JOIN OVER 100,000 PROS! Join Log In | Member Benefits

STICKYMINDS

Agile Testing DevOps Security AI

Agile or Not: How to Get Things Done

ARTICLE

By Steve Berczuk | October 19, 2011

SUMMARY

Agile software development always felt intuitive to me. Developing software incrementally, in close collaboration with the customer is the obvious way to deal with the uncertainty inherent in both software requirements and implementation. The technical practices of automating necessary but time consuming tests, and deploying, early and often are the obvious ways to give an team the ability to evaluate the functionality you have and to to decide if the software works as expected. And it's also important to decide if what you built still makes sense given the current environment.

Agile software development always felt intuitive to me. Developing software incrementally, in close collaboration with the customer is the obvious way to deal with the uncertainty inherent in both software requirements and implementation. The technical practices of automating necessary but time consuming tests, and deploying, early and often are the obvious ways to give an team the ability to evaluate the functionality you have and to to decide if the software works as expected. And it's also important to decide if what you built still makes sense given the current environment. Agile isn't the only way that people build software, and it may not be a perfect approach, but it's one of the best ways of dealing with a system that has unknowns.

Agile software development acknowledges uncertainty. The ability of agile methods to make it very obvious very quickly when a project, or even a process, is failing makes people uncomfortable. The visibility of the failure leads to a desire to return to more "traditional" processes. For example, a common complaint is that agile organizations don't create "good enough" specifications. Someone might blame an unforeseen problem on the lack of a spec that mentioned it. This is possible, but it's only true if the person writing the spec could have foreseen the problem.

The desire to point back to a lack of specification also points to a lack of buy-in to a

We use cookies to ensure you get the best experience on our website.

Accept

Read More

- The code was not agile enough, so the bad assumptions were embedded into the code so deeply they were difficult to address.
- Communication was bad.

More complete specifications that address all of the issues a system might encounter are one way to build software. But the best that people can do is to write specifications that address *what they know*. Quite often no one has complete advance knowledge of what a system needs to do.

There are projects where things are known with enough certainty that a waterfall process can work. People can write good specs, the team can implement to those specs, and the end result is as exactly what everyone wanted. I've worked on projects where this was true, and in these cases, the specifications were reviewed and tested as much as code might.

Even projects that seem suited to waterfall fail. For any method to be successful:

- Everyone involved needs to be committed to the approach and
- There needs to be a feedback loop to correct errors in time.

The second is point is more important than the first, since people will make mistakes. But being committed to the process is what lets people accept and offer constructive feedback. The reason that waterfall projects have such a bad reputation is that many are implemented in a way that results in problems surfacing late.

Agile methods when done well have the advantage of having built in feedback loops, so that customers and teams have ways of identifying problems early. When agile projects fail it's often because people ignore the feedback and let things degrade for longer than necessary. (Failing fast can be considered success!)

So, Agile or not, your process will only work for you if everyone works within the framework to the best of their abilities, and if you have mechanisms in place to help people do the right things. Otherwise you can blame your process, but odds are, that's not where (most of) the fault lies.

Topics: agile, development lifecycles, process improvement,

We use cookies to ensure you get the best experience on our website.

ABOUT THE AUTHOR

Steve Berczuk

Steve Berczuk is a Principal Software Engineer with experience as a manager, Scrum Master and technical lead in Boston, MA. The author of *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*, he is a recognized expert in software configuration management and agile software development. Steve is passionate about helping teams work effectively to produce quality software. He has an M.S. in operations research from Stanford University and an S.B. in Electrical Engineering from MIT, and is a Certified ScrumMaster. Contact Steve at steve@berczuk.com or visit berczuk.com and follow his blog at blog.berczuk.com.

UPCOMING WEBINAR

LETS HANG!

Log In or Register to post comments

We use cookies to ensure you get the best experience on our website.

1 comments

Keith Collyer | January 9, 2014 - 11:29am EST

Steve, excellent article. A lot of people think that the deliverable for agile development is code. It isn't, it's a working system. And that system should have exactly as much documentation (specs, user guides, whatever) as is needed. There is absolutely no reason why these cannot be developed in an agile manner as needed. Particularly if it is necessary to use a complex algorithm, that algorithm needs to be explained, how it works, why it was chosen and the problem it solves. The overall architecture (which I know is a dirty word to a lot of agilists) needs to be defined somewhere. I always refer people to David Parnas's classic paper "A Rational Process: How and Why to Fake It", which is a great explanantion.

Log In or Register to post comments

Join the crew for FREE! Access exclusive content, contribute, and more.

See All Member Benefits

We use cookies to ensure you get the best experience on our website.