THE SMARTER WAY

When Requirements Go Bad

Understanding the causes for Requirements errors can help prevent them

By Kurt Bittner
Chief Technology Officer, Americas

April 2009

IVAR JACOBSON INTERNATIONAL

Introduction >

That error in Requirements is a contributing cause to the failure of many projects is nothing new. It is easy to attribute the problem to lack of clarity in the requirements - that if only the ambiguities could be eliminated all would be well. Ambiguous, unclear or incomplete requirements are serious problems, to be sure, but they are not the sole source of requirements errors. Requirements errors have many sources, however, and ambiguous or unclear requirements are only part of the problem. In order to reduce requirements errors, it's worth exploring a number of common types of errors to find their source and strategies for dealing with them. For the purposes of discussion, I find it useful to group requirements errors into three major categories:

- errors of conception
- errors of specification
- errors of implementation

Each one of these types of errors has separate symptoms and causes, and each must be solved in a different way.

What is a Requirement, Really?

Before talking about requirements errors, it's worth pausing for a moment to consider what requirements are. A requirement, I assert, is a statement that describes a desirable, or sometimes a mandatory, aspect of a solution. The degree to which the requirement is mandatory is often the subject of lengthy scoping debates. The observation that a requirement describes a solution is important, but at first this statement probably seems innocuous and perhaps even, well, obvious. The importance of recognizing a requirement as describing a solution comes from the possibility that the solution that the person writing the requirement has in mind may not actually be the only solution, let alone the best solution, to the problem at hand. If more than one person is writing requirements it is possible that each person actually has a slightly different mental picture of the solution, which means that even if individual requirements are clear and unambiguous, taken as a whole the requirements may not be consistent with one another. In the extreme case they may actually be contradictory, much like the old logic teaser:

The following statement is true.
The previous statement is false.

Taken individually, each of these statements is clear and unambiguous, but taken together they are nonsensical. It is easily possible for requirements to exhibit the same qualities.

Some teams try to look at requirements as a complete and unambiguous definition of the solution to be delivered. In today's world where a premium is typically placed on speedy delivery of a "good enough" solution, this perspective is often impractical. It is virtually impossible to specify every interesting aspect of a solution; just as it is impossible to completely and precisely describe even the simplest of chemical compounds, it is also impossible to completely and precisely describe every interesting aspect of something as complex as a software application. Even if it were possible to do, it would be prohibitively expensive to do so.

This leads to something more pragmatic: the requirements of a solution are the means by which the stakeholders and the development team arrive at a shared understanding of what it is the project is trying to do. The requirements themselves can be useful in facilitating interesting discussions and documenting understanding, but the important thing is the communication that occurs, not the requirements themselves.

The view that requirements are a complete and unambiguous definition of the solution derives from the use of requirements as a kind of contract. This may be appropriate in cases where an external firm is being hired to build the system, but there are typically contractual mechanisms for accomplishing this. Contracts in legal settings are most useful as establishing the terms for lawsuits. In the context of software development, if you have to result to "contractual commitments" the project has typically failed and the participants are simply trying to lay blame. It is better to work more proactively to achieve success, using the requirements as a communication vehicle rather than as a contractual document.

In this respect not all requirements have to be stated in an absolutely context-free and unambiguous manner. As long as there is a consensus about what is needed, some requirements can be less formal and open to creative alternatives, whereas some other requirements must be absolute. As an example, the exact look and feel of the user interface often presents a great opportunity for creative innovation, but the financial calculations performed by a billing system are absolute and not open to creative interpretation. Treating all requirements with the same rigor is a waste of time and may squelch creativity. The key is to understanding what must be precise and what can be less formal, and making sure that everyone understands the decision criteria.

Armed with this perspective, we can now turn our attention to different ways in which this communication can break down.

Errors of Conception >

Errors of Conception occur when the requirement is poorly conceived, when it solves the wrong problem or is based on flawed assumptions about the solution. These type of errors usually occur because the objectives of the solution are poorly understood. When these errors are present, the resulting requirements may be unambiguous and clear, but nevertheless wrong. As a result of these errors, even when the implementation is sound and of high quality, the resulting solution is of little or no value. It is in cases such as this where resorting to the defense that "the requirements were satisfied" is of little consolation.

Symptoms of errors of conceptions include:

- lack of clarity in the solution goals or the problem being solved
- unused or unneeded features
- unneeded complexity

There have been some interesting studies that suggest that this is actually a fairly large problem. These studies have shown that up to 50% of consumer electronics product returns are due to complexity - users could not figure out how to use the products (but presumably wanted the functionality that the product provided). The important thing about this that significant effort was expended to create the capabilities that people were unable to use, or perhaps did not even want. Other data from the Standish Group's Chaos Report suggests that even for successful projects, up to 30% of the function they delivered were unneeded or unused. Clearly someone is not clear about what is really needed in these solutions.

Root Causes and Avoidance Strategies >

It would be easy, but wrong, to write these errors off as simply due to poor communication. Poor communication is always present, but in most cases of errors of conception, someone is actively communicating the wrong information.

The usual source of the problem is talking to the wrong people, not finding the right subject matter expert with the right perspective to define the right solution. A story about a failed project illustrates the point:

A customer service system was developed and deployed at great expense. The leader of the requirements effort was very keen to use the latest windowing technologies, with lots of sophisticated capabilities to provide expert advice on how to handle different kinds of inquiries. The system was released to great fanfare and was, surprisingly, a complete failure.

It turned out that the customer service representatives who used the system were accustomed to using an old CRT system that had keystroke-accelerator commands for frequently performed actions. In addition, the staff were extensively trained on customer scenarios and had years of industry experience and did not need or welcome the intrusive guidance on how to do their jobs. As a result of all the extra mouse-clicks and distractions of the new system, call handling throughput actually declined and the old system was eventually turned back on.

What went wrong here? Basically, the "expert" was solving problems that did not need to be solved and had ignored the real needs of the intended users. Because the requirements dictated the "wrong" solution and were not questioned, the development team built the wrong solution.

A couple of things could have been done to avoid this problem:

- First, find the right subject matter expert. Since the development team rarely has the necessary business subject matter expertise, finding the right person is essential. In a sense, you should do enough investigation of the business problem to have a sense when purported experts really know what they are talking about.

- Second, get more than one opinion. There should be a consensus among the stakeholders about the nature of the problem being solved, and as the definition of the solution evolves you should continuously validate the "concept" of the solution with the stakeholders.

- Third, this continuous validation can't occur if you are heads-down writing requirements documents. You need to capture the ideas simply and succinctly and communicate them frequently.

An old adage to remember is that users don't know what they want, but they know what they don't want when they see it. This means that you will probably have to develop a number of different prototypes to walk-through various aspects of the system in order to make sure that you are building the right solution.

Other sources for errors of conception relate to poor understanding of the needs of the business (or sponsoring organization if the solution is being developed for a not-for-profit organization or government agency), which really means the needs of the stakeholders for the project. By this I mean going beyond what they say they need and uncovering the real needs that often lie hidden. An example from another domain illustrates the difference:

A patient walks into a doctor's office saying that he needs to get a prescription for painkillers. When the doctor asks why, the patient complains of severe headaches, but insists that it's no big deal, he just needs to get something to dull the pain and he doesn't need more tests or treatments. Against these protestations the doctor insists on more tests and eventually finds that the patient has an operable but benign tumor. After the operation, the patient's headaches disappear completely and he makes a speedy recovery.

In this example, giving the patient what he is asking for is exactly the wrong course of treatment. Similarly, if you give your users exactly what they ask for you may simply be treating the symptoms of a problem and fail to find a better solution to the real problem at hand. Users are not very good at coming up with solutions to their problems - that's where you can add real value - and letting them "dictate" a solution by dictating requirements usually ends up with everyone less than satisfied.

There are a variety of techniques to breaking through the initial requirements posed by the users, but they all center on focusing on uncovering the needs of the stakeholders and not, at least initially, soliciting requirements from them. Eventually you will uncover requirements, and as you do you need to relate them directly back to needs; if the connection to needs is weak then eliminate the requirement - it's either not needed (or there is a need that you are not clear about).

A final problem can occur that is a kind of "error of conception": sometimes the problem to be solved or the needs are clear, but it's just not a problem that is worth solving. This situation occurs when the cost of developing a solution outweigh the benefits, or the solution is simply technically infeasible. It's best to figure this out early, and so focusing early on identifying needs and then exploring the cost and technical feasibility of solutions through one or more prototypes is the best way to figure this out quickly.

Conclusion >

Errors of conception are the most significant kind of requirement errors, and they are the hardest to detect, but avoiding them has the greatest benefit. By preventing them you ensure that the solution will better meet the real needs of stakeholders; failing to prevent them, no amount of "disambiguation" of requirements language will yield a good result. No amount of precision can help a requirement that specifies the wrong behavior.

In this first article in a two-part series, I have covered the "errors of conception", which relate to requirements errors that arise from problems with the basic conception of the solution. In next month's issue I will discuss "errors of specification", which relate to errors in how the requirement is expressed, as well as "errors of implementation", which arise from problems in how the requirements are implemented.

About the Author

Kurt Bittner works for Ivar Jacobson International as their Chief Technology Officer for the Americas. In a career spanning twenty-four years successfully applied iterative approaches to delivering software solutions in a number of industries and problem domains. He is a co-author, with Ian Spence, of Use Case Modeling, published by Addison-Wesley in 2002, and Managing Iterative Software Development Projects, published by Addison-Wesley in 2006.

THE SMARTER WAY

Europe
+44 (0)20 7025 8070
info-eur@ivarjacobson.com

Americas
978-649-2856
info-usa@ivarjacobson.com

Australia
1300 567 280
Info-aus@ivarjacobson.com

Asia
+8610 82486030
info-asia@ivarjacobson.com

Find us on the Web:
www.ivarjacobson.com

Ivar Jacobson International

Ivar Jacobson International is a global services company that helps software organizations transform and improve the way in which they develop software solutions as well as guide them in meeting the expectations of the business. Our consultants provide an environment of experiential learning to develop the right competency levels amongst all roles and functions by becoming an intricate coach and mentor to the entire team. We have a framework that we adapt to effectively define and communicate business and technical expectations across the organization as well as create collective responsibility by teams and individuals for project outcomes. We introduce a proven practice driven approach that is goal oriented, incremental and measureable and is highly successful with either an existing software project or the implementation of new systems. We support our customer engagements with a rich set of technology assets inclusive of training materials, practice guides, and tooling.