

Re: How do you make decisions that optimize software quality?

Source: <http://coding.derkeiler.com/Archive/General/comp.object/2004-08/1194.html>

From: John Roth (newsgroups_at_jhrothjr.com)

Date: 08/12/04

Date: Wed, 11 Aug 2004 20:14:33 -0400

"Andrew" <analystresearch2002@yahoo.com> wrote in message
news:43cf64b0.0408111141.4cc1a6dc@posting.google.com...

> *Hi there,*

>

> *Thanks to all for your feedback on my previous question on unit*

> *testing.*

>

> *I want to step back and ask the group a broader question. Imagine*
> *you are the VP Engineering at a large software company with the*
> *mandate that you must deliver software at the absolute lowest possible*
> *total cost of development and deployment – that means, low cost to*
> *engineer but with high enough quality that post-release deployment*
> *costs are also low (limited bugs, etc).*

I think the first thing is to get an economic model, and since I'm answering this from an XP perspective, it's going to be the simplest economic model that could possibly be informative.

Post-deployment costs of defects start with lost productivity due to the defects, which in turn cascades into lost sales, etc. Those, in turn, go back to engineering costs in finding and fixing the defects, which in turn displaces other activities that could be used to enhance revenue.

In other words, I find it hard to believe that there is such a thing as a field-reported defect where the economic loss in the field is less than the economic cost of not letting it out in the first place. Other people may differ on this, however.

The second limb of this is that code quality directly affects the ease of adding additional features (and of repairing defects) in an existing code base. Remember that I'm answering this from an agile perspective: most of what is delivered will consist of code that has been extensively modified in the process of developing the product.

comp.object: Re: How do you make decisions that optimize software quality?

The metric here is velocity: how does the ease (that is to say, the cost) of adding a feature change over time. If it's stable, then you've got a code base whose quality is "good enough."

These two measures of quality are related, but not identical. For many business applications, the second (quality as seen by the developers) may exceed the quality as needed by the end users. For life, safety or mission critical applications, the reverse may be true.

- > 1) *What are the top 3–5 problems or issues be in your experience that would strain my ability to be low cost AND high quality?*
- >
- > 1a) *How do people resolve these 3–5 problems? Tools? Software lifecycle methodologies? Which of these tools and methods can be provably shown to mitigate those 3–5 problems or is it just statistical/anecdotal suggestion?*

XP teams can deliver substantial projects that have less than one field reported defect per month. See, for example,

<http://martinfowler.com/bliki/VeryLowDefectProject.html>

By can I don't mean that they manage to do so as a routine consequence of using XP! I mean that there are some real world examples of projects that do it. Regardless of some proponent's rhetoric, XP is not a magic bullet that cures every software development ill under the sun. It is, combined with experienced project management and a dedicated development staff (and possibly a secret sauce) capable of getting the job done.

There are several basic processes needed.

1. Executable (meaning repeatable) acceptance tests. These tests must be accessible to non-technical people on the customer side (including analysts, QA technicians, and documentation specialists.) That means that they can't be written in xUnit or, even worse, some variety of GUI automation scripting language.
2. Every line of code the developers write must be integrated that night. The integration and build system must run all of the passing acceptance tests to date, and all of the unit tests must run at 100%. The build must finish with a properly packaged deployable. The build system may run analysis code like code coverage metrics; if so, unit test statement and branch coverage must be at 100%, and acceptance tests coverage and branch metrics must be fairly high.

Re: How do you make decisions that optimize software quality?

comp.object: Re: How do you make decisions that optimize software quality?

3. QA must be done statistically. That is, it is the job of QA to insure that the product meets quality standards, not to discover specific defects. A failure to meet quality standards means that the process needs to be amended until it can produce a product that meets quality standards.

The interesting thing about this list is that it's not completely specific to XP. XP mandates the first two and strongly suggests the third. Most shops doing standard software engineering could put in 1 and 3 without seriously affecting their work flow. Item 2 is not possible unless the shop shifts to TDD rather than sequential design, code, test and integrate phases on each module.

The other thing about this is that it doesn't mandate a lot of expensive tools. The current practice on 1 involves the FIT and FitNesse packages, both of which are GPL.

Code analyzers (like coverage analyzers) in item 2 may well cost a bit, but they're worth every cent both from the standpoint of knowing where you're at and from the standpoint of controlling cowboy coders.

Likewise, some tool support of item 3 may well be appropriate.

> 2) *What are the top 3–5 risks (i.e. not guaranteed to be a problem, but could manifest as a problem) that would strain low cost/high ability?*

The top risk is not paying enough attention to keeping the code base sufficiently maintainable that you get a flat velocity curve. Crap code costs money. So does deteriorating design.

> 2a) *How do people prevent those risks from arising? Tools? Methodologies? Can the tools and methods be provably shown to suppress the risks or is it statistical/anecdotal suggestion?*

At this point, there are projects that succeed with just about every imaginable process, and projects that fail with the exact same processes.

The only thing I know of that's helpful across the board is to keep believable progress indicators, and the only way I know of doing that is to keep WIP (work in progress) to an absolute minimum. WIP is anything that's somewhere between "detailed requirements analysis started" and "successfully integrated, all acceptance tests pass."

Re: How do you make decisions that optimize software quality?

comp.object: Re: How do you make decisions that optimize software quality?

- > 3) *What are the top 3–5 decision tradeoffs that a VP Engineering has*
- > *to make to deliver software – i.e. not problems here, but just*
- > *decisions that have to be made between deliving in one way vs.*
- > *another?*

The biggest factor with an XP team is that it's capable of putting out a fairly predictable amount of function per iteration. Once a team has geled, it's not all that easy to add or subtract people, or change process. Increasing the amount of function per iteration is a slow process that requires a lot of attention.

The lever that can be controlled is the actual functionality that's worked on in any iteration going toward a formal release. Pulling all of the stakeholders together for a project is a problem that's fully worthy of a VP's time.

The other issue is that there will be times when the product needs major restructuring work to maintain design integrity, and that may not correspond to actual deliverables. This can be deferred for a while, but it cannot be ignored or the ability of the team to deliver at a constant velocity will suffer.

- > *3a) How do people determine the optimal decision for those tradeoffs?*

I've never figured out how a marketing department comes up with their estimates.

- > *Any input on these questions – even if only for a subsection would be*
- > *greatly appreciated. I hope it actually spurns some good discussionn*
- > *because my guess is that different people have different opinions.*

You're welcome.

John Roth