Editor: Suzanne Robertson ■ The Atlantic Systems Guild ■ suzanne@systemsguild.com

The Top Risks of Requirements Engineering

Brian Lawrence, Karl Wiegers, and Christof Ebert

The worst thing that can happen in requirements engineering is that your set of requirements, however expressed, doesn't accurately represent your users' needs and consequently leads your team down the wrong development path. The whole point of requirements engineering is to steer your development toward producing the right software. If you don't get the requirements right, how well you execute the rest of the project doesn't matter because it will fail. So how are we led astray? The risk is greatest at several points.

Overlooking a crucial requirement

Perhaps the greatest risk in RE is missing a critical functional or attribute requirement. If you overlook an important user class, you'll probably have to do a big job—usually larger than stakeholders care to tolerate—to add in what that user needs. Missing a critical quality or performance attribute is typically even worse. Often, the only way adapt a software-based system to accommodate an important attribute is to re-architect. An example that many software developers have encountered recently is scalability in e-commerce. If designers don't keep scalability in mind when choosing their architectures (and many don't), they find themselves in a tough position when the usage load on their software leaps to a thousand or more times what they were expecting. System performance is inextricably tied to system architecture. In most instances, the only way to improve performance is to choose another one—that is, to start over from scratch. This is not popular when your senior managers promised you'd deliver next week. It's even worse if you make the discovery when your software is already in the field.

Inadequate customer representation

One of the central activities in RE is negotiating agreement on requirements. To achieve this agreement, you must find out what your customers really need. Not much of a negotiation will take place if you never actually interact with them. "Take it or leave it" happens all too frequently when we assume our design ideas suit our customers and don't bother to check if this assumption is really true. Customers only discover if we had the right idea when they attempt to use our software. If our confidence was misplaced, that's a late time to discover it. For example, a corporate IT development team told Karl that they recently rolled out a new application for internal use, but that they developed the system with virtually no customer input. The first time the users saw it was on delivery, and they immediately rejected the system as completely unacceptable. On the day you proudly unveil your new baby to the world, you don't want to hear, "Your baby is ugly!"

Modeling only functional requirements

Both the requirements literature and our practices have historically focused on functional requirements—the things our software systems are supposed to do. Functional requirements are the most obvious ones to the user, so most elicitation discussions focus on them. Perhaps more important, though, is gaining agreement on quality attribute requirements—the characteristics you intend your software to exhibit. Old standbys include reliability, performance, security, robustness, ease of use; others are scalability, innovation, coolness, or fun. Functional models, such as use cases, frequently gloss over the attribute requirements altogether. The attribute requirements are the heart and soul of why your customers will value your software. They determine why using your software is better than whatever they did before to achieve the same end.

For example, a system that fails to handle exceptions effectively will not be robust and will crash when unexpected conditions occur. It does you no good to simply record the requirement that "the system shall be robust." A skillful requirements analyst knows to ask the prompting questions that will elicit the user's implicit expectations about various attributes, explore what the user has in mind when he or she says "robust," and negotiate the inevitable tradeoffs among conflicting attributes.

Not inspecting requirements

The evidence is overwhelming and long known that the cost to remove defects in requirements increases geometrically with time. Once your software hits the field, removing a requirements defect costs at least a hundred times as much, assuming you can fix it at all. Inspecting your requirements models is the most effective way to identify ambiguities, unstated assumptions, conflicting requirements, and other defects at the earliest possible point. Of course, to hold an inspection, you must have something inspectable. And you have to believe that your set of requirements has defects that you need to identify. Personally, in all the years I've (Brian) seen requirements for countless software projects, I've never seen a defect-free set. If there's an ironclad rule in software development, it's "Always inspect your requirements." And choose inspection teams with a broad constituency, including testers. One company we know measured a 10-to-1 return on investment from performing inspections on requirements specifications, sustained over five years. For more on inspecting requirements, check out Karl Wiegers' article "Inspecting Requirements" at StickyMinds.com.

Attempting to perfect requirements before beginning construction

The time when we could know everything we needed to know before starting software construction is long past. Today, we live in an emergent world—some information simply isn't available early in our projects—and only emerges later. We can't possibly know everything we'd like to know before we start development. It's safer to assume that our requirements are going to change than that they won't. Yet for some projects, participants feel as though they must completely understand the requirements before any other work begins. For most projects, this is a mistake. You need to do some design and construction before you can tell how hard the job will be and how much each part will cost. As this kind of information becomes evident, it could well affect your views about your requirements, further changing them. Do the best job you can early to get a good set of requirements, but don't be discouraged if everything isn't absolutely certain. Identify those areas of uncertainty and move on, ensuring that someone is responsible for closing those gaps in your knowledge before construction is complete. Track the uncertain requirements carefully as your project proceeds.

Representing requirements in the form of designs

Possibly the subtlest risk in requirements engineering is letting designs creep into, and then remain in, your requirements specifications. When you view designs as requirements, several things happen. You run the risk of choosing a particular solution that might not be the best one to implement. Also, you undermine your ability to validate your system, because you are specifying the problem you hope to solve in terms of how you intend to solve it. All you can really do is verify that you built what you said you would. One surefire indicator that you're falling into this trap is highlighted by references to technology. Any time a technology is specified, you're using a design to represent the underlying requirement. Using designs as requirements is a subtle risk because although you might have specific information about what you want, it doesn't represent the underlying need and is consequently vulnerable to mistaken assumptions.

Whereas these risks are pretty serious, the greatest threat to project success is not performing requirements engineering at all. Requirements are your project's foundation. They define the level of quality you need, facilitate decision making, provide a basis for tracking progress, and serve as the basis for testing. If you let them remain unstated, you have no opportunity to examine and negotiate them with your customer and no way to tell when your project has met its objectives. Without clear requirements, how will you know when you're ready to release your product?

Brian Lawrence is a principal at Coyote Valley Software, a software consulting firm in Silicon Valley, California. He helps software organizations model and manage requirements, plan projects, and conduct peer reviews. Contact him at brian@coyotevalley.com.

Karl Wiegers is the principal consultant at Process Impact, a software process education and consulting company in Portland, Oregon. He has written books about peer review in software and software requirements. Contact him at kwiegers@acm.org.

Christof Ebert is director of software coordination and process improvement at Alcatel in Paris. He is also the *IEEE Software* associate editor for requirements. Contact him at christof.ebert@alcatel.be.