



## **Ambiguous Business Value Harms Software Products**

Jeff Patton

Vol. 25, No. 1  
January/February 2008

This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.

IEEE  computer society

© 2008 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

For more information, please see [www.ieee.org/web/publications/rights/index.html](http://www.ieee.org/web/publications/rights/index.html).

## Ambiguous Business Value Harms Software Products

**Jeff Patton**

**O**ver the last seven years, working in agile development environments, I've heard many variations of the following conversation between two business stakeholders. Here, let's call the stakeholders Theresa and Eric:

**THERESA:** This feature needs to be in the next release.



**ERIC:** There's no room in the release. We'd have to push out another feature.

**THERESA:** What about this security feature? I don't see its business value.

**ERIC:** It's actually a really valuable feature for my target users.

**THERESA:** Well, my users really need this new feature—it has lots of value.

**ERIC:** Hmm... I guess we can defer the security feature until the next release. We've deferred it once already.

**THERESA:** Great, that will really please my users. This is the first feature in a whole set of functionality they can eventually use to make their lives easier.

### **Do you sense anything wrong in this conversation?**

You might have detected a few problems. I'll focus on three in particular.

#### **The business value isn't tangible**

Business value is something that delivers profit to the organization paying for the software in the form of an increase in revenue, an avoidance of costs, or an improvement in service. This is the dusty IRACIS (increase revenue, avoid costs, improve service) idea from Chris Gane and Trish Sarson's 1977 book *Structured Systems Analysis* (McDonnell Douglas Information).

Often, the term "value" is used a bit too subjectively.

I value something if it makes me feel good. If I'm representing the business, then I might view something that makes me feel good as a "business value." We need to tie value back to some tangible gain for the business.

#### **Sources of value aren't prioritized**

Another problem is that the two stakeholders are driving at different sources of value. Different people consider different things valuable. Prioritizing work becomes a tug-of-war in those circumstances. If we share a common idea of what's valuable, then we needn't pull in opposite directions. Clear statements about value and priority help.

#### **The software isn't expected to be used**

Theresa's plea for a feature that her users "can eventually use" raises the hair on the back of my neck. To realize value, software must not only be delivered but also used by its target users. Eventual use means eventual value. Giving users a little of what they want might show them you're listening. On the other hand, if those users can't actually use the software because there isn't enough there, they might assume you haven't heard them.

#### **Building a business-goal model**

Avoid these problems by building a simple business-goal model that concisely articulates and prioritizes business goals. You can build the model in a relatively short time using a collaborative workshop. You just need a conference room, a deck of index cards, and a handful of Sharpie markers.

#### **Brainstorm on business goals**

Assemble a group of business stakeholders—the people you believe are responsible for setting goals for your software development. Ask them to envision the software's next release and consider, if the software is successful, what goals would it achieve and what problems would it solve? Ask the participants to write their ideas on the index cards, one

idea per card. After a few minutes, everyone should have a small stack of cards with goals and problems.

**Merge goals**

Once everyone has written down their goals ask them to read the cards aloud one at a time and then place them on a table. As the second and third participants read their cards, you'll find some similarity. Ask them to cluster duplicate or similar cards.

**Distill clustered goals**

Discuss each cluster of similar goals or problems as a group. Agree on a goal that could stand in for—or distill—the ideas in the clustered cards. Write this on a card and place it on top of the stack. If a card is all by itself, it's a cluster of one. You'll end up with a number of clusters, each represented by a single card.

**Vote on priority**

Now ask the participants as a group to consider what goals the software must achieve (or what problems it should solve). Each member can vote for multiple goals. To determine how many votes each participant will get, divide the number of clusters by three and round up. For example: 11 clusters / 3 = 3.6, or 4 when rounded. Each member votes by marking the cards representing each cluster.

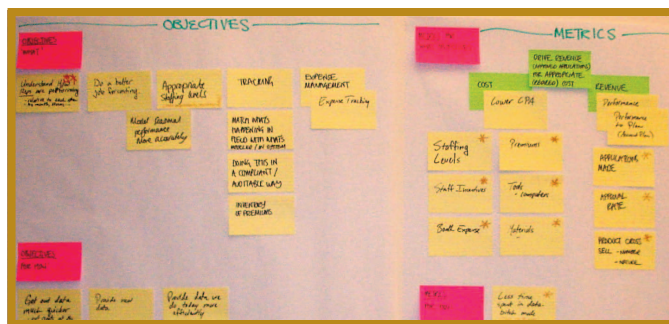
When the group is done, you'll likely find some clear winners—that is, one or two clusters that have many votes. Others will have only a few votes. Separate the top goals and order them on the basis of votes received.

**Identify metrics**

Now it's time for a discussion. For the high-priority goals ask, "How would we know if we were reaching this goal or solving this problem? What tangible changes would we observe in the organization?" The answers will result in observable changes that, ideally, you can measure.

Participants might find that achieving a goal will result in multiple measurable characteristics in the organization. Look for characteristics that are analog rather than just true or false. For example, "decrease the number of customer service calls" might be a good one, because it lets you measure the change in customer service calls after a release.

**Figure 1. A typical goal model built from index cards fixed to poster paper.**



You might need to rely on subjective measures. A questionnaire to gauge user satisfaction might be in order. In some cases, it's tough to identify what will change in the organization once you've reached the goal. Participants might need to rephrase or discard that goal.

Can you see a clear connection between each goal, its resulting metric, and your organization's profitability? As you discuss this, write down observable changes, one per card. Place these metrics on the table near each goal. You'll end up with a short list of prioritized goals and a few observable metrics for each one (see figure 1).

**Post the results**

Convert your list of goals and metrics into a poster that you can display publicly or into a PowerPoint slide for future presentations. Refer to these goals often, especially when considering features in the upcoming release.

**You've just built a simple goal model**

I call this a model because it concisely distills what we understand our goals to be today in some context and shows the relationship between those goals and their tangible outcomes. You'll find it more useful than other heavier-weight documents you might have ignored in the past.

The model was built using a collaborative-affinity-diagramming approach. An affinity diagram, in its simplest form, is a clustering of like information. Building this model collaboratively creates a common understanding of the source of value in our software.

Finally, we create metrics for our goals using a variation of the goal-question-metric method. Victor Basili published the GQM ideas as an approach for measuring software quality improvements. However, the GQM idea is so simple, you can apply it to just about anything.


**Publish your model with the goal's context**

Notice how the questions focused on the software's next release. This ties the goals to some point in time. Otherwise, people often set long-range, unachievable goals that are easy to ignore. When you see such "world peace"-type goals, make sure you've put the goal into context. When you post your model, note this context where applicable.

You'll need to run this exercise, or something like it, for your next release as well—or for each new context. And it wouldn't hurt to run it for your product as a whole to ensure that your release-level goals support your product's goals.

**B**ut wait," you might say. "This is the User-Centric column. You're supposed to focus on the needs of the user, not the business!"

The business doesn't get its value unless the software is used. In fact, focusing on getting the business its return will compel you to focus on the people who will eventually use the software. In organizations where we consistently fail our users, we also fail the business paying for the software. When users try to use software that's inefficient or prone to high error rates, or when they opt out of using the software altogether, any anticipated value evaporates. You would notice this lack of value quantified in the metrics you'd identified.

The first step to really focusing on users is to understand how it pays your organization to do so. Identifying and communicating business goals and metrics are the foundation of a good user-centered design approach. 

**Jeff Patton** teaches, consults, and coaches independently and as a member of ThoughtWorks. He is the founder and list moderator of the agile-usability Yahoo discussion group, is a columnist for StickyMinds.com, and won the Agile Alliance's 2007 Gordon Pask Award for contributions to agile development. Contact him at [jpatton@acm.org](mailto:jpatton@acm.org); [www.agileproductdesign.com](http://www.agileproductdesign.com).