

My experience in Agile processes is twofold – Scrum and the standard startup/dot.com method of none (which you could also name “punt,” “wing-it,” and “the abyss”). As such, I will briefly talk about Scrum and XP, then describe what I feel to be the challenge to process, agile or otherwise. While many consider these two agile processes to be complimentary, I find XP makes a good basis for comparison.

I believe that some interesting points of the Agile Manifesto are:

- 1) Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- 2) Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for the shorter time scale.
- 3) Simplicity--the art of maximizing the amount of work not done--is essential.

Both Scrum and XP support time-boxing and iterative development cycles (called sprints in Scrum), that allow the developers to focus on a fixed set of requirements (supporting point 2 above). Because it is time-boxed, the deadline is the most important criterion, and both processes allow the developers to drop or simplify features so that they may complete the deliverable on time. Yet the two processes begin to differ at this point. XP is somewhat free-formed, allowing the developers to address new issues or requirements on the fly. In fact, XP advocates involving the customer in these cycles, so it is quite likely that requirements will become more clear or even change in the middle of the cycle. Scrum, on the other hand, prohibits the introduction from the outside of new features in the middle of a cycle (any rule can of course be broken when the situation warrants). Developers can factor in new requirements at the start of the next cycle, which because the cycles are short, doesn't cause long delays. The purpose of this practice is to ensure that the delivery is on time, and to prevent feature creep.

It is this third feature, maximizing the amount of work not done, that I find critical to successful development. Simplicity (an XP concept) in design can be quite liberating, freeing developers to produce quality software quickly. Simplicity entails designing a robust system, while avoiding the tendency to over-engineer a product to handle every situation that someone conceives that can possibly happen. Otherwise, developers are continually thinking about “how to make the product better and more robust”, and “what if this situation ever occurs,”- but simplicity is merely a concept. Scrum goes one step further and reinforces this philosophy by freezing the sprint's contents, effectively limiting these “enhancements,” and hopefully forcing the developers to consider whether some particular new feature actually belongs in a subsequent load or not.

It is precisely these little “enhancements” that cause so much trouble. My experience in the startup environment leads me to believe that many developers misuse the term “Agile” to describe any lightweight development style that is not overly restrictive. Unfortunately, the procedure they describe isn't a bona fide process. They may support iterative development (point 2), but often the first deliverable is more akin to the complete product rather than a unified portion. Their development methodology supports last minute requirements (point 1), without defining a method to minimize the amount of work it entails (point 3). As with any other project, the requirements are poorly defined, but without a system of checks and balances, such as Scrum or XP, these organizations tend to treat time as the primary factor to consider when deciding whether to implement a feature – i.e. we have two weeks left, that should be enough. In this environment, developers often make snap decisions on how to handle a particular situation without fully considering the effects on others. I have often noticed that a “small” feature that was thrown in the mix to solve a small problem gains a life of its own, and grows to require much of the development time of future cycles.

Competitive pressures, especially the “time-to-market” myth, have led many companies to abuse time-boxing – throwing together something that works, without spending the time to craft it, or plan for the future. Time-to-market is important, as small companies that don't get something done quickly don't last. But that isn't sufficient! Companies that do get their products to market are finding themselves struggling with issues of scalability and adaptability. Their product might work exceedingly well for some number of users, but not to the magnitude that customers expect. Having thrown together a quick solution, companies are finding poorly designed systems that prove resistant to change. And customers aren't buying these products for these reasons.

Other than the resulting human misery, I think the tragedy of the dot com era was the assault on software process, agile or otherwise. Had many of these companies continued to the next phase of development, I am convinced that we would have heard a lot more on the need for truly following an agile process. Instead, we have a new generation of software developers who think that processes is archaic; they feel that they wrote truly great software, but were the victims of the internet bubble collapsing.

**Paul Bramble** is a Senior Software Engineer with Emperative, Inc., and specializes in Object-Oriented Software Development. He is the co-author of the book "Patterns for Effective Use Cases, and is teaching a tutorial on this subject at OOPSLA 2002. He has more than 20 years of software development experience, including telecommunications, avionics, operating systems, mainframe computer manufacturing, and e-commerce systems for several different organizations. Paul received his MS degree in Computer Science from Arizona State University in 1989, designing portions of an Object-Oriented distributed operating system for his master's thesis.