Chapter 19:

DESIGN VS. IMPLEMENTATION

Define The User Experience Before Proceeding To Build

There are many things in the software development process that can and should be done in parallel. For example, I have long argued that requirements (functionality) and design (user experience design) are intertwined and should be done together. I don't like the old waterfall model of a product manager doing "requirements" and handing that off to interaction designers that do "design." Most teams understand now that this is an obsolete view of product development.

I also believe that great strides have been made by software engineering teams that have learned the value of doing implementation and testing in parallel. The old model of the engineer writing software and then handing it all off to a QA person to test actually takes longer and the result is less reliable. *Agile* methods like *XP* demonstrate the value of doing implementation and testing in concert.

That said, one thing that many teams try to do in parallel—but should not—is user experience design and implementation.

There are several reasons for this:

First, there is a dynamic in software teams that is important to

recognize: once implementation begins, it becomes increasingly difficult to make the fundamental changes that will likely be necessary as you work through your user experience design ideas. Partly this is technical—engineering teams must make some early architectural decisions based on assumptions about the requirements and designs in order to make progress. These early decisions are important and have big consequences. Partly, this is psychological—there is a mindset that takes hold once the team shifts into implementation mode, and it's demotivating to go backwards. Partly, this is practical—the clock is ticking, and rework and churn just compounds the pressure the team is under. So even though methods like *Agile* advocate embracing change, you quickly find that some changes are much more welcome than others.

Second, user experience design deals with very difficult questions of both usability and value and, in order to come up with a product that is both usable and valuable, you will need to try ideas out—early and often. One common response is "We'll get feedback during beta," or with *Agile* teams, "We'll test the idea out at the end of the sprint." Unfortunately, this is far too long to wait to test out an idea. A good user experience designer will want to try out dozens of ideas and approaches in a matter of days, and the thought of waiting even for a two- to four-week sprint would be debilitating as the frequency is an order of magnitude too slow.

Third, and related to the above, I argue that to try out an idea you need a high-fidelity prototype. Some will argue that the beta release can be viewed as the prototype, or that the result of the sprint can be viewed as the prototype. But even beyond waiting too long for that software to be available for test, it's important to realize that prototype software is far different than production software. Prototype software needs to be truly disposable. It needs to be something that can be changed substantially in a few hours. What is necessary for production software is like dragging around a boat anchor for a prototype. You'll also find that different types of people like to write prototype versus production software.

Fourth, while it often makes excellent sense to break up a release into several iterations to implement (this reduces risk, improves quality, and eases integration) a user experience is often not something that can be designed in pieces. You have to look at the user experience holistically—it has to make sense to the user at each release. While it's easy to "stub out" software that's not yet available, it's not so easy to do the same for the user experience.

Finally, user experience designers don't necessarily require a lot of time (just as with software engineering, it depends on the methodology they are using, the particular product requirements, and the skills and experience of the specific people), but they do require some time. Even if it's only a week or two.

If you try to start implementation at the same time as design, here's what you will almost certainly see: the designers will be stressed trying to accomplish weeks worth of work in just days, the engineers get anxious as they wait for the designers to give them something, soon the designers will reluctantly make some preliminary guesses to allow the engineers to get going and then hurry to try to get something decent put together before the engineers get too far down the path. However, when they finally do have something, it'll be too late and the engineers will say "we can get to it in the next round" but of course the next round has its own priorities. The designers do not feel good about what is built and shipped, and the customers don't like the result either.

In the worst-case situation, the designers come to the conclusion that they need to go work for a company that prioritizes the user experience.

Fortunately, this really isn't a hard problem to solve. The key is that the user experience design needs to happen before the implementation begins. This is one situation where sequential is important. The requirements and design happen together, and then implementation and test can happen together.

For *Agile* teams, the product manager and user experience designers use the "sprint zero" concept to stay a step or two ahead of the engineers. You are still working to design in as small of increments as possible. It requires a somewhat richer definition of what's in the backlog, but the team will be happier and the product will be better for it.

The exception to the rule is when the engineers have a lot of backend infrastructure work to do. In this situation, the engineering team can be working on this while the user experience is being defined. There will be some interdependencies, but they can be managed. If your user experience designers are about to revolt, have your engineers work on the infrastructure for a release cycle or two, as this gives the designers time to work on creating a backlog of good design.

Note that although I'm advocating that requirements and design are both done before implementation begins, you will still need at least someone from engineering to review the design work from the start, as it's critical for them to assess feasibility and costs along the way. This is necessary to inform the design process. Remember that the objective is to ensure that you discover a product definition that is valuable, and usable.

There is a remarkable amount of confusion out there today in terms of incorporating good design, especially as many teams experiment with *Agile* methods. I think this is unfortunate as with only a few caveats and adjustments, the *Agile* methods can be a huge step forward for teams that previously used conventional waterfall methods. I talk about the root causes of this confusion and how you can get the best of both worlds in the chapter *Succeeding with Agile Methods*.