrity of that package.¹

"In short, the system analyst owns the big picture, and the requirements specifier works on the details," Smith explained, noting that Harriet's project had neither a system analyst (as RUP defined it) nor a lead analyst (as he defined it). Her team needed a leader not only to coordinate the team, but also to ensure consistency by formulating requirements guidelines. Otherwise, one requirements specifier might mistakenly think that another was documenting requirements for a certain area, and crucial requirements areas might slip through the cracks. The lead analyst, Smith emphasized, plays a critical role in bridging gaps and ensuring the completeness of requirements.

"The user representatives also need a leader. Otherwise, you'll find yourselves in endless debates and stalled decisions that will delay the requirements gathering process," he continued. "Tell the end-user team to identify such a person before you begin work. A successful project demands strong leadership on both the project side and the end-user side.

Requirements are a means to an end

Smith smiled at Helen, who, upon overhearing the discussion, had joined Harriet in his cube.

Knowing that Harriet was a perfectionist who liked things spelled out to the smallest detail, he wanted to help her adopt a balanced perspective when it came to managing requirements. "She has to avoid focusing on documentation for its own sake and stay focused instead on understanding the problem and gaining consensus on how to solve it," he thought to himself. So next he drew three overlapping circles and marked regions to represent what the customer needed, what would be captured as requirements, and the system that would eventually be delivered (see Figure 1).

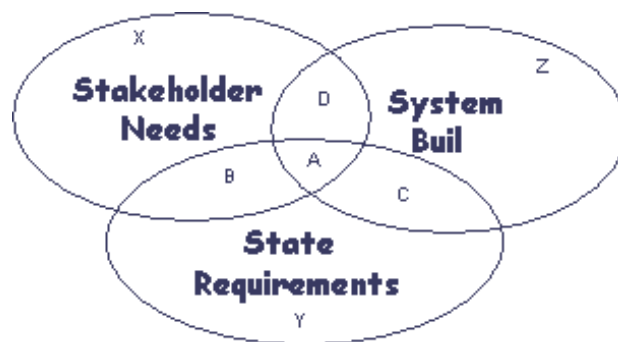


Figure 1: Capturing requirements effectively

"These three circles represent the framework you need to track project progress," Smith said, and went on to explain each of them as follows.

- **Stakeholder needs and goals.** Systems are built to meet certain stakeholder needs or goals; these define what the system is supposed to do.

- **Stated requirements.** During requirements gathering, stakeholder needs and goals are refined into requirements.
- **System built.** The system is built to conform to the stated requirements, and validated against stakeholder needs and goals. This closes the loop.

"Never lose sight of the fact that requirements are only a means to an end. The end goal is to have a useful working system that meets stakeholder needs and goals," Smith told Harriet. Then he added letters to each area within the requirements circles, as shown in Figure 1.

"All right, let's try a little quiz. In which of these lettered parts does action take place?" Smith asked.

"Definitely in part A," Harriet replied.

"Yes, A is the set of stakeholder needs that were identified and for which requirements were written, and that represents the part of the system that has been built and validated," Smith agreed.

"I think action takes place in part D, too," Helen suggested.

"Precisely! If a system meets stakeholder goals, it doesn't matter whether you've written down the requirements. In rare instances, when everyone has an extremely strong understanding of the project goals, there is no need to state requirements -- or the requirements specifications can be minimal."

This really got Harriet thinking. "You mean in some instances we can forget about requirements?" she said.

"Absolutely not! But remember I said that requirements are means to an end, which is a useful system," Smith said. "The main purpose of requirements is to bridge the gap between the stakeholders' perspective and ours, especially in areas we don't understand or disagree about."

"I'm still not sure I get it. Does that mean we should have more meetings and fewer documents?" Harriet asked.

"You should hold meetings to gain consensus, and use documents to keep track of what has been agreed to and what is still outstanding," Smith replied.

Choosing appropriate techniques and artifacts

"So the whole idea of requirements is to have continual agreement between the stakeholders and us. How does the technique of use cases apply here?" Helen asked.

"Identifying business actors, business workers, and business use cases, as well as system actors and use cases, helps us clarify the goals and scope of the system, and its role in meeting business objectives. Use-case

specifications help us clarify the interactions between the actors and the system," Smith replied.

"The key problem with the traditional approach of expressing requirements in terms of 'The system shall' statements is that these statements don't translate directly to acceptance tests; that requires an additional thought processes. *Use cases bridge this gap*," he said emphatically. He went on to explain that use-case flows of events describe actor requests and system actions (e.g., to display processing results or modify a system state), which are useful in formulating test steps and verification points when working on test procedures. Using system acceptance criteria as a basis for eliciting and documenting requirements in the early stages gives team members a great deal of control.

Requirements artifacts

"Our system has different kinds of requirements: user interfaces, business processes, infrastructure requirements, data requirements, and interfacing requirements. How do we capture these in use cases?" Harriet asked.

Smith replied by describing a few key artifacts for capturing different kinds of requirements² in addition to use cases, as depicted in Figure 2.

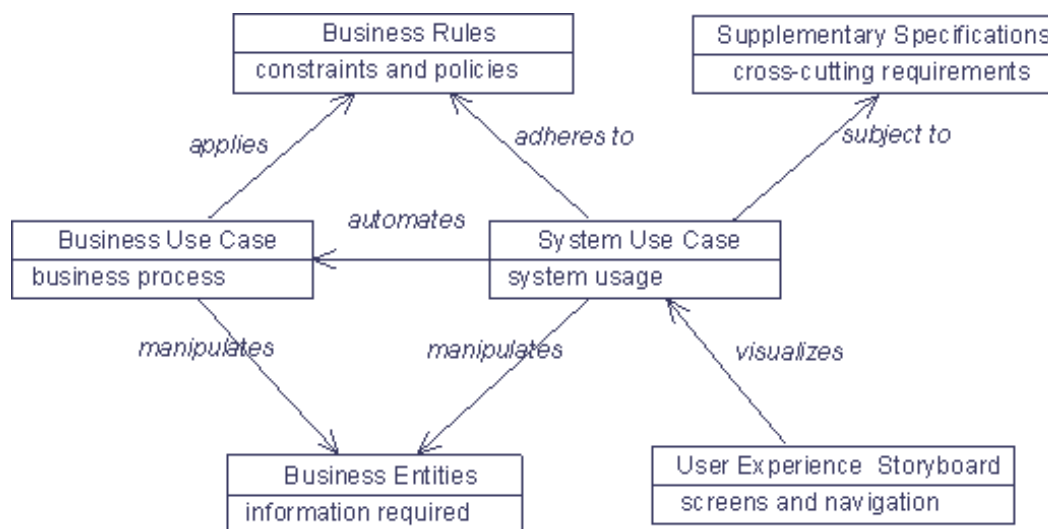


Figure 2: Summary of requirements artifacts

- Business use-case model.** Frequently, the objective of the desired system is to solve business problems or exploit business opportunities by offering value added services. Business use cases extend the concept of use cases to describe business processes. The business use-case model (together with the business use-case specifications) provides a way to evaluate the scope of the desired system -- which parts are automated, which parts are not, which parts are addressed by changing the business process. This allows us to evaluate the completeness of the use-case model from a business perspective, as each system use case must support one or more business use cases.
- Business entities and domain model.** Most systems need to manipulate and present business information. A business entity

represents a group of related information fields as classes. Business entities are handled and manipulated by a business process (i.e., business use cases), which are subsequently automated via system use cases. The sum of all the business entities and their relationships constitute the domain model, which describes the problem domain. Each system use case will manipulate some entities, and the entities are usually involved in multiple system use cases.

- **Business rules.** The complexity of systems today is typically the result of complex business rules that the system must conform to. Business rules are referred to by both business and system use cases, and can be in the form of decision tables, computation rules, decision trees, temporal diagrams (describing what events must occur before or after what other events, and the process resulting from that), algorithms, and so on. Describing business rules in the flows of the use cases usually clutters the use-case specifications. Hence, they are normally captured in separate artifacts or as annexes to the use-case specifications.
- **User-experience model and storyboards.** User-experience modeling is a convenient way to capture user interface requirements without resorting to drawing screen layouts that may take significant effort and yet are very likely to change. User-experience modeling abstracts actual user interface screens as a UML class, which has the stereotype «screen». Attributes identify what the user can see on a screen; operations identify what the user can do on each screen; and associations identify navigation paths. User-experience storyboards are subsets of the user-experience model used to describe the screens involved with system use cases.
- **Supplementary specifications.** Supplementary specifications describe requirements that affect multiple use cases. For example, all use cases are subject to access control, audit trail, personalization, and so on. Supplementary requirements are usually technical in nature and can pertain to functionality, usability, reliability, performance, and supportability. They are usually expressed as declarative statements in the form of "the system shall do ..."

Types of use cases

"How do I know which one of these artifacts I should use?" Harriet asked.

"Well, as Chairman Mao said: 'Black cat or white cat -- makes no difference as long as it catches the mice.' As long as the technique does the job, it is fine," Smith replied.

"I understand what you mean, although I believe it was the late Deng Xiaoping who said that," said Harriet. "But I still need some general guidelines."

"Let's begin by looking at different types of use cases," Smith replied, and he proceeded to list those shown in Table 1.

Table 1: Types of use cases

Use Case Type	Description
Data maintenance use cases	These are typical Create, Read, Update, Delete (CRUD) use cases. Writing one per entity is repetitive: Normally it is easier to write such use cases generically so they can be applied to multiple entities.
Request approval use cases	These are use cases for routing requests through a series of approval stages. Some approval stages can be skipped or combined, depending on the nature of the request. Each stage in the approval adds a little information about the request.
Data analysis use cases	These use cases are for analyzing transactions on entities that are manipulated by other use cases. For example, one might want to analyze the room booking trends for a hotel chain.
Payment use cases	These use cases are for making payments. The complexity comes when there are multiple payment methods, such as cash, check, and credit card. Some payments can be made immediately (e.g., credit card payments), whereas others take time to clear (e.g., checks). If the payment is not good, the purchase may have to be made null. Moreover, payments may be required for a number of activities: booking a room, enrolling in a membership programs; and so forth. Computation of payment amount also involves a number of business rules, which might be configurable.
Loyalty program use cases	These use cases allow customers to accrue credits when using a service, and these credits can be used either as a payment method or as rewards for corporate gifts. Accruing of credits is normally an extension to some other use cases, whereas the redemption of the credits is a use case in itself.

"This is really helpful," said Harriet. "Will I need different use-case specification templates to create different artifacts?"

"You can use the same basic use-case specification template for all the artifacts if you supplement it with appropriate appendices," Smith answered. "For example, you can attach corresponding user-experience storyboards, class diagrams of participating entities, related business rules, and so on, as appendices to your use-case specifications. That doesn't mean that every use-case specification needs a full set of appendices; include only those that will promote understanding."

"So which artifacts are required for the use-case types in your list?" Harriet asked.

"I really want to refrain from making recommendations -- lest you treat them like the immutable word of God," Smith said. "It really depends on the project context. However, there are some obvious ones. For example, data maintenance use cases can be described very well by domain modeling, and possibly user experience modeling. I find domain modeling appropriate for data analysis use cases, since they are data-centric. Approval use cases can be supplemented with business use-case specifications if they are non-trivial. Payment use cases require quite a lot of business rules in many cases. Loyalty use cases are interesting because they insert themselves into existing use cases."

"I cannot answer your question completely," Smith went on. "Use-case types are more like use-case patterns -- design patterns -- but at the use-

case level. You will encounter different categories of use cases in your project, so select a representative use case from each category early on and work on it. That will help you decide what writing style and what appendices are required. If you'd like, I can help you identify your use-case patterns when the project starts."

Evaluating completeness and detail

"How will I know when I have finished my use cases?" Harriet asked.

"You have to evaluate the use-case model with respect to some frame of reference, such as business needs, business domain, and so on. However, your use cases will never really be complete until well after the project is over because your understanding -- and your end users' understanding -- of the system will be refined and will evolve over time. That's why you'll have to proceed incrementally and iteratively," Smith answered. "At first, you'll want to focus on whether each use case is sufficiently detailed for development to proceed."

"Yes, but how do I know if I have sufficient detail?" Harriet asked.

Smith replied by listing the following criteria:

- **Use-case specifications.** The basic flow and the alternative flows are clear and are understandable by the development and test teams, and have received approval from the end users.
- **Business use-case specifications.** The specifications are very clear on when the use case is invoked within the business process. Also, the team has verified that the outcomes are of value to the steps in the business process. In other words, the team has verified the flow of events in the use case against the business processes.
- **Business entities.** All the business entities that will be manipulated by the use case have been detailed, and their attributes and subcategorization have been defined. Subclasses are often used to evaluate the completeness of the flow of events and the business rules. For example, *booking* would be an entity for booking for a hotel room, but there are different types of bookings: advance; immediate; priority; and so on. These different types should be treated in alternate flows of events, and the business rules must also consider them, by computing room charges under different booking types, for example.
- **Business rules.** The business rules required to support the use case are clear. For example, if there is a use case to make a room booking, there must also be computation rules to determine the room charges.
- **User experience storyboard.** The screens required for user interactions defined by the use case are identified, including fields and navigation paths.
- **Supplementary specifications.** It is clear how supplementary specifications affect the use-case flow of events. For example, if the

use case requires user identification, it must be clear *when* the identification is requested and what authorization is required. For an audit trail, it has to be clear what needs to be tracked and under what conditions.

"Wow! That seems like a lot of work!" Harriet exclaimed.

"I did not say that specifying requirements is easy, but as I said before, requirements are a means to an end -- a useful and working system," said Smith. "Use a table (see Table 2) to decide which artifact makes most sense for your project. Then, determine how much detail you need for the use case and use the same table to track your progress in gathering the information you need. In each cell, indicate whether you have collected the artifact's content and validated it with the users."

Table 2: Tracking requirements gathering progress

Use Case	Flow of Events	User Experience Storyboard	Business Use Case Specifications	Business Entities	Business Rules	Supplementary Specifications
UC1	yes	yes	yes	yes	yes	no
UC2	yes	no	yes	yes	yes	yes

Introducing use case techniques iteratively

"This is such a lot to learn!" Helen exclaimed.

"I agree. My team and the user representatives aren't familiar with use cases, and picking and choosing techniques will not be easy," Harriet added. "I don't think we have much time for training, either."

"You're right, this is a complex undertaking. Let's see how you can tackle it," said Smith. He suggested the three-stage approach shown in Figure 3 and proceeded to explain each stage.

- Stage 1: Techniques workshop
- Stage 2: Hands-on mini iteration with samples and guidelines
- Stage 3: Use of advanced techniques

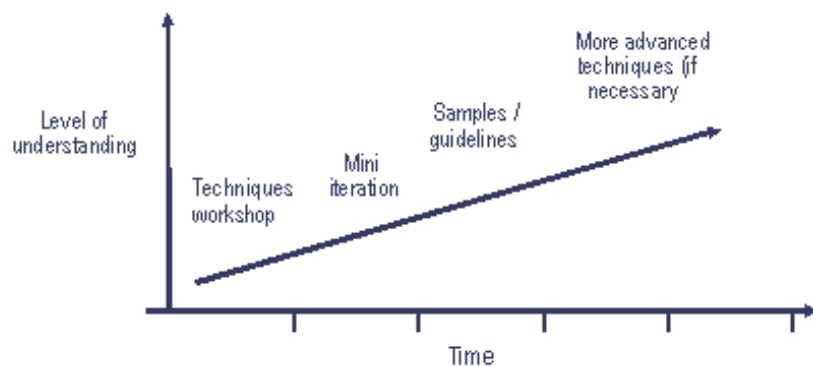


Figure 3: Introducing requirements techniques on a project

Techniques workshop

Most project teams have members with different levels of experience in using requirements techniques, especially use cases, so it is wise to conduct a workshop to establish a common understanding of the requirements techniques you plan to use. Both analysts and end-user representatives should attend, since both will be involved in authoring and reviewing the documented requirements.

The facilitator can be from either inside or outside the team, as long as he or she is a knowledgeable person whose professional skills are known and respected. It is important that the facilitator discuss techniques in the context of the current project; otherwise, the discussions will be too abstract. Supply project information in advance, even if he or she is not a member of the team.

Hands-on mini iteration

Following the techniques workshop, the project team should pick different types of use cases (see Table 1), then detail, implement, and test them in a mini iteration that attempts to resolve skills risks. This is extremely important when most members of the project team have little experience with use cases. By walking through the mini iteration, the project team will gain firsthand experience with important skills:

- Writing effective use cases.
- Determining additional artifacts required to supplement use cases.
- Understanding how use cases drive development (design and test).

The goal of the mini iteration is to quickly scale up the team's competency. If some team members have difficulty acquiring the skills, it will be necessary to prolong the facilitator's involvement or restructure team roles.

Because the goals are to nurture skills and help the team transition from a waterfall paradigm to an iterative one, the scenario for the mini iteration should be simple, so that the team can move quickly from requirements to design and then on to code and test. This small pilot project will help the team understand how to execute an iteration and use the appropriate artifacts.

Use advanced techniques when necessary

The workshop and mini iteration should cover a variety of use case types: datacentric, workflowcentric, data maintenance, reporting, and so on. This will allow team members to try out different writing styles and requirements patterns. At the end of the mini iteration, the team should review how requirements were organized and documented, and identify areas for improvement. This may require the introduction of more advanced techniques to structure the requirements through use-case notation:

«include», «extend», and so on.

Inexperienced teams may want to avoid using advanced techniques at the start of the project because these can spark methodology debates and distract the team from their primary goal of gathering and capturing requirements. It is best to get the requirements down on paper first, before seeking to improve them.

By the end of the mini iteration, the team will have gained hands-on experience with use cases, and they'll also have a good understanding of the project requirements. The advanced techniques will help them structure their requirements in a way that maximizes understandability of their use cases, and reduces repetition in use-case writing. The facilitator should recommend which techniques to use and provide the necessary guidance on how to do so.

Next month, in Part II, we will look in again on the CATALYST project team to see what happened when they began putting use cases to work.

Notes

¹ From Rational Unified Process, v2002.

² For more information on capturing business requirements, see "Effective Business Modeling with UML: Describing Business Use Cases and Realizations," in the November 2002 issue of *The Rational Edge*:

http://www.therationaledge.com/content/nov_02/t_businessModelingUML_pn.jsp



For more information on the products or services discussed in this article, please click [here](#) and follow the instructions provided. Thank you!

Copyright **Rational Software 2003** | [Privacy/Legal Information](#)