Top Ten Ways Project Teams Misuse Use Cases -- and How to Correct Them

Part II: Eliciting and Modeling Use Cases

by **Ellen Gottesdiener**
Principal
EBG Consulting

Use cases are a favorite way to describe the desired functionality of a software system under development. But creating use cases is by no means a foolproof process. In my many years of facilitating software development teams, I've encountered many cases of what I call (tongue firmly in cheek) "use case abuses" -- misuses and mistakes teams make when documenting and developing use cases. This article is the second in a two-part series that focuses on my own top ten list of "Misguided Guidelines" teams use for creating use cases.

Part I, published in the June issue of The Rational Edge, *examined ways to correct the first six of these erroneous "guidelines," which relate to use case form and content. In this issue, I explore the remaining four mistakes and misuses, which revolve around the process of eliciting and modeling use cases.*

Here's a quick recap of those "Ten Misguided Guidelines":

1. Don't bother with any other requirements representations. (Use cases are the only requirements model you'll need!)
2. Stump readers about the goal of your use case. (Name use cases obtusely using vague verbs such as *do* or *process*. If you can stump readers about the goal of a use case, then whatever you implement will be fine!)
3. Be ambiguous about the scope of your use cases. (There will be scope creep anyway; you can refactor your use cases later. Your users keep changing their minds, so why bother nailing things down?)

4. Include nonfunctional requirements and user interface details in your use-case text.
   (Not only will this give you a chance to sharpen your technical skills, but also it will make end users dependent on you to explain how things "really work.")
5. Use lots of extends and includes in your initial use-case diagrams.
   (This allows you to decompose use cases into itty-bitty units of work. After all, these are part of the UML use-case notation, so aren't you supposed to use them?)
6. Don't be concerned with defining business rules.
   (Even if they come up as you elicit and analyze use cases, you'll probably remember some of them when you design and code. If you must, throw a few into the use-case text. You can always make up the rest when you code and test.)
7. Don't involve subject matter experts in creating, reviewing, or verifying use cases.
   (They'll only raise questions!)
8. If you involve users at all in use-case definition, just "do it."
   (Why bother to prepare for any time with the users? It just creates a bunch of paperwork, and they keep changing their minds all the time, anyway.)
9. Write your first and only use-case draft in excruciating detail.
   (Why bother iterating with end users when they don't even know what they want, and they only want you to show them meaty stuff, anyway?)
10. Don't validate or verify your use cases.
    (That will only cause you to make revisions and do more rework, and it will give you change control problems during requirements gathering. So forget about it!)

If you recognize yourself in any of the last four "guidelines," you're not alone. Fortunately, there are some ways -- many of them surprisingly easy, all of them time-tested -- to avoid falling into these traps.

Correcting Misguided Use-Case Modeling Guidelines

Let's start with one mistake many teams make: trying to construct use cases without the help of the people who ultimately need and use the software.

7. Don't Involve Subject Matter Experts in Creating, Reviewing, or Verifying Use Cases.

One way to play Russian roulette with your requirements is to define them without user input. "Users" is a broad term that includes:

- *Direct end users* who will interact directly with the system (human actors).

- *Business subject matter experts* who have content knowledge but may not also be direct end users.
- *Customers* who sponsor the development project with resources. For commercial and business systems software, customers are the people or organizations who commission a software project. For shrink-wrap software, customers are end users of the software or perhaps buyers who might not interact directly with the software.

It can be hard to figure out how to implement a cost-effective system for involving users. Each category of users has insights into user requirements, yet it often seems that they are all too busy, uncommitted, or inaccessible to the project. If you're having trouble getting good feedback, it's time to rethink your user involvement strategies. First, consider what you've done in the past as an opportunity to learn what does *not* work in terms of getting user and customer involvement. After all, as Albert Einstein (or Benjamin Franklin, depending on which authority you consult) once quipped, "The definition of insanity is doing the same thing over and over and expecting different results." Don't assume various user classes are inaccessible. Instead, invent creative ways to involve them in your requirements process. Don't get stuck with elegant but inaccurate use cases. After all, user requirements should describe what users really need and not the project team's interpretation of possible needs.

To mitigate requirements risks on one project, I helped the development team conduct a chartering (start-up) workshop that included risk analysis. Participants generated a list of requirements risks, ranked them, and then identified risk mitigation strategies for the high-probability, high-impact risks. One such risk was lack of access to the "real" subject matter experts. It turned out that the best strategy was one we had already employed: inviting the project sponsor and stakeholders to the workshop. When these people saw the likely outcome if experts did not fully participate in the requirements work, they came up with several creative approaches. For example, they changed the timing of monthly reports to give the experts time to participate in requirements elicitation, offloaded some work to mid-level experts to expand their skills, and shifted the timing for requirements workshops to permit business experts to handle high-priority issues at the start and end of each workshop day.

How much user participation do you need? At a minimum, end users should verify requirements by doing use-case walkthroughs. Scenarios -- sequences of behaviors or example uses of a use case -- are the best way to conduct a walkthrough, especially if the end users have developed the scenarios. Even if they can't be involved in ongoing use-case specification, they can help you fix and evolve the use cases during that hour or two walkthrough.

Another approach is to engage subsets of customers: *sponsors* (who allocate resources to the development effort) and *champions* (who keep the project alive and people motivated) to help increase end-user involvement. To do that, take the pause that refreshes -- do a requirements debrief (also called a retrospective). Gather your team to assess your most recent requirements work. Seek to understand what

happened during requirements -- the good, the bad, and the ugly -- including how customers and users were (or weren't) involved. After learning about what worked, what didn't work, and what could be improved, name specific actions that you can take to do better next time. Then present this list to your managers and get their support. If managers and stakeholders don't participate in the debrief, then invite them to a brief presentation of your findings.

If you're developing commercial software, it is worth doing whatever it takes to gain access to your true end users. A big concern for requirements in these projects is a possible disconnect between what real end users need and want and what surrogate users *think* end users will need and want. In general, it's not a good idea to develop detailed use cases before doing some reality checks with real end users. Try to get them to commit to a session in which you can conduct reviews or walkthroughs with them, or show them early prototypes.

If you simply can't get the customer's time, then product managers can serve as stand-ins for end users. They can conduct focus groups to get feedback on draft user requirements -- or prototypes -- for some scenarios covered by your draft use cases. Or, you can ask knowledgeable business people in the marketing organization to role-play representative end users, inventing names and personal backgrounds for each person to make the role play experience more realistic. One commercial vendor conducted requirements workshops with its top three customers at each of the customers' respective locations, and promised those companies that they would be beta site customers. This was a win for both parties.

How can you use users' time most effectively? The users themselves can help you figure that out. As you attempt to involve them more fully, periodically solicit their feedback about how the process is working for them. Tell them what you need for the requirements development process to be successful, explaining the risks associated with insufficient user involvement. This will allow both of you to adjust your interactions and build sound and trusting relationships.

Now, on to the next common mistake.

8. If You Involve Users at All in Use-Case Definition, Just "Do It."

User requirements don't come from thin air. As you begin use-case modeling, there is always *something* to start with, even if it's a simple business goal statement or objective jotted on a napkin, a context diagram reverse-engineered from an existing system, or a list of customer complaints and change requests.

As a starting point, you need to draft some requirements models, even if they're wrong or incomplete, before gathering users together. These draft models should look rough and unfinished, inviting users to fix and elaborate on them. Low fidelity tools such as whiteboards, poster boards, and walls with sticky notes or cards pinned to them are good ways to do initial documentation for user requirements.

Draft models are a solid basis for asking focus *questions*, queries that direct people's attention to a specific topic. These questions give you the information you need to generate, evaluate, filter, elaborate, and verify the content of your models. For example, suppose you've drafted an actor table or map, and now you want to name use cases. Your focus question would be, "What goals does this actor have for interacting with the system?"

Use a variety of starting models, combining text and diagrams if possible. For example, you might start with a use-case list and an actor map, a context diagram and an event table, a list of stakeholder classes and a workflow (process map) diagram, or an analysis class model and some scenarios. Remember to pick models that fit the business problem domain (see the discussion of Misguided Guideline #1 in Part I of this series, published in the June issue of *The Rational Edge*).

Using multiple models (also discussed in Part I) helps you to quickly get details for your entire requirements set. For example, suppose you're using a high-level domain model or a statechart diagram. Focus questions for those models might be, "What do you do with <domain> to get your work done?" and "What system interactions are needed when the <domain> is <in statename> (e.g., "What do you need to do when claims are pending?").[1]

Be sure the right people are working with your user community. Requirements work is difficult and takes certain skills and proclivities, including the ability to listen, question, and abstract, as well as a genuine interest in people's work life, a sense of curiosity, and a tolerance for ambiguity. If team members working on requirements lack these skills, then seek training and mentoring for them or grow requirements expertise in others who have natural skills.

Let's examine another common error.

9. Write Your First and Only Draft in Excruciating Detail.

You can specify use cases using various forms and formats and with varying degrees of precision. To write highly detailed use cases, start with high-level descriptions and then provide greater detail iteratively, after you clarify each use case and ensure that it is important to the project. This strategy will save you unnecessary work, allow you to correct defects along the way, and speed your overall requirements effort.

Table 1 shows some sample use-case forms. Note that the level of complexity increases as you move down the table.

Table 1: Possible Use-Case Forms and Formats

Text Format	Visual Format
Use-case name only ("verb + [qualified] object")	Use-case diagram (ovals and Actors icons, à la the Unified Modeling Language, or UML)
Use-case name plus single sentence goal statement	Same as above
Use-case brief paragraph description (three to six sentences explaining what the use case does)	Use-case dependency diagram (UML ovals with dependency notation[2])
Sequence format (use-case header information plus a list of ordered steps)	Use-case flow or steps (activity diagram, flowchart, process map)
Conversational format (use-case header information plus two columns -- one for Actors and one for system responses -- written in a conversational style)	Use-case flow or steps (sequence diagram -- with an Actor class, activity diagram, flowchart, process map)

Here's an effective way to plan your iterations.

- First, decompose your various requirements models into their component parts. For example, a use-case paragraph is part of a complete use-case description, and data attributes are part of an analysis class model.
- Next, group these component parts at roughly the same level of detail. For example, business rules written using a business rule template would group with the sequential or conversation format use case and a fully attributed data model.
- Deliver these groups in chunks, or iterations, verifying each iteration with your development team or users in use-case reviews, walkthroughs, or prototype walkthroughs before beginning the next iteration.

This approach lets you correct requirements and adjust the process itself as you develop the requirements.

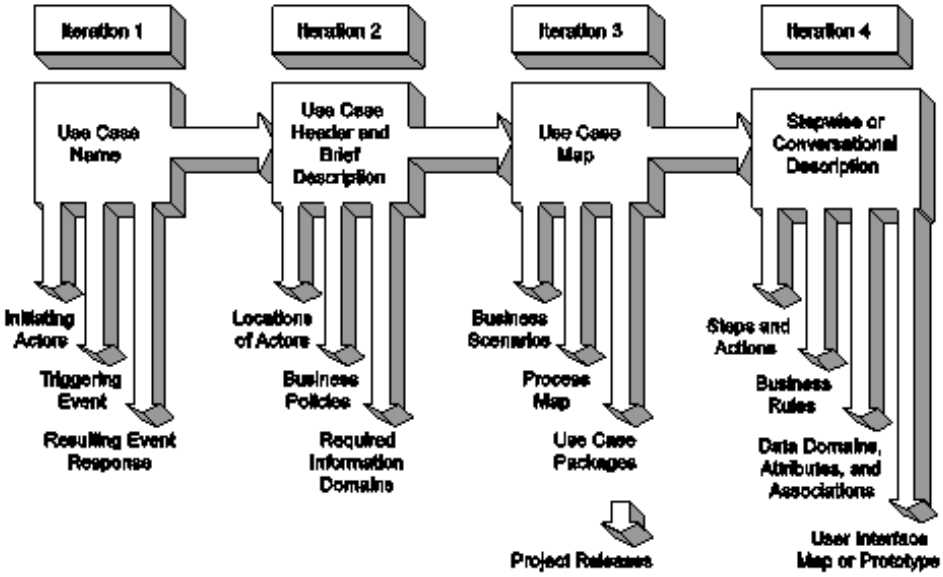

Figure 1: Iteratively Delivering Detailed Use-Case and Related Requirements Models

Figure 1 shows an example of a top-down path through your user requirements. I find that three or four iterations works best, so this one shows four iterations.

1. In Iteration 1, you discover use cases (and name them well!) and related scope-level requirements.
2. In Iteration 2, you define the use-case header and write a one-paragraph description of each use case. You also list the physical locations of the actors, associated business policies, and a high-level domain model (class model or data entities).
3. In Iteration 3, participants create a use-case map, which shows the sequence of use-case execution. At this point, you understand enough about the requirements to logically group the use cases into packages, forming cohesive use-case sets. These sets, in turn, should be prioritized and used to define releases or increments for delivery.
4. Finally, in Iteration 4, you rework the use cases by listing their steps, defining the business rules associated with each step, naming the data attributes needed by the steps and their rules, and sketching user interface prototypes for each use case.

For one use case, this set of iterations can take minutes to hours to define, depending on the use case's complexity and the knowledge of the people doing the work. To keep your momentum going, decide ahead of time how you will reach closure on the each iteration, and bite off the most important use cases first.

Creating a high-level, first-cut set of use cases can often give you enough

information to prioritize user requirements. You can then elaborate on only the most important use cases -- or those that are most unclear and therefore pose special risks. In the end, this can help you avoid requirements scope creep and optimize the time you devote to requirements development by spending it wisely on use cases that matter. It also enables you to avoid rework that might result from going down the wrong path.

Compiling a survey of use cases before you detail them is particularly helpful if you have a large project with multiple teams working concurrently on the system. The teams should work through the use case and related requirements at roughly the same level of precision, periodically regrouping to review each other's requirements to find shared requirements and avoid duplication of effort.

Calibrate the level of desired detail for use cases according to the project's needs. Alistair Cockburn[3] aptly points out that more correct and complete documentation is necessary on projects with a large number of team members producing mission-critical software with nondiscretionary funds. Factors to consider when deciding how detailed to go include:

- Project size (the number of people who have to be coordinated).
- Criticality of the software (human lives or simply human comfort at stake).
- Project priorities (Are the funds at stake essential or discretionary?).
- Project velocity (Are you driven by time as opposed to cost or functional scope?).
- Project team's familiarity with the problem domain.
- Project team's familiarity with use cases.

If the developers really know the domain and speed is of the essence, then a set of use-case names each with three to six sentences each will do. On the other hand, software governing such systems as airline cockpit controls, missile guidance, or human clinical trials must be precise and well documented. Decide how much detail you need and plan your iterations accordingly.

Finally, here's the last common mistake.

10. Don't Validate or Verify Your Use Cases.

Validation involves checking your use case for necessity, to ensure that your project is delivering the right functionality. Crosscheck each use case to be sure it satisfies one or more elements of your project vision or goals and objectives. *Verification* involves testing your use cases for correctness, completeness, clarity, and consistency, to ensure that you created the right thing. By "testing" I don't mean doing something on a computer after you write test scripts and build test data. Instead, you should challenge your use cases using other requirements models, such as scenarios, or by

using walkthroughs and reviews. Be sure to involve testers and quality analysts throughout requirements elicitation.

Scenarios are one of the most effective ways to test (and elicit) use cases. As you iterate through your use cases, try to test them with scenarios that business users have generated. Walk through each use case, beginning with the *happy case* (ideal) scenarios. Then move on to the *unhappy case* (error and exception) scenarios involving business rules violations.

In one project, we conducted several such one- to two-hour customer walkthroughs of use cases during requirements development. Because we didn't want to bog down this larger group with detailed use-case text, we walked through the scenarios using rough screen shots of the interaction flow. Participants had created the scenarios earlier, in short meetings with the primary subject matter expert, who was also a team member. Another team member adjusted the use-case text, business rules list, and domain model, as our lead designer walked through the scenario-driven prototype screens.

On another project, the project team generated test cases at the same time as use cases. The final use-case workshop then became a walkthrough session, in which participants led each test case through the use cases to see if the system functionality met expectations.

Another approach to validation is *peer reviews*: short meetings that focus on a work product, such as a set of use cases and related requirements models, to improve it and remove errors.[4] For these sessions to be successful, reviewers must prepare by individually checking the work product beforehand. Give them quality assurance (QA) checklists or questions that might help them find errors. For example:

- For each event in our event table and context diagram, is there at least one associated use case?
- Which use case handles each scenario?
- What states does this use case cover?
- For each use-case step, have all the business rules been defined?
- What data is needed to support those business rules?

The questions should suit the models you employ and their levels of detail.[5]

A complement to peer reviews is *perspective-based* reviews,[6] which invite various concerned parties -- perhaps specific actors, a tester, a designer, a help desk technician -- to examine the use cases from their unique perspective. Testers and quality analysts should also be active participants in your use-case modeling process. On one project, our test lead used our use-case templates as the basis for developing and documenting test cases. During our modeling sessions, he asked questions that helped us uncover missing data attributes and business rules. Involving testers and QA experts encourages a "test first" approach to requirements development, which yields higher quality requirements from the start.

Finally, do your best to get business experts to participate in your walkthroughs and reviews. In their absence, surrogate users, such as product development managers or business-savvy developers, can role-play being end-users and uncover important defects.

Using any or all of these verification techniques will help you find errors in your requirements that you might otherwise detect much later -- when they cost much more to correct.

The Case for Use Cases

Use cases are a wonderful way to define behavioral requirements for your software. In your zest to use them, however, don't fall into the trap of turning them into abuse cases. If you're guilty of following any of my ten "misguided guidelines," then it's time to reform your process and make your requirements effort more efficient and productive.

Here are some specific actions you can take:

- Use multiple models to represent requirements, and trace each to the other to maintain associations.
- Create strongly named use cases,[7] and don't rely solely on use-case diagramming elements to describe the use case (see Table 1).
- Proactively specify business rules as distinct requirements associated with your use cases, and keep nonfunctional requirements and GUI constructs out of your use-case text.
- Find ways to engage users in developing use cases and plan your time with them. Begin with draft requirements models and pose focus questions to see how well the models match users' actual requirements.
- Plan and follow an iterative use-case development process, beginning with rough-cut use cases. Prioritize the use cases as you go.
- Verify that each use case really belongs within your project scope. Continue to use checklists and conduct periodic reviews with users and project team members to verify that each use case is still necessary. Carefully control the scope of valid use cases.

The time you spent on developing and managing requirements and use cases is just a small part of your overall development effort, yet it can have a huge impact on the quality of your end product. By working to transform your habitual mistakes into positive actions, you can make your use cases a powerful means for delivering what your user community really needs.

Acknowledgments

I would like to thank the following reviewers for their helpful comments and suggestions: Alistair Cockburn, Gary Evans, Susan Lilly, Bill Nazzaro,

Paul Reed, Debra Schratz, Karl Wiegers, and Rebecca Wirfs-Brock.

References

Alistair Cockburn, "Use Cases, Ten Years Later." *Software Testing and Quality Engineering Magazine* (STQE), vol. 4, No.2, March/April 2002, pp. 37-40.

Alistair Cockburn, *Agile Software Development*. Addison-Wesley, 2002.

Alistair Cockburn, *Writing Effective Use Cases*. Addison-Wesley, 2000.

Alistair Cockburn, "Using Goal-Based Use Cases." *Journal of Object-Oriented Programming*, November/December 1997, pp. 56-62.

Martin Fowler, "Use and Abuse Cases." *Distributed Computing*, April 1998.

Ellen Gottesdiener, *Requirements by Collaboration: Workshops for Defining Needs*. Addison-Wesley, 2002.

Ellen Gottesdiener, "Collaborate for Quality: Using Collaborative Workshops to Determine Requirements." *Software Testing and Quality Engineering*, vol. 3, no 2, March/April 2001.

Ellen Gottesdiener, *Requirements Modeling with Use Cases and Business Rules* (course materials, EBG Consulting, Inc., 2002).

Daryl Kulak and Eamonn Guiney, *Use Cases: Requirements in Context*. Addison-Wesley, 2000.

Susan Lilly, "How to Avoid Use-Case Pitfalls." *Software Development Magazine*, January 2000.

Forrest Schull, Ioana Rus, and Victor Basili, "How Perspective-Based Reading Can Improve Requirements Inspections." *IEEE Computer*, 2000, vol.33, no. 7, pp.73-39.

Karl Wiegers, *Peer Reviews in Software: A Practical Guide*. Addison-Wesley, 2002.

Rebecca Wirfs-Brock and Alan McKean, "The Art of Writing Use Cases." (Tutorial), OOPSLA Conference, 2001. See http://www.wirfs-brock.com/pages/resources.html.

Notes

[1] For a comprehensive list of focus questions you can ask to elicit a requirements model by starting with an existing model, see the requirements workshop asset titled "Focus Questions for Modeling Tasks " on my Web site: http://www.ebgconsulting.com/reqtsbycollab.html.

[2] See: http://www.ebgconsulting.com/publications.html, section "Use Cases" for an example.

[3] See Alistair Cockburn, *Agile Software Development*. Addison-Wesley, 2002.

[4] See Karl Wiegers, *Peer Reviews in Software: A Practical Guide*. Addison-Wesley, 2002.

[5] For a more complete list of questions, see "QA Checklist for Checking the 'Doneness' of Requirements Models" on my Web site: http://www.ebgconsulting.com/ReqtsByCollab.html

[6] See Forrest Schull, Ioana Rus, and Victor Basili, "How Perspective-Based Reading Can Improve Requirements Inspections," *IEEE Computer*, 2000, vol.33, no. 7, pp. 73-79.

[7] See "Misguided Guideline #2" in the list above, the use-case naming guidelines in Part I of this series, published in the June 2002 issue of *The Rational Edge* (http://www.therationaledge.com/content/jun_02/t_misuseUseCases_eg.jsp) and also Leslee Probasco's advice in the March 2001 issue of *The Rational Edge* (http://www.therationaledge.com/content/mar_01/t_drusecase_lp.html).

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