Adopting use cases
Part II: Putting learning into practice

by Pan-Wei Ng
Software Engineering Specialist
Rational Software Singapore
IBM Software Group

Last month, in Part I of this series, we introduced a hypothetical case study in which Harriet, an aspiring analyst, along with her team members on the CATALYST project, began a journey with use cases. Through her discussions with Smith, an experienced project manager, we learned about types of use cases and related artifacts. We also received some pointers on how to introduce the use-case technique into an organization and how to determine when use cases are complete. Now, in Part II, we will trace the execution of the CATALYST project that Harriet is working on and highlight how her team members applied the principles Smith recommended.

Smith to the rescue (week 5/40)

Four weeks had gone by since Harriet and Helen had discussed requirements with Smith, and he had not heard back from them. Wondering what was happening with the project, he decided to give Harriet a call.

"Things are not going well," Harriet reported. "The member of our analyst team who was supposed to be quite well versed in use cases is throwing the rest of the team and the users into confusion by turning all requirements into use cases. We now have hundreds of use cases, and when we attempt to detail them, we have a hard time identifying the steps."

"What is this person's background?" Smith asked.

"Roland is a contractor for this project. He has a postgraduate degree, and took one course on object oriented analysis and design, which he did very well in. He graduated only recently, but he'd acquired quite a lot of industry experience before going back for his master's degree," Harriet explained. "He has been the most vocal person in our group and led the user sessions. But the user representatives who had some exposure to use cases knew that what Roland was doing was not correct. This was really detrimental to our image as analysts. But we couldn't refute him in front of those
"representatives – that would have made matters worse!"

"Maybe I can help," said Smith. "Tell me what he was recommending."

"Functional decomposition, for one thing," said Harriet.

"Ahh, that's a common and sometimes deadly error," said Smith. In fact, I'm compiling a series of tips on project management, and avoiding functional decomposition is first among them. Let me send you the tip," he added, and e-mailed the text below to Harriet.

**Avoid functional decomposition**

Functional decomposition is breaking a problem down into small, isolated parts. The parts work together to provide the functionality of the system, but often do not make sense in isolation. Creating use cases is not the same thing as functional decomposition. Use cases define the context in which the functions are used. Having hundreds of use cases is a sure sign that you are using functional decomposition. With that many use cases, each of them is so small that it may have only one step, and the alternate flows tend to be extremely repetitive. For find out more about this, read "Thoughts on Functional Decomposition" by Murray Cantor and "Why Use Cases Are Not Functions'" by Kurt Bittner, in the April 2003 and December 2002 issues of *The Rational Edge*, respectively.

"OK, now let me ask another question: Did you hold a techniques workshop to synchronize everyone's understanding of use cases and requirements?" Smith continued.

"I sent a request for one to Simon, the project manager, but he said we didn't have time. He thought Roland was experienced in these things, so he made Roland the lead analyst and left the educating up to him. In truth, Simon doesn't understand the concept of use cases well enough to comment on what Roland is doing. He was eager to transfer responsibility for requirements elicitation to us and has taken quite a hands-off approach. But now he is really getting pressure from user representatives because things are not turning out well," Harriet replied.

"Yes, just appointing a lead analyst doesn't necessarily mean you've fulfilled that role," Smith agreed. "The responsibilities have to be performed effectively. I'll send you another tip about that." And he sent Harriet the following:

**Have strong leadership**

Strong leadership is a prerequisite for project success. A leader is different from a manager. A leader not only commands respect but also inspires action. Leaders need not be technical experts, although they must have minimal proficiency with the technology (e.g., a good understanding of use cases) so they can determine whether an assigned task has been completed correctly.

Inexperienced project managers often take a team member's word to confirm task completion, but this method is not reliable. If a project manager lacks the expertise to correctly evaluate the team's work, then a facilitator is required. This is common practice for projects staffed by first-time users of new technology or methodologies. Then, it becomes the project manager's duty to ensure that the facilitator is capable.
"I agree with everything you're saying here, Smith, but this project clearly does not have strong leadership, and a lot of damage has already been done," said Harriet, an edge of desperation in her voice. "The user representatives are now considering abandoning use cases in favor of a traditional requirements list and user-interface drawings," she added.

"I think the real problem is that the requirements elicitation sessions did not go well, and the end users are treating the requirements capture technique as a scapegoat. Typically, teams use a combination of techniques for requirements. A list alone is usually insufficient, and user-interface prototyping typically takes too long. User-interface drawings take significantly more time than user-experience storyboards. If you're committed to using use cases, you have to give it some time. It's a worthwhile investment. I'll send you some advice on this, too," said Smith, and e-mailed Harriet the following:

**Give your team time to learn**

Learning a new technology or methodology requires patience. Use cases force a team to explore different scenarios and how they are handled, something that they normally ignore when using a traditional requirements listing approach. There's no question that use cases promote better requirements, but they do demand greater effort, especially from new users. Teams sometimes grow impatient because of the time required to explore scenarios, and may blame the technique for lack of progress or even abandon it altogether. This is not prudent, because sooner or later the team will have to explore the use-case scenarios -- if not during the project's early stages, then during testing. Doing it early can help them reduce risks and create a much sounder system. If a team is using use cases for the first time, it is wise to get commitment from upper management and enlist their help in making the paradigm shift.

"Do you think it would still be useful to hold a techniques workshop now? We have already eaten up about 10 percent of the project schedule, and Simon is getting worried about the lack of progress," said Harriet.

"With so much confusion, a techniques workshop is even more crucial now," Smith replied. "The CATALYST team started on the wrong foot, and it will not be easy to set things right. Normally, in such situations, external intervention is necessary, as the user representatives have lost confidence in the project team. Gaining back their trust will require double effort."

**The techniques workshop (week 5/40)**

That afternoon, Simon called Smith. "We really need your help to conduct the techniques workshop you mentioned to Harriet," Simon said. "I have spoken to Jones, and he has agreed to let you do it." Smith hesitated for just a moment and then said, "OK, I'll get on it soon."

Not much later, Jones (Smith's senior manager) called Smith. "We'd better give Simon some help with the project," Jones said. "I'm trying to convince our organization to apply use cases and iterative development. This project must succeed. Grace is monitoring the project, so it had better be good. I'm sure that you can do it, after observing what you did for REALITY-J. May the RUP be with you."

Grace was the head of the project group, and Jones had been trying for some time to get her endorsement for a more systematic approach to software development.
Smith quickly arranged a meeting with the analysts from project CATALYST, and then he studied the requirements to make sure he understood them.

The following week (week 5 of the project), Smith held a workshop with user representatives and the analysts to discuss the principles of requirements management. He divided the attendees into various groups and let them work on identifying use cases in different business process areas. Before the workshop, Smith and Harriet had already identified critical business use cases and system use cases and worked out sample outlines for these, together with the initial domain model and storyboards, based on their knowledge of the project requirements.

During the workshop, Smith observed each group. Each was led by one of the analysts: Harriet, Helen, or Roland. Both Harriet and Helen were patient; they let the user representatives state their requirements, and they made sure everyone understood them. Both Harriet and Helen were also able to lead the discussion to explore the scenarios through role playing, and to work with the domain model, state charts, and use-case storyboards, as was appropriate.

Roland, on the other hand, quickly jumped to conclusions, and on several occasions Smith had to intervene so that the discussion could continue. It was apparent that Roland had trouble with requirements elicitation and the use-case technique. The user representatives, however, quickly caught on to the use-case concept and were able to explore the business and system scenarios. Roland eventually started acting like a scribe and simply took down what they said. This led Smith to fire off another bit of advice to the analyst team as soon as the workshop was over:

**An analyst is not an expensive scribe**

When it comes to the analyst capability (ACAP) measure defined in COCOMO II, the difference in project effectiveness between a very immature and a very mature analyst can be 100 percent. Effective analysts are more than just note takers; their role is to guide the user representatives and to uncover holes in the way they articulate their needs, goals, and requirements.

This is not to say that taking notes is not important. In fact, effective scribes are crucial to ensure that the discussion outcome (agreements, outstanding issues, rationale, etc.) is properly documented. Unless someone captures and publishes the outcome, people tend to forget what they say and what other people say, so they may revisit the same discussions unnecessarily.

Real analysts need to be more than scribes, though; they must function as detectives during requirements elicitation sessions. And they should know how to use techniques such as domain modeling, storyboards, and role-playing for exploring business and system usage scenarios.

The user representatives were pleased and satisfied with the workshop. Gary, the key user representative who reports directly to the project sponsor told Simon, "You should have done this on the first day; that way, we wouldn't have wasted four weeks."

**The law of diminishing returns (week 7/40)**

Simon had attempted to compress the requirements gathering process into the first twelve weeks of the project. The first four weeks were totally chaotic, but then Smith
conducted the three-day refresher workshop in the fifth week. That left eight weeks for Simon to catch up. Smith had suggested that they tidy up the use-case model survey and domain model survey, and then allocate the use cases that had to be detailed and developed across the project lifecycle. But Simon hadn't listened.

Instead, Simon had planned daily requirements sessions with the user representatives for the next eight weeks. The project team began intensive requirements gathering sessions, attempting to detail all the use cases and to supplement them with storyboards, a domain model, business rules, and so on, as appropriate. But the requirements for CATALYST were not simple. The user representatives had not thoroughly thought through the new business process areas, and consequently they were unsure how scenarios related to these processes would be handled.

Fortunately, the analyst team was committed to resolving this problem -- even Roland. Thanks to Smith’s coaching, all the analysts made sure the user representatives were quickly learning the requirements techniques -- which weren't really difficult -- and that they were able to move from one technique to another to explore the requirements. Here is what Smith had told them:

**Teach end users to become use-case users**

A typical concern for project teams attempting to apply use cases is whether end-user representatives will be able to read and understand them. But with a little instruction, end-user representatives can quickly become conversant with use cases. Unlike the development team, they do not need to grapple with technology implementation issues -- they do not need to code. Moreover, they generally have a better understanding of the problem domain because they live and operate in that space. Exploring alternative conditions and determining how to handle them is generally easier for them than for the development team. Moreover, I've observed a growing number of end-user representatives who have software project experience and understand the value of exploring scenarios early in the project. These people get used to reading and reviewing use cases very quickly -- even if they're first timers.

The first two weeks of requirements sessions after the refresher workshop proved to be very fruitful. However, the analyst team had to work late each evening to consolidate the requirements for that day and prepare for the next day. At seven weeks into the CATALYST project, the analyst team began losing steam: the late nights had taken their toll. Harriet had dark rings under her eyes, and Helen started to feel weak and had cramps. Roland's weight increased from 100 kg (220 pounds) to 110 kg (242 pounds). The user representatives were also losing stamina. It became increasingly harder to schedule requirements sessions because they had other business operations to attend to, and the daily requirements sessions had caused a significant backlog in their work. Moreover, a number of requirements entailed policy changes that the end-user representatives had to find time to discuss internally.

"This is getting tougher," Harriet told Smith. "Last week I woke up at 3 a.m. and couldn't sleep because I was dreaming about one of the users' business processes. Their attendance is dropping off, too. And when the key people are not around, it's difficult to make decisions; we're revisiting the same issues again and again."

"You're experiencing the law of diminishing returns," Smith pointed out. "So are the user representatives. This is a very typical effect of the waterfall approach to requirements. You simply cannot cram all the requirements-gathering sessions together. Requirements take time to think through and time to digest. If you force people to eat meal after meal without a break, they're bound to get indigestion."
"Well, I really do feel sick to my stomach," Harriet agreed. "Simon is still pushing for a requirements sign-off by week twelve, though. We need to work hard to meet that schedule."

"I don't think you're conceptualizing this in the right way," said Smith. "I'm going to send you some more advice." Here is what Smith sent to Harriet:

**Avoid diminishing returns**

*The Columbia Encyclopaedia, Sixth Edition, 2001,* states that you will experience diminishing returns if you increase one factor of production while keeping the others constant; overall returns will relatively decrease after a certain point. For example, if you add more and more laborers to harvest a wheat field, at some point each additional laborer will add relatively less output than his predecessor did, simply because he has less and less of the fixed amount of land to work. This principle, first thought to apply only to agriculture, was later accepted as an economic law underlying all productive enterprise. In the context of requirements elicitation, the first few sessions are usually extremely fruitful, as you collect new information daily. However, at some point, the information starts to dry up. No matter how hard a team tries, they will have little to add. Instead of continuing to elicit requirements, it is usually more prudent to take stock of the requirements you have (how many are captured, approved, outstanding, etc.). Then the team can determine which requirements are highest priority and stable; then they can proceed with iterative development for these requirements.

"How much more complete or detailed can your requirements become if you continue what you're doing now?" Smith asked.

"You're right. We won't make much progress," Harriet answered. "The user representatives have already told us most of what they can. They really need to sort out some internal issues. But they still want the sessions to go on -- to discuss every individual use case."

"Do you know why the user representatives want to go on?" Smith asked. "It is because you want them to do a formal sign-off! To have a formal sign-off (as the waterfall approach prescribes), they must include everything -- even beyond what is necessary to protect themselves. In the next few weeks, you'll definitely see the user representatives trying to increase the scope of the requirements, especially for areas they are unsure of. *That* is how they will protect themselves. I'm going to send you another tip about this."

Smith added, "I suggest that you quickly do a baseline of the requirements as they are today, or maybe at the end of the week. Then, start to development incrementally and iteratively for the stable parts, and give them ample time to sort out their internal issues in the other areas. Demonstrating a working system will boost their confidence and help you avoid scope creep."

"I think you're right," Harriet replied, "I'll speak to the team."

"Guess what I'm doing," said Smith.

"Sending me another tip?" asked Harriet.

"That's right," said Smith, and he sent her this:
Be cautious about formal requirements sign-offs

Requirements specified in a contract are generally high level and open to interpretation, so teams try to nail down more detailed requirements before proceeding with design and development. They have two options: 1) detail all requirements up front and get a formal sign-off, or 2) work out a use-case model and domain model, outline the critical use-case specifications (those targeted for the Elaboration phase), and iteratively detail them over the project lifecycle. The second approach is more favorable because its goals are to produce working software quickly, identify problems, and address them early in the development lifecycle.

If the end-user representatives would feel more secure with a formal sign-off up front, they can sign off on the use-case model survey and domain-model survey, because all use cases share entities. They can then sign off on individual use-case specifications following their respective iterations.

In the waterfall approach, payment milestones may be tied to the requirements review and design review at the beginning of the project. But the users' goal should be to pay for working software, not just documents. With iterative development, you can tie a payment milestone to the use-case model survey and the domain-model survey at the end of Inception. At this point, you will have allocated your use cases across the project lifecycle; you can tie payment milestones to iteration or phase exit criteria, such as allocated use cases, updated use-case specifications, or tested and working builds or releases.

Counting sheep (still week 7/40)

The next day, Simon invited Smith to join the CATALYST weekly analyst meeting. Smith repeated what he had told Harriet regarding the law of diminishing returns and iterative development. The analysts' faces brightened with agreement. Roland was very keen to move into development.

Simon asked, "So how should we proceed?"

"What is the status of your project?" Smith questioned.

"What do you mean?" Simon asked.

"I am referring to the table that tracks the use-case flow of events and other artifacts," Smith answered, referring to Table 1 (which we first saw in Part I).

<table>
<thead>
<tr>
<th>Use case</th>
<th>Flow of events</th>
<th>User experience storyboard</th>
<th>Business use-case specifications</th>
<th>Business entities</th>
<th>Business rules</th>
<th>Supplementary specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>UC1</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>UC2</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

"I have no idea," Simon said."Everyone was busy with requirements gathering, and no
one had time to track the progress."

"So how do you know if you're ahead or behind schedule?" Smith asked. "I'm going to give you a little tip on status assessment." Here is what Smith told Simon:

**Conduct regular status assessments**

It is not sufficient to assign tasks and deadlines to team members. The project manager needs to have a sense of whether those deadlines can be met, how much has been done, and how much more is left to be done. A task must not be like a long, dark tunnel marked by a start and an end. There must be intermediate visibility points for large tasks. If a task takes more than a week or two and does not have any intermediate checkpoints, that usually signifies a lack of understanding about the task (i.e., what is involved and the completion criteria). A project manager who understands what is involved in the task can potentially break it down and track status according to these smaller units.

In iterative development, there are broad plans (the software development plan and phase plans) that cross the project lifecycle, and detailed action plans (iteration plans) for the current and next iteration. The iteration plans should be of a finer granularity to facilitate checking.

Team members should keep the project manager updated about their status so the project manager doesn't have to chase them. If project status has not been updated for two weeks, it is usually a sign of poor coordination or lack of discipline. Automation can minimize the effort required for status updates and should be seamlessly integrated with daily activities for minimal intrusion.

Project teams that do not dutifully track status objectively tend to have delays. This is not surprising, because they have no means to highlight delays early in the project and steer the project accordingly.

Simon said little, but he instructed the analysts to submit their status the next day. "Let's not wait until tomorrow," Smith said."Let's do it now." He then asked each analyst how many use cases and business use cases they were assigned, how many they had discussed with the user representatives, and how many had user-experience storyboards, business entities, or supplementary requirements.

The results were tabulated in Table 2. The total columns indicate the project scope allocated to the analysts. The detailing status columns give the number of use cases that have been detailed with system use-case flows of events, business rules, entities, storyboards, or supplementary specifications. For example, Harriet had twenty-five use cases. She had detailed twelve of them, captured business rules for five of them, and worked out the storyboard for nine of them. But she had not yet considered how the supplementary specifications impact her use cases. Both Harriet and Helen diligently started examining their documents and counting. Roland was very quick to give answers.

| Table 2: Aggregated requirement status by analyst (absolutes) |
Smith looked at the figures and remarked, "I have yet to review your use-case specifications. At present, 105 use cases seems like quite a large number, but let's assume for now that all are reasonable. Harriet completed ten use-case specifications in two weeks. If things proceed at the same pace, she will require three more weeks to complete all her use-case specifications."

"How did you arrive at that?" Harriet asked.

"It's really simple arithmetic. Ten use cases take two weeks to detail, so you can do five per week. If you have fifteen more use cases, that will take three more weeks," Smith said. "Given that use cases are not of equal complexity, we can give or take a week of variation. So Harriet has two to four weeks to go."

The team immediately understood the usefulness of the progress metrics. Simon quickly said, "Well, it seems that our progress is quite good. Harriet, being the slowest, needed three more weeks, and we have six more weeks to go before requirements sign-off."

"Don't be too quick to jump to conclusions. Consider Harriet's business rules: She's captured them for only five use cases over the past two weeks. Extrapolate that for the remaining twenty use cases, and she would need another eight weeks." Smith cautioned. "But even if only half of them require business rules, she will still have about four weeks to go."

"If you look at Roland's numbers, he is four times faster than Harriet. This is suspicious," Smith said. "If you compare percentages, Harriet has 12/25 = 48 percent of her use cases detailed. Helen has 15/30 = 50 percent of her use cases detailed. Roland had 40/50 = 80 percent of his use cases detailed, which is a far greater percentage than Harriet's or Helen's," Smith said, "but this number is suspicious. Either there is an unequal distribution of work, Roland is extremely efficient, or the measurements are incorrect."

Smith examined Table 2 and started to normalize the counts for use-cases detailed against the use cases assigned to each analyst. The results are shown in Table 3.

<table>
<thead>
<tr>
<th>Analyst</th>
<th>Total</th>
<th>Consensus (Normalized)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Use Cases</td>
<td>Flows of Events</td>
</tr>
<tr>
<td>Harriet</td>
<td>25</td>
<td>48%</td>
</tr>
<tr>
<td>Helen</td>
<td>30</td>
<td>50%</td>
</tr>
<tr>
<td>Roland</td>
<td>50</td>
<td>80%</td>
</tr>
</tbody>
</table>
"What's more interesting is that Roland has 70 percent of the supplementary specifications, whereas Harriet and Helen have none," Smith noted.

"Roland, do you have any interfaces to external systems and any reports to generate?" Smith asked.

"Yes, plenty of them," Roland replied, beginning to feel annoyed.

"Are you clear on how you're going to interface to them? Have the user representatives clarified the report formats and fields, and the role of these interfaces?" Smith asked.

"Not really; we have yet to discuss them," Roland began to blush. He was uncomfortable with Smith's interrogation.

"OK, here's another lesson. When it comes to project metrics, you have to do a little digging to really understand them," said Smith, who began clicking the keyboard on his laptop.

"I can see it coming; I bet you're sending another tip," said Harriet.

"It's on its way now," replied Smith, who e-mailed the following to Simon and the analysts.

**Don't take project metrics at face value**

When project managers ask development team members about project status, they often get answers such as "I'm 50 percent done" or "I'm about 80 percent complete." The question is, where do these percentages come from, and do they reflect the truth? A percentage is always based on a numerator divided by a denominator: two absolute values that can be counted. Hence, when someone speaks of a percentage, there must be two sets of things to count, and counting involves pinpointing something.

If a team member doesn't know what the numerator or the denominator is, or what exactly is being counted, then the percentage given is highly questionable and of no value to the project manager.

In the discussion we had earlier today, we used a bottom-up approach: we did some counting before I asked for the percentage. That way, I could be sure you would know what the percentages mean. Even when there are things to count, sometimes team members just throw out numbers (as Roland did at first); it is necessary to validate the numbers.

Never use project measurements to measure individual productivity. For example, I would never use our numbers to penalize Harriet for being too slow. **Project measurements are there to indicate the overall health of the project and to highlight areas that need attention.**

"OK, I think everyone should go back and work out a more accurate representation of their progress," Simon said.

"These interfaces and reporting requirements are crucial for development," Smith added. "It normally takes time for the IT infrastructure departments to provide such
information. The team should initiate discussions about these things quickly, and even the discussions may take some time to schedule. In addition, there is an ongoing debate about how to categorize interfaces in the development process. I'll send you some advice on how to handle this."

**Requirements versus design**

There is a constant debate about the distinction between requirements and design. Should interface specifications -- or supplementary specifications that impact use cases, data fields, and so on -- be considered part of design, or are they part of requirements? Instead of debating the difference between the two, a better approach is to ask what details are needed to implement the system. Then, initiate discussions as early as possible and begin gathering information and input from end-user representatives or external system owners.

Struggling to distinguish between what represents a requirement and what represents design comes from the waterfall school of thought. If you are using an iterative development approach, you simply need to ensure that you have a working build or release at the end of an iteration. All requirements and design allocated to the iteration must be complete within that iteration, and it is likely that activities will be performed in parallel. Instead of drawing a line between requirements and design, consider what is needed for the iteration, and then work accordingly.

**Fashion show (week 8/40)**

Before the end of the (seventh) week, the user representatives requested that all the gathered requirements be submitted to them for initial review, to determine whether they were sufficiently detailed. They also requested an overall progress report on the requirements.

The development team spent a couple of days touching up their requirements artifacts and then submitted them to the user representatives at the end of the eighth week.

Gary, the key user representative, responded, "The progress metrics are quite good. Now at least I know where we are." Then, with some sarcasm, he added, "The artifacts are like a fashion show with styles from all over the map -- Paris, Milan, and New York! Some of these artifacts are 'fully dressed,' whereas some are 'scantily clad.' Shouldn't there be guidelines to make them consistent? It would make them much easier to read."

Now, Simon knew exactly what to do. If he had a problem, he should contact Smith. The first thing Smith did was to request copies of the use-case specifications and other requirement artifacts, since he had not seen them since the technology workshop conducted in week five. Then he scheduled a meeting with the team later in the week to walk through his comments.

As Smith read, he noted mistakes as well as better ways to organize and document the requirements. After that, he conducted a use-case structuring and guidelines workshop with the analysts that also included guidelines on domain modeling, use-case storyboarding, and using Microsoft Word styles.

When he met with the analysts, Smith helped them reduce the number of Roland's use cases from fifty to twenty-seven. There were some instances of functional decomposition, but fewer than at the beginning of the project. Some of the actors had
similar roles, and consequently similar interactions with the system, so Smith suggested that the actors could be generalized and the use cases merged. The team also restructured the other analysts' use cases slightly. Now, the project had a total of 75 use cases instead of 105. However, by putting related flows into the same use case, the analysts were able to identify more alternate flows by looking at the larger context, so the desired system was actually more complex than they had originally thought.

Smith also got the analysts to do use-case analysis and identify test cases. This helped them evaluate whether the use cases had sufficient detail for development and testing. It was also useful in preparing Harriet, Helen, and Roland to be project leaders when the development team joined them.

"Did we just decrease the scope of the project by reducing the number of use cases?" asked Gary. "Not at all. We restructured some of the use cases to improve the overall understandability of the requirements," Smith assured Gary. "If you look at the use cases now as compared to last week, you'll definitely find that they are clearer and that they handle more alternate conditions than before." Gary went through the use-case specifications again and confirmed Smith's statements. Gary also found that the style was now more consistent. "A lot of people have questions about the right quantity of use cases, so I'm going to send you some advice," said Smith, and he e-mailed him the tip below.

**How many use cases should be in a system?**

This is a common question. In this project there are 105 (see Table 3), or as few as 75 after restructuring. Although this is more than the twenty concrete use cases Ivar Jacobson suggested in his article "Use cases -- Yesterday, today, and tomorrow" in the March 2003 issue of *The Rational Edge*, neither is it a huge, unsound number like 700. But as Jacobson cautions, prescribing numbers of use cases for a system may lead people to "take undesirable actions to get the 'right' number."

The issue here is not how many use cases the project should have, but rather what should facilitators do when they encounter a sizeable number of use cases? Should they attempt to merge them or leave them as they are? The facilitator must take note of the project schedule.

In our case, we are in the eighth week of the project; time is not on our side, so overemphasizing use cases might cause more harm than good. The main idea is that the team should be able to effectively identify test scenarios from the use cases. I've taken a strong stand only when use cases had blatant errors, such as functional decomposition. Otherwise, I've let the analysts justify their practices and suggest their own improvements so they could learn more. The project belongs to the team, not to me.

How to structure use cases is a topic in itself, which I won't go into right now.

**Prioritizing requirements (week 9/40)**

In the ninth week of the project, Gary asked Simon if he thought they could achieve requirements sign-off by the end of the twelfth week. The progress metrics indicated that was not possible, and Gary himself was aware that there were still a number of internal issues to be sorted out.
"Maybe we should start development for what we already know. Your in-house expert, Smith, has reiterated the benefits of iterative development, and now I'm beginning to see his point. It seems there is no way we can sign off on requirements in three weeks. But can we start development for the high-priority and stable parts of our system?" Gary asked.

"Contractually, we are required to have requirements sign-off by the twelfth week," Simon reminded Gary.

"I'll make some recommendations to my management in exchange for some development: If your team can produce an executable by the twelfth week, I'll see if they'd be willing to shift the sign-off date by a month or two, to the twentieth week. We can work out the scope of the executable," Gary suggested, knowing very well that some of the critical issues really needed about a month to resolve.

"That sounds like a good trade," Simon said, knowing that he needed to start development or risk delays.

"There are seventy-five use cases, and we have about thirty weeks left. That means that we need to complete about two use cases per week," Gary said.

"Wow, you do arithmetic like Smith!" Simon said.

"Smith and I went to the same high school," Gary replied. "We had the same physics teacher."

Although Simon was pleased that Gary was offering some flexibility, he still had some reservations. "I think we can only complete one use case per week for the next couple of weeks," he said, knowing that the first programmer would not join the team until the following week, which was week ten.

"Then you'll need to catch up after that," Gary replied.

After this exchange, Simon called Smith. "Will I be able to complete two or more use cases every week?" said Simon. "At the moment, that doesn't seem realistic."

"I'll send you what I know about estimating completion times for use cases," said Smith, and he sent Simon the following advice.

**How long does it take to implement a use case?**

According to the literature on development, it takes about twenty person-hours to develop one use-case point. A simple use case has about five use-case points, which means that a simple use case takes about 100 person-hours, or approximately 2.5 person-weeks. But with all the right ingredients -- full-time workers with good application experience, good OO experience, high motivation, and so on -- the time required can be reduced by half. So it is not impossible for one person to complete one use case per week, especially if it is a simple use case. A spreadsheet for estimating project schedule based on use case points can be found at [http://www.saspin.org/ProjectEstimateMethod.xls](http://www.saspin.org/ProjectEstimateMethod.xls)

"OK, I've read your advice, and even without using that spreadsheet, I know that if my team was going to complete seventy-five use cases within thirty weeks (2.5 use cases per week), including all the rework, defect fixes, and change requests, they'll have to work faster," said Simon. "If I ask them to begin work immediately on some of the use cases, which ones should I choose?"
"Ah," said Smith. I can send you some advice on that," and he sent Simon the following.

**Which use cases should you develop in initial iterations?**

IBM Rational Unified Process®, or RUP®, recommends that architecturally significant use cases be allocated to the Elaboration phase. These are use cases that have large architectural coverage, fulfill central business functionality, and stress delicate points in the architecture.

There may be constraints that prevent the allocation or completion of critical use cases or scenarios in the earliest iterations. For example, external interfaces may not be available yet. Instead of waiting for such external dependencies to be ready, it is more prudent to work with what is known and use a proxy class to encapsulate the unknown -- in fact any encapsulation technique will do.

Moreover, at the beginning of the project, the team needs to "get its act together" and sort out process issues: who does what, and the handover points from one person to another. Choosing simple use cases for the first iteration (i.e., a mini iteration) allows the team to strengthen its processes without being overcome by all the technical challenges of the system in process. This is especially helpful for project managers making a transition from waterfall to iterative development mode: Simple use cases allow the team to produce an executable more readily and to sort out process issues.

However, this is not to say that you can have simple use cases for the entire Elaboration phase. If the team succeeds first with simple use cases, then they can deal with the architecturally significant ones in subsequent Elaboration iterations. If they have problems with the simple use cases, then there is a serious issue: They may lack development skills, which means there is a high risk that Elaboration will stretch out.

After reading this advice, Simon called the team together and asked, "Which use cases can we start developing?"

"I think a couple of mine are ready -- those that don't require any external interfaces," Roland volunteered.

Simon remembered how Roland had problems with requirements at the beginning of the project and even after Smith's technology workshop, so he was skeptical at first. But then he said to himself, "Hmmm, since Roland might not be able to contribute effectively as an analyst, I might as well get him to do some development and prove his worth." So Simon reshuffled his team, making Harriet the lead analyst and splitting the requirements between Harriet and Helen. Roland and the new programmer, Freddy, would do the development. Simon kept his fingers crossed.

With a smaller analyst team, it would be easier to coordinate and detail the use cases and related requirements. Roland would do the development for Harriet's use cases, and Freddy would do Helen's. Instead of two weeks to complete the requirements, Harriet now had ten more weeks. This was tremendous relief to her; she simply needed to ensure that there were sufficiently stable requirements for the development team to work on for each four-week iteration. At the end of each iteration, Harriet would submit a progress report, using the formats in Tables 2 and 3.
Iterative development (week 12/40 and later)

By the end of the twelfth week, Roland and Freddy had come up with quite an impressive deliverable. Roland had greater flair for design than for requirements, and Simon was relieved that he had made the right decision. By this time, most of the programming resources had joined the project. There were three development teams headed by Helen, Roland, and Freddy, respectively. Simon decided that Harriet alone should be in charge of requirements, as she had the best command of English and the best relationship with the user representatives. Moreover, she personally knew Smith, who the user representatives had come to respect. Furthermore, Simon intended to assign her the test manager role so she could plan the test environments and identify test cases.

Harriet was pleased with the arrangement because she could really focus on use cases and related requirements techniques. "I suggest you adjust the level of detail in your requirements to match the demands of your development team," Smith advised her when he heard of the new organization.

When Simon called him, Smith congratulated him. "You did what I would have recommended if you’d asked my advice," he told him.

"That's good to know -- but why don't you send me the advice anyway?" said Simon. Here is what Smith sent.

Know your team's capabilities

Everyone is good at something, and it is important for a project manager to find out each team member's special abilities. A mismatch between skills and roles can be quite disastrous, as you've seen on your own team. Project managers need to know the strengths of individual team members and must participate actively in the project to make first-hand observations and organize the team accordingly. They must not reorganize the team too often, however, because it is quite disruptive.

Project end (week 40)

At the end of the project, at week 40, only sixty out of the seventy-five use cases were implemented. At the beginning of the project, the user representatives had deliberately overscoped their requirements out of fear that they would be bounded by the requirements sign-off: They had identified more use cases than they really needed. Furthermore, some of the internal policy issues were never sorted out, so fifteen of the original use cases could not be implemented and were never detailed. This reduced a lot of work for Harriet. If CATALYST had followed a waterfall approach, she would have spent a lot of effort on these fifteen unnecessary use cases. She was not sitting around, however. Policy changes had resulted in changes to five original use cases, and they were not minor.

From Gary's perspective, having the system operational by the fortieth week was a critical success factor. He was willing to forget about the fifteen missing use cases in exchange for more development and tests up front. However, he let go of those fifteen use cases only at the thirtieth week, which was fairly late in the project, as it was a very useful bargaining chip to get what he wanted.

Happy ending
Although "CATALYST" did not complete according to plan, it did complete on time, at the fortieth week. Harriet felt relieved. The project had started on the wrong foot, but somehow, by applying several key principles, she and Simon had steered it toward success. Now that they had learned these precious lessons, both of them would be more confident on subsequent projects.

Both of them were happy with their introduction to use cases. There had doubts that use cases are an effective technique to describe behavioral requirements, but they also recognized that the technique is subject to the same difficulties as other requirements techniques.

- **Poor coordination between analysts leads to poor requirements;** having a strong and skilled leader helps.
- **Requirements are not always easy to express clearly in words;** using a combination of techniques helps: use cases, domain models, user-experience storyboards, and so on.
- **The number of requirements can become unmanageable if not controlled;** tracking the progress of requirements elicitation is necessary. Proceeding iteratively is the best policy.
- **Team members have different levels of experience and exposure to use cases and requirements techniques;** a technology workshop and a hands-on mini iteration under the guidance of an experienced facilitator can help.

It is also important to know that use cases, like any other requirements technique, are a means to an end. The goal is to deliver a useful system that meets actual needs.

Project "CATALYST" did not start well, and there could have been many possible outcomes. At each point we have discussed how the team managed to steer back on track. Having an experienced facilitator definitely helps. But even more important is the project team's and end-user representatives' commitment to make the project succeed. It is important to seek help early in the project. As long as we are willing to learn, with some practice and proper guidance, we can make the project succeed.

*Note: In a future article, we'll discuss use-case patterns and explore how to describe them effectively.*

Acknowledgments

I would like to thank the reviewers for their comments on this article. Thanks especially to Laurent Mondamert, who made many detailed and helpful comments that helped me improve the article significantly. Thanks also to Mike Bernard for his suggestion on the user-experience storyboard. And I would like to thank Catherine Southwood for her patience and excellent job in editing and tidying this article.

Notes
1 See http://sunset.usc.edu/research/COCOMOII/ and Software Cost Estimation with COCOMOII by Barry Boehm et al. (Prentice-Hall, 2000).


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