point

Building What Stakeholders Desire

Ian Alexander

The devil is in the details of who the stakeholders are and what they desire.

The key to successful systems is building what the stakeholders desire. In a way, this point can hardly be gainsaid: nobody wants to build what the stakeholders don't desire. But as usual, the devil is in the details.

I will contrast two examples to illustrate what it can mean to build what stakeholders desire:

- Project A is a role-playing game on a portable device for the teenage market.
- Project B is an embedded software system for controlling a railway line's operations.

Project A's product manager is in no doubt that the key to success is building what its teenage consumers want: excitement, speed, a feeling of mastery, and so on. These emotional experiences are very far from traditional requirements, as David Callele recently argued ("Emotional Requirements in Video Games," *Proc. 14th IEEE Int'l Requirements Eng. Conf.*, IEEE CS Press, pp. 299–302) And beyond them, what players want is hard to pin down because the market changes fast, and at any moment a competitor (a powerful negative stakeholder) might produce a game that transforms expectations for all future products.

Other stakeholders include regulators who decide on the public's behalf, for instance, whether the game is excessively violent or sexual. Such stakeholders are important—they can even be show-stopping. The regulator, in turn, is influenced by politicians, who respond to public mood. Clearly, balancing the regulators' desires with the players' is essential, but so is defeating the competition: building what negative stakeholders *don't* desire. Finally, project A is a commercial success if it makes money—that is, if it satisfies its financial beneficiaries, such as company directors and shareholders.

The onion model in figure 1 illustrates the competing stakeholder pressures that the product manager must mediate for project A.

Project B is different. Its project manager and lead engineer know that the key to success is building a railway that works safely, safely, safely. The safety regulator's approval is mandatory, but first the project must integrate all the subcontractor components to make a system that actually works. So, the interfaces specified in the system's architecture are crucial. In theory, the system team can work out the interfaces in advance and impose requirements on subcontractors so that, for example, a signaling system's output exactly

Continued on page 64

0740-7459/07/$25.00 © 2007 IEEE

Don't Just DWTTY

Kent Beck

There's an old saying in software development that the users don't know what they want until you give them what they asked for. Experienced developers often smile ruefully when they hear this, having been told at least once in their careers that a system they had delivered was totally unacceptable when it was exactly (as far as they could tell) what was requested. There's more to delivering good software than following orders. Here, I describe what positive role developers can play in discovering and exploring requirements.

It's certainly an exaggeration to say that the worst thing you can do is deliver exactly what the users tell you. Making promises and not delivering anything is worse. Delivering what you want to develop without regard to what they want is worse. But putting most of yourself on the shelf while you develop is certainly far from the best way to deliver satisfying software.

One superficial critique of DWTTY (Do What They Tell You) is that it leads to lots of change requests. Actually, this is a strength of listening and responding to what customers tell you. Requests for change are a sign that customers are engaged, they care, they are thinking. Whatever developers do beyond DWTTY, they shouldn't lose this sense that the customers want to continue improving the system.

What happens when developers DWTTY? First, the features requested are often more expensive than they need to be. There are generally several ways to implement any given feature. By discussing a feature with customers, developers can offer options with their associated costs. Second, the requested features can lack coherence. Developers have the unique advantage of being able to see a whole domain with fresh eyes. Sometimes they can uncover hidden commonality between seemingly diverse operations and so dramatically simplify the interface. Third, customers often have a monolithic understanding of the system they envision, leading to riskier development. It often takes the developers' help to see how to slice the system into incremental releases, each delivering value. So, DWTTY leads to more expensive, less coherent systems delivered in large, risky releases.

Aside from these technical problems, DWTTY suffers from the fatal flaw of lack of responsibility. "Hey, don't whine to me. I just did what I was told" isn't a responsible position. Software development works best when everyone involved accepts responsibility for the system's value. Responsible developers apply their creativity and effort more fully and avoid short-

Continued on page 64

There's more to delivering good software than doing what the customer tells you.

0740-7459/07/$25.00 © 2007 IEEE

continued from page 63

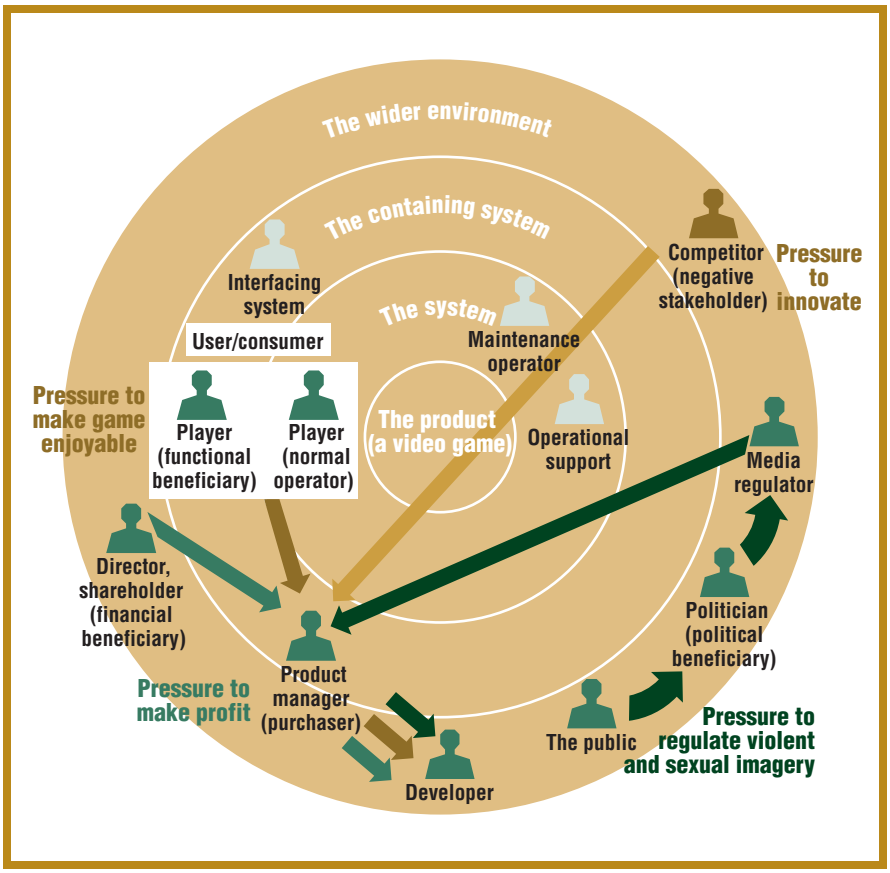

Figure 1. The onion model of video game stakeholder relationships.

fits the train. But if the train has already been designed, the system team faces a difficult integration task. The train manufacturer is a powerful stakeholder, and the project can only make progress by negotiation. Preferably, less powerful stakeholders' desires aren't trampled underfoot. The onion model for this case would look very different from figure 1.

I hope these examples show that phrases like "what the stakeholders desire" conceal a multitude of perils. Dealing with those desires involves trade-offs among large opposing forces that vary widely from project to project. The first step is to find out who the stakeholders are, and they will include not only beneficiaries but also the negative stakeholders, whose desires should *not* be met.

So, should we give stakeholders what they desire? It's impossible to please everybody, of course, but overall satisfaction of the key stakeholders' desires is crucial for product success.

Ian Alexander is an independent consultant specializing in requirements engineering for systems in the automotive, aerospace, transport, telecommunications, and public-service sectors. Contact him at iany@easynet.co.uk.

continued from page 63

cuts that remove value from systems.

There are many alternatives to DWTTY. A binary-thinking developer would say, "Well, if they aren't in charge, then I'm in charge." This strategy denies customers their right to be responsible for the system. The developer, with a necessarily shallow understanding of the domain, will likely make many wrong decisions, especially in the critical early development stages. The question isn't, "Who's in charge?" but rather, "How can we work together?"

The second edition of *Extreme Programming Explained* lists 14 principles consistent with development excellence. All of them apply to the requirements dialogue. I could go through all 14 principles, but I only have 750 words and I'm eager to see what my worthy opponent has to say, so I'll mention one and let you explore the rest. The principle of baby steps suggests working in small, concrete steps. If we apply this principle to requirements, the requirements dialogue will encourage everyone to achieve something concrete quickly and frequently. Baby steps create feedback, reduce wasted effort, and encourage trust among the team members.

Overall, a team could manifest these

principles in many ways. An Extreme-Programming-style team might sit together, pair a customer and a developer for challenging tasks, or release software to production weekly. A waterfall-style team might use the same principles to hold joint design sessions, train the developers as users of the system to be replaced, or split the release cycle in half (two releases a year will encourage more collaboration than one).

The precise practices aren't as important as the intent behind the practices. If everyone on the team accepts responsibility for the system's value, if developers commit to listening to the customers' real needs, if customers digest feedback about their wishes, then the whole team will accomplish much more than they would if the developers just "do what they tell me."

Kent Beck is the director of Three Rivers Institute in southern Oregon. Contact him at kent@threeriversinstitute.org.

Ian Responds

Kent Beck's point about just taking orders is well taken, as is his statement that practices aren't as important as intent. If that means appropriate agility for each project, few would gainsay it. His illustration of a "waterfall-style team" using appropriately agile practices such as joint design sessions is very welcome.

It can't always be right to develop the requirements during coding—still less, while concrete is being poured and metal is being cut. But it's certainly right to select a life cycle that encourages dialogue, with iteration in "baby steps." Those steps could be creating mock-ups and prototypes in any form—sketches, diagrams, film, acted scenes, clay models, whatever—followed by feedback from stakeholders.

On a much larger scale, iteration can take place over candidate designs that might meet the customer's goals. These are then traded off according to how well they're predicted to perform. The developers' role here might be to propose alternative designs. The customer sets the goals and evaluates the alternatives against all the stakeholders' points of view. If complete agreement isn't reached first time, as is likely, then the developers explore the refined goals and design options in search of a better compromise. The detailed requirements effectively flow from this repeated trade-off cycle.

We seem dangerously close to agreeing that appropriate agility is far better than believing what you're told at the start. Waterfall activities make sense only within an appropriate life cycle—very likely, an iterative one.

Kent Responds

I would like to thank Ian Alexander for his clear exposition of the case for "implement customer desires" as a development metaphor. His diagram, in particular, admirably supports his points. However, I'm not convinced that his picture of development is sufficient to guide projects.

My most serious objection is the lack of feedback loops in his model. "Desires" aren't a static target to be met or missed. If together the team can choose to defer a difficult-but-not-urgent task, and if the customers later decide that the whole area of functionality needn't be implemented, I would argue that they have been extremely effective even though they didn't "implement customer desires."

Software development at its best is a rich web of relationships among people with different perspectives who choose to work together on common goals. Everyone is responsible for stating their needs, listening to the needs of others, and working toward those goals. In the process, they get some of their needs met and leave others to be met in other ways. It's not just what the customers want (or what the programmers want, for that matter), it's what everyone can accomplish together that spells success or failure.