David Chappell

March 2012

THE BUSINESS VALUE OF AGILE DEVELOPMENT

Sponsored by Microsoft Corporation

When it comes to creating custom applications, too many of us live in denial. We want to believe that it's possible to predict accurately how long a group of developers will take to build software that meets our requirements. We also want to believe that we can know what those requirements are up front, before we've seen any part of the application, and that the requirements won't change during development.

Sadly, none of these things are true for most projects. We can't predict how long development will take, largely because we can't get the requirements right up front and we can't stop them from changing. Because we deny these realities, many organizations still use software development processes that don't work well.

Fortunately, this is changing. Agile development processes get more popular every day, primarily because they're rooted in reality: They're designed to accommodate change. Doing software development in this way can be scary at first, especially for the business leaders who are footing the bill. This needn't be the case, however. The truth is that agile processes are usually better both for development teams and for the business people who pay them. To understand why this is true, we need to start by understanding what an agile process really is.

What Agile Development Means

The challenge is always the same: We need to create software in the face of uncertainty. At the start of a development project, we don't know whether we've defined the project's requirements correctly. We also don't know how those requirements will change while we're building the software.

A traditional development process does its best to pretend this uncertainty doesn't exist. In the classic waterfall approach, for example, an organization creates detailed plans and precise schedules before writing any code. But real development projects rarely comply with these plans and schedules—they're notoriously unruly. The core reason for this is that even though we use the term "software engineering", writing code isn't like other kinds of engineering. In traditional engineering projects—building a bridge, say, or constructing a factory—it's usually possible to define stable requirements up front. Once you've done this, creating plans and schedules based on previous experience is straightforward.

Software development just isn't like this[1]. Creating stable requirements up front is usually impossible, in part because people don't know what they want until they see it. And since every development project involves some innovation—if it doesn't, you should be buying rather than building the software—uncertainty is unavoidable.

Rather than resisting change, an agile process embraces it.

Traditional development processes work against these realities. Agile processes, however, are designed for this situation. Because requirements change, an agile process provides a way to add, remove, and modify requirements during the development process. Rather than resisting change, an agile process embraces it. Just as important, the process recognizes that short-term plans are more stable than long-term plans. You might not know exactly what you want to be doing three months from now, but you probably do know what you want to do in the next three weeks.

To accomplish this, an agile development process relies on iteration. Rather than the traditional approach of defining all of the requirements, writing all of the code, then testing that code, an agile process does these things

[1] In fact, maybe we should stop talking about software engineering—it's a misleading term.

over and over in smaller iterations. Each iteration creates more of the finished product, with the requirements updated at the start of each one. Figure 1 illustrates this idea.

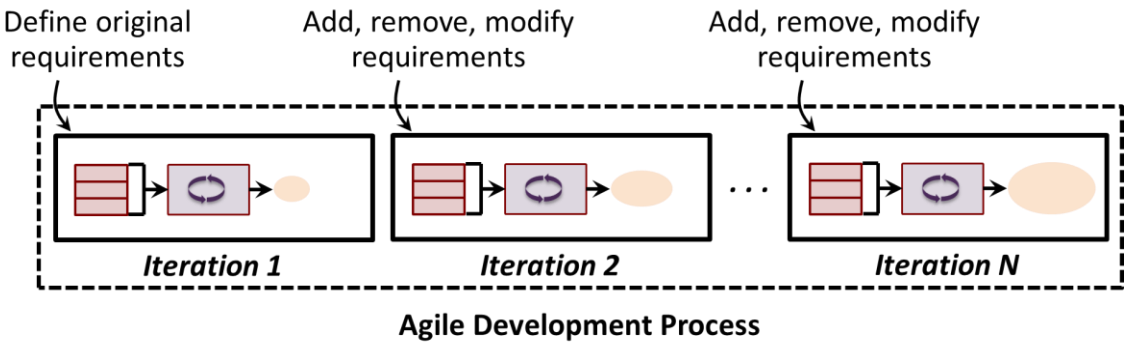

Figure 1: An agile development process is iterative, with the ability to add, remove, or modify the project's requirements at the beginning of each iteration.

The core idea is simple: Because requirements can be reassessed at the start of each iteration, an agile process accommodates change. And since change is all but certain to happen in software development projects, this is a very good thing.

An Example Agile Process: Scrum

Many agile processes exist today, but the most popular is almost certainly Scrum. Scrum is simple to understand, and it provides a concrete illustration of how agile development works.

A key idea in Scrum is the notion of a *product owner*. The product owner represents the sponsor of the project, such as the business group that's paying for this development effort. The product owner doesn't need to understand code—it's not a technical role—but he or she does need to have a clear sense of what the software needs to do. Figure 2 summarizes the basics of Scrum from the perspective of the product owner and the business sponsor this owner represents.

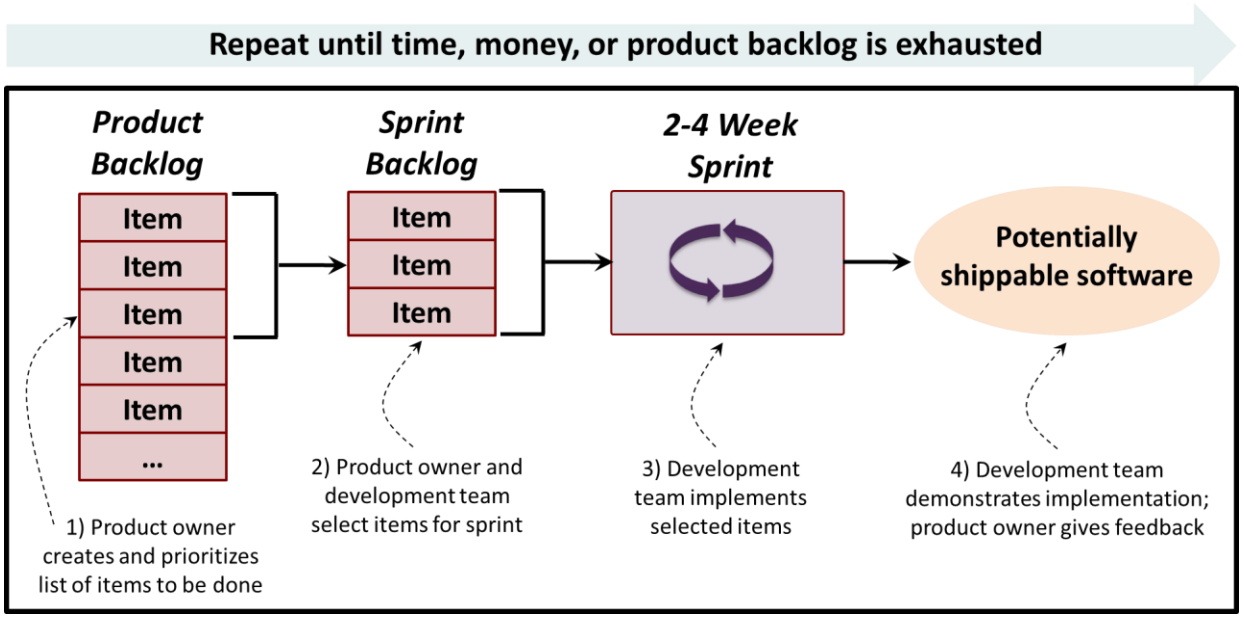

Figure 2: Each Scrum iteration includes the steps shown here.

Like agile processes in general, Scrum is iterative: Figure 2 shows a single iteration. From the perspective of a business sponsor, the iteration has four steps:

- First, the product owner creates and prioritizes a list of items to be done (step 1). This list is called the *product backlog*, and it contains all of the requirements for the product. It's a dynamic thing; what's on the list will change with each iteration.
- Next, based on priorities defined by the product owner, the development team and the product owner select items from the product backlog for the *sprint backlog* (step 2). These items will be implemented by the development team during this *sprint*, which commonly lasts two to four weeks.
- The development team now implements the selected items (step 3). How they do this is entirely their responsibility; Scrum development teams are self-organizing. The result of the sprint is potentially shippable software. In other words, it's a usable and tested implementation of the items in the sprint backlog.
- Once the sprint is over, the development team demonstrates the implementation to the product owner (step 4). By getting concrete, hands-on experience with the software, the product owner can understand what's actually being created. He or she then gives feedback, including suggestions for new or changed features that become part of the product backlog for the next iteration.

By the time this iteration completes, requirements might have been added, removed, or modified. Whatever has changed, the product owner re-prioritizes the product backlog and the next iteration continues. This process continues until one of three things happens: The project runs out of money, the time allocated for the project expires, or all of the items on the product backlog are implemented.

Notice how different this is from a traditional development process. Agile development still needs a plan, but the plan is only detailed on an iteration-by-iteration basis. In fact, lower priority items in the product backlog are

intentionally left loosely defined; things might change by the time they're implemented. Progress isn't defined by adherence to a pre-defined plan. Instead, it's measured by the amount of usable software created.

The Business Benefits of Agile Development

Developers like agile development for various reasons, not least because it gives them more autonomy in how they do their job. But organizations should adopt agile processes because of the business benefits they bring, not just because they make developers happier. Those benefits are very real, and they include the following:

- The development team is more likely to create what the business really needs. The product owner—who represents the business, remember—is directly responsible for creating and prioritizing the items in the product backlog. The development team builds the most important things first, and so the sponsoring business group is much more likely to get what it most wants, even if resource constraints don't let it get everything it would like. And because the process is iterative, there are many chances to make changes, even near the end of the project.

- The product owner has a much better understanding of what's being created during the project. He or she has direct visibility into the project at every iteration, with the ability to change what happens in the next one. And nothing beats seeing and using early versions of software, which is by far the best way to understand what the software's users really want.

- The risk of a failed project is reduced. Agile's iterative approach lets you measure the right thing, which is whether you're getting the software you want. In a traditional process, you can only measure whether the project matches the plan, which isn't what you really care about. Without an agile process, you don't know whether you've gotten the right software until the project is over, when it's too late to do anything about it.

Adopting an agile process changes how a development team interacts with its business sponsor. The changes include these:

- The business sponsor must be involved throughout the project in adding and removing requirements, re-prioritizing them, and more. Rather than defining requirements up front, then hoping the team delivers the right thing many months later, the product owner plays an active role in each iteration.

- The business sponsor should expect continually improved business value, expressed in working software, rather than precise compliance with a long-term plan. This changes how project management is done, which can upset traditional project managers. Steeped in knowledge about predictable processes, people in this role can find the move to agile processes uncomfortable.

- What a development group promises its business sponsor changes. Unlike traditional development processes, an agile process doesn't attempt to precisely specify all three variables—time, budget, and scope of work—at the beginning of a project. This can feel alien, especially when dealing with an outside development firm. Most business sponsors are used to contracts that specify a fixed set of features implemented in a fixed amount of time for a fixed price. Traditional development processes often promise this, but it's important to realize that those promises are lies. Unless the project really does allow defining an unchanging set of requirements up front—a rarity—expecting this is a recipe for disappointment.

Organizations should adopt agile processes because of the business benefits they bring.

Being better at writing custom software brings a competitive advantage. To get this advantage, we have to give up the illusion that we can accurately define up front all of the requirements and timelines for a development project. We have to embrace reality.

Believing that you can use a predictable process when you can't is dangerous—it doesn't help. What does help is adopting a development process designed to accommodate change. What helps is agile development.

About the Author

David Chappell is Principal of Chappell & Associates (www.davidchappell.com) in San Francisco, California. Through his speaking, writing, and consulting, he helps people around the world understand, use, and make better decisions about new technologies.