medium.com

Ten Cosmic Truths About Software Requirements - Analyst's corner - Medium

Karl Wiegers

14–18 minutes

These facts apply to nearly every project. Ignore them at your peril.

Abstract photo created by kjpargeter — www.freepik.com

I have worked in the domain of software requirements and business analysis for more than 35 years, as a practitioner, consultant, and trainer. Having worked with more than 100 organizations of all sizes and types, I've observed some facts about requirements that appear to be universally applicable. This article presents some of these timeless "cosmic truths" and their implications for practicing business analysts, product owners, and product managers.

Cosmic Truth #1: If you don't get the requirements

right, it doesn't matter how well you execute the rest of the project.

Requirements serve as the foundation for all the project work that follows. By "requirements" I don't mean an initial specification you come up with early in the project, but rather the full set of requirements knowledge that is developed incrementally during the course of the project.

The purpose of a software development project is to build a product that provides value to some set of customers. Requirements development seeks to determine the mix of capabilities and characteristics in a solution that will best deliver this customer value. This understanding evolves over time as customers provide feedback on the early work and refine their expectations and needs. If a business analyst (BA) doesn't adequately explore these expectations and craft them into a set of product features and attributes, the chance of satisfying customers is slim.

One technique for validating requirements is to work with suitable customer representatives to develop user acceptance criteria or acceptance tests. Acceptance tests aren't a substitute for thorough system testing, but they do provide a valuable perspective to determine whether the requirements are indeed right.

Cosmic Truth #2: Requirements development is a discovery and invention process, not just a collection process.

People often talk about "gathering requirements." This phrase suggests that the requirements are just lying around waiting to be picked like flowers or to be sucked out of the users' brains by the BA. I prefer the term *requirements elicitation* to *requirements gathering.*

Elicitation is an exploratory activity. It includes some discovery and some invention, along with recording the requirements information that customer representatives present. Elicitation demands iteration. The participants in an elicitation discussion won't think of everything they'll need up front, and their thinking will change as the project continues.

A BA is an investigator, not simply a scribe. An adroit BA asks questions that stimulate the customers' thinking, uncover hidden

information, and generate new ideas.

It's fine for a BA to propose requirements that might meet customer needs, provided the customers agree that those requirements add value. A BA might ask, "Would it be helpful if the system could do <whatever idea he has>?" The customer might reply, "Wow, that would be great! We didn't even think to ask for that feature, but it would save our users a lot of time." This creativity is part of the value the BA adds to the requirements conversation.

Cosmic Truth #3: Change happens.

Requirements inevitably will change. Business needs evolve, new users or markets are identified, and business rules and government regulations are updated. Requirements become clearer as the key stakeholders are prompted to think more carefully about what they really are trying to do with the product.

Some people fear a "change control process." The objective is not to inhibit change, but rather to ensure that the project incorporates the right changes for the right reasons. You need to anticipate and accommodate changes to produce the minimum disruption and cost to the project and its stakeholders. Excessive churning of the requirements after they've been agreed upon suggests that elicitation was incomplete or ineffective — or that agreement was premature. When I implemented a change control process in a web development group once, the team members properly viewed it as a useful structure that helped them deal with their mammoth backlog of change requests.

Nearly every software project becomes larger than originally anticipated, so expect your requirements to grow over time. A growth rate of several percent per month can significantly impact a long-term project. To accommodate some expected growth, build contingency buffers into your project schedules. These buffers will keep your commitments from being trashed with the first change that comes along.

Instead of attempting to get all the requirements "right" up front and freeze them, baseline the first set of requirements based on what is known at the time. A *baseline* is a statement about the requirements set at a specific point in time: "We believe these requirements will meet a defined set of customer needs and are a suitable foundation for proceeding with the next stage of design and construction." Then

implement the initial, top-priority set of requirements, get some customer feedback, and move on the next slice of functionality. When an agile project commits to implement certain stories in a specific iteration, they're defining a requirements baseline for that iteration.

Change is never free. Even the act of considering a proposed change and then rejecting it consumes effort. Software people need to educate their project stakeholders so they understand that, sure, we can make that change you just requested, and here's what it's going to cost. Then the stakeholders can make appropriate business decisions about which changes should be incorporated and when.

Cosmic Truth #4: The interests of all the project stakeholders intersect in the requirements process.

A *stakeholder* is an individual or group who is actively involved in the project, who is affected by the project, or who can influence its outcome. Figure 1 identifies some typical categories of software project stakeholders.

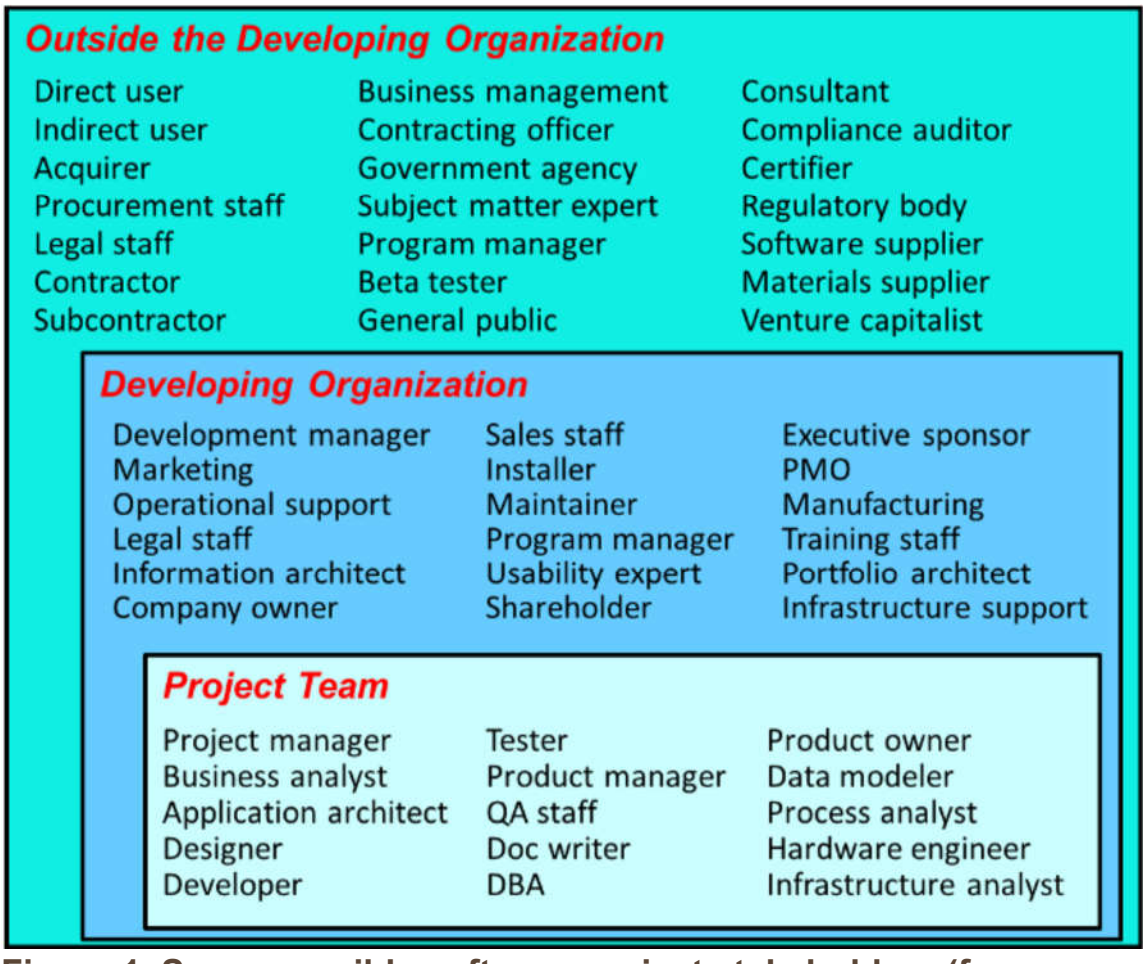

Figure 1. Some possible software project stakeholders (from Software Requirements, 3rd Ed.*, by Karl Wiegers and Joy Beatty)*.

The BA plays a vital communication role, interacting with all these stakeholders to specify a solution that will best satisfy all their needs, constraints, and interests. Identify your key stakeholder groups at the beginning of the project, and determine which individuals can best represent the interests of each group. You can count on stakeholders having conflicting interests that must be reconciled. Therefore, identify the decision makers who will resolve these conflicts and have them agree on their decision-making process — before they confront their first significant decision.

Cosmic Truth #5: Customer involvement is the most critical contributor to software quality.

Inadequate customer involvement is a leading cause of software project failure. Customers often claim they can't spend time working on requirements. However, customers who aren't happy with the delivered product always find plenty of time to point out the problems. You're going to get the customer input eventually. It's just a lot cheaper — and a lot less painful — to get that input early on, rather than after you've implemented the solution.

Customer involvement requires more than holding a workshop or two early in the project. Ongoing engagement by empowered and enthusiastic customer representatives is a critical success factor for software development. Following are some good practices for engaging customers in requirements development.

Identify user classes. Customers are a subset of stakeholders, and users are a subset of customers. You can further subdivide your user community into multiple user classes that have largely distinct needs. Some might be favored user classes, whose needs are more strongly aligned with the project's business objectives. Unrepresented user classes are likely to be disappointed with the project outcome.

Select product champions. You need to determine who will serve as the literal voice of the customer for each user class. I call these people product champions. Ideally, product champions are actual members of their user classes. Sometimes, though, you might need surrogates to speak for certain user classes to the best of their ability.

Agree on customer rights and responsibilities. People who must work together rarely discuss just how they'll collaborate. The BA should negotiate with the customer representatives early in the project to

agree on the responsibilities each party has to the requirements process.

Build prototypes. Prototypes let user representatives interact with a preliminary portion of the ultimate system to help refine the requirements and explore possible solutions. Prototypes are far more tangible than written requirements specifications and easier for users to relate to. However, prototypes aren't a substitute for documenting the requirements details so everyone is working toward the same objective.

Cosmic Truth #6: The customer is not always right, but the customer always has a point.

We all know the customer is not always right. Sometimes customers are in a bad mood, uninformed, or unreasonable. If you receive conflicting input from multiple customers, which one of those customers is "always right"? Following are some examples of situations in which a customer might not be right:

- Presenting solutions in the guise of requirements.
- Failing to prioritize requirements or expecting the loudest voice to get top priority ("decibel prioritization").
- Failing to respect business rules and other constraints.
- Expecting a new software system to drive business process changes.
- Failing to make timely decisions when an issue must be resolved.
- Not accepting the need for trade-offs in both functional and nonfunctional requirements.
- Demanding impossible commitments.
- Not accepting the cost of change.

The customer might not always be right, but the BA needs to understand and respect whatever point each customer is trying to make through his request for certain product features or attributes. Rather than simply promising anything a customer requests, strive to understand the rationale behind the customer's thinking and negotiate toward an acceptable solution.

Cosmic Truth #7: The first question to ask about a proposed new requirement is "Is this in scope?"

Anyone who's been in information technology or product development for long has worked on a project that has suffered from scope creep. It's normal — and often beneficial — for requirements to grow over the course of a project. Scope creep, though, refers to an uncontrolled and continuous increase in requirements that makes it impossible to deliver a product as planned.

To minimize scope creep, the project stakeholders must first agree on a scope definition, a boundary between the desired capabilities that lie within the scope for a given product release and those that do not. Then, whenever some stakeholder proposes some new functionality the BA can ask, "Is this requirement in scope?" If the answer is "no," then either defer (or reject) the requirement or expand the project scope, with the associated implications of cost and schedule increase. A poorly defined scope boundary is an open invitation to scope creep.

Cosmic Truth #8: Even the best requirements document cannot replace human dialog.

Even an excellent set of written requirements won't contain every bit of information developers and testers need to do their jobs. BAs and developers will always need to talk with knowledgeable users and subject matter experts to refine details, clarify ambiguities, and fill in the blanks. This is the rationale behind having product champions work closely with the team throughout the project.

A set of written requirements is still valuable and necessary, though, whether stored in a document, a spreadsheet, index cards, a requirements management tool, or some other form. A documented record of what stakeholders agreed to at a point in time — a *group memory* — is more reliable than human memories, and it can be shared with people who weren't privy to the original discussions.

The requirements specifications need more detail if you won't have opportunities for frequent conversations with users and other decision makers. This happens when you're outsourcing the implementation of a requirements set your team created. Expect to spend considerable time on review cycles to clarify and agree on what the requirements mean in those situations.

Cosmic Truth #9: The requirements might be vague, but the product will be specific.

Specifying requirements precisely is hard! You're inventing something new, and no one's exactly sure what the product should be and do.

People sometimes are comfortable with vague requirements. Customers might like vagueness because it means they can redefine those requirements later to mean whatever they want them to mean. Vague requirements can allow developers them to build whatever they think they should build. This is all great fun, but it doesn't help you create high-quality solutions.

Ultimately, you are building only one product, and someone needs to decide just what that product will be. Customers, BAs, or product owners who don't make the decisions force developers to do so, though they likely know far less about the problem or the business. Precise requirements lead to a better shared expectation of what the team will deliver.

Cosmic Truth #10: You're never going to have perfect requirements.

There's no way to know for certain that you haven't missed some requirement, and there will always be some undocumented (assumed or implied) requirements. Rather than declaring the requirements "done" at some point, define a series of baselines, snapshots in time that define an agreed-upon foundation for subsequent work and modifications.

Striving for perfection can lead to analysis paralysis, which can have a backlash effect. Stakeholders who have been burned once by a project that got mired in requirements issues sometimes are reluctant to invest in requirements development at all on their next project. This is an even more certain path to failure.

You don't succeed in business by writing a perfect set of requirements. Practically speaking, strive to develop requirements that are *good enough* to allow the team to proceed with design, construction, and testing at an acceptable level of risk. The risk is the threat of having to do expensive and unnecessary rework. Keep this practical goal of "good enough" in mind as you pursue your quest for high-quality requirements.

==========

This article is adapted from *More About Software Requirements* by Karl

Wiegers. Karl is the Principal Consultant at Process Impact. His most recent books include *Software Development Pearls*, *The Thoughtless Design of Everyday Things*, *Successful Business Analysis Consulting*, and *Software Requirements* .