

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

*Source:* <http://coding.derkeiler.com/Archive/General/comp.object/2004-08/1352.html>

---

*From:* Ronald E Jeffries ([ronjeffries\\_at\\_acm.org](mailto:ronjeffries_at_acm.org))

*Date:* 08/14/04

Date: Sat, 14 Aug 2004 10:27:44 -0400

On Fri, 13 Aug 2004 23:19:05 -0200, Juhan Leemet  
<[juhan@logicognosis.com](mailto:juhan@logicognosis.com)> wrote:

>On Thu, 12 Aug 2004 15:41:14 -0400, John Roth wrote:

>> "Juhan Leemet" <[juhan@logicognosis.com](mailto:juhan@logicognosis.com)> wrote in message

>> news:pan.2004.08.12.18.20.57.825960@logicognosis.com...

>>> On Wed, 11 Aug 2004 20:14:33 -0400, John Roth wrote:

>>> > "Andrew" <[analystresearch2002@yahoo.com](mailto:analystresearch2002@yahoo.com)> wrote in message

>>> > news:43cf64b0.0408111141.4cc1a6dc@posting.google.com...

>>> > 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%...

>

>>> That sounds pretty arbitrary to me: esp. "every night"...

>>

>> Actually, any period will not do. There is a growing body of evidence

>> that the amount of trouble you have in integration is super-linear in

>> the amount of time code has stayed out without being integrated. In

>> other words, "integration Hell" is an easily predictable consequence of

>> leaving integration until the end of the project.

>

>OK, I mis-spoke. It's obviously not "any period" because the bean-counters

>would then have us only do one build at the very end = cheapest. OTOH too

>frequent can burn up resources running on a treadmill without much result.

Well, um, I must respectfully disagree. Integrating only at the end is not in fact cheapest. That integration will take infinitely long, and every defect put into the system over the course of the whole project will still be there.

As for too frequent, it depends. If the integration is fully automated — and if we're doing it frequently at all wouldn't it be — then its just burning integration machine cycles which would otherwise be spent in the idle loop. And the builds would be throwing off information. So it might not be wasteful at all.

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

>

>Maybe I'm thinking of larger teams which have to be structured in groups.  
>In that case, I would prefer to have better defined deliverables,  
>periodically submitted into the system build. I don't believe that every  
>design (esp. redesign) can be completed within a day. Is that always true?  
>Personally, I think for a schedule that runs out years, one could have  
>full system integrations done, say, weekly. The groups themselves might do  
>their own "integrations" against a stable baseline (working system) more  
>frequently.

What size projects are you working on?

>

>As I write this, I realize that there is no point in continuing this. If  
>you're talking "XP speak" and I'm thinking concepts and terminology over  
>some decades, we're probably not communicating well at all.

You certainly may withdraw on that basis or any other. As for decades, I wrote my first computer program for pay in the summer of 1961. So while I bring an XP attitude to the situation, I also bring quite a few other things I've learned along the way. I might have a favorite approach, but it's not for want of having tried no others.

>

>I guess I'm uncomfortable with the notion that "everyone is encouraged to  
>change anything". Sounds too much like "programming with your spurs on". I  
>happen to think there should be some thought given to interfaces and they  
>should be protected "somewhat". If they can be defined (best) then they  
>should not be changed arbitrarily, but only after consultation and  
>thought. I like the idea that everyone is "deputized" to be on the lookout  
>for quality issues. None of that "not my problem" kind of attitude.  
>Everyone has to be engaged to deliver good product in a timely manner.

The XP notion that triggered the "everyone is encouraged to change anything" includes everyone on the project working together in a single workspace, on a single code base, and programming in pairs.

In a less integrated team, I might share the concerns you raise as well.

>I haven't worked with XP (that I know? maybe some similar principles?). I  
>suppose I should investigate it to some depth. I get a little put off by  
>the "golly gee" kind of evangelism that I seem to hear. OTOH, if those  
>techniques are good and they really work well, then we should all use them.

Yes, you get to decide what you want to learn about. I applaud the notion of looking into it before dismissing it.

>

>Personally, I believe a lot of the problems of "standard software  
>engineering" methodologies are more related to their implementations. I  
>remember it related that Gane & Sarson did 2 pilot projects of their  
>(first iteration?) SSADM. One was a success (those that grasped the  
>concepts and their purpose) and one was a total failure ("hostile" group?

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

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

>just went through the motions, and generated useless "paperwork"). I'll  
>bet there are some duds out there that can make a total hash of XP, too.

Sure. It all comes down to people.

>

>If you never look at the bugs, it doesn't mean they are not being  
>generated. How many are really being fixed? I'd prefer to have some idea  
>of how many there are: say starting from a full system integration, not  
>the itty bitty ones that arise from the low level "fizz" of programming  
>and subsystem integration. How does XP deal with bugs/problems?

There are two levels of tests, programmer tests and customer acceptance tests. Programmer tests must run perfectly all the time. Really.

Customer tests are (a) defined by the customer, so they serve as acceptance tests. Team practices vary but the better practice here seems to be keeping them running perfectly as well.

Whenever a defect shows up in spite of all the tests, what I do is to write a customer test that shows the defect, then a programmer test that shows the defect, then fix the defect. Then I reflect on what other tests I might be missing, based on what I just learned, and begin getting those written as well.

The intention that I hold is to have zero defects. Whenever I know of a defect, I go right after it. One might think that one would make no progress with this attitude. But what really happens is that one makes very good progress and it's not tainted as much by defects that will show up later.

>

>I've worked with a small team that delivered near perfect (3 "service  
>calls": 1 RTFM, 1 modification feasibility question, and only 1 spelling  
>mistake bug, working at 400% load for years). The travesty was that  
>"management" did not recognize what they had and "refused" to reward this  
>form of behaviour. The project was canned. I believe the IP could have  
>been sold outright for something like \$200K but that seemed to be too much  
>trouble?!? Others since have commented (incredulously) "that was too  
>good", perhaps hinting that they would have tried to screw our budget?

>

>The point being that I think I understand what it takes. I've also worked  
>with teams that have recorded 1400 and 11000 bugs/issues during  
>integration. I can't remember how many of those "leaked out" into the  
>customer site/environment, maybe a third? half? There was a lot of yelling  
>and screaming. These same groups refused to believe there would be bugs.

Yes. Rewards notwithstanding, I prefer to work in the former situation. And you?

>

>I get the impression that the XP process does not record any problems.  
>Must start at some point? How do you know that you're not "going round in

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

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

>circles"? Are "busted build" problems recorded? Only customer bugs?

There are no mandatory recording requirements in XP. Since the team works closely together, and since they fix problems /immediately/, most of the learning is immediate as well.

Individual teams track and record the information that they find useful. I like to record huge volumes of raw data: every single test result, every single code checkin. Then, if I get interested in some topic, I'll write an analysis that scans back over all the data and asks about whatever I'm interested in. We might start thinking that Feature XYZ breaks a lot. So we scan the test results and discover that it and feature ABC are about tied for first place, with ten times more breakages than any other feature. Hmm ... let's look into that.

But I trust that the team will get the sense of any trend by paying attention much faster and better than by looking at a lot of stats.

>

>When was that not true? I would suggest that those people who couldn't  
>make "waterfall" methodology work were just doing it really badly. Now  
>maybe XP is better (but not quite the same thing as a methodology)?  
>Whatever process you choose, you have to understand the purpose of every  
>thing you do and all "deliverables", otherwise you're just going through  
>the motions, and you'll get nowhere (good). The earliest proponents that I  
>met that wanted to do "spiral development" seemed to suggest that they  
>would just spin their wheels and we should be happy with whatever they  
>happen to produce. Bloody hell! Not on my dime. Call your shots.

Well, there are, in my opinion, some fundamental reasons why waterfall methods will be inferior to agile methods in /any/ situation where both are applicable.

Whether agile methods are always applicable is another question.

>

>I have a feeling that we're probably in "violent agreement", but maybe not  
>totally viz. methods. I don't believe "one size fits all": XP is the  
>"silver bullet"? I'm mindful of Brooks and others who warn against any  
>complacency, to think that you can find one thing that'll do it all.

I don't recall anyone mentioning that XP was a silver bullet. However, if one looks at the principles and values and practices, one is likely to find things of value. Even the famous Mr Ed is now reduced to objecting to XP on the grounds that "this isn't new." We didn't say it was new. We said it was good. Check it out, it might be good. :)

>

>> I'm not sure what you're saying here, other than that every process has  
>> known limitations. You cannot arbitrarily specify scope, time, budget  
>> and quality, and expect that, by some miracle, it will be met. It  
>> doesn't work in building bridges, and it doesn't work in building  
>> software. That doesn't stop MBA trained managers from trying.

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

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

So true!

>

>*Can you suggest a good overview of XP, to put it into perspective for me?*

>*I'm not sure I have time to read several books and do some projects before*

>*I can make up my mind about it. I'm a little bit uneasy. Seems too facile.*

Everything is facile in a newsgroup.

Try [www.extremeprogramming.org](http://www.extremeprogramming.org) for one angle. Try chromatic's extreme programming book for brief. Try Kent Beck's white book for the original source. Try the pink book (Jeffries, Anderson, Hendrickson) for details of how you do it.

Or, if you'd like to start with a single technique, Beck's, or Astels' books on Test-Driven Development are nice. But they're not overviews of XP.

Regards,

--

Ron Jeffries

[www.XProgramming.com](http://www.XProgramming.com)

I'm giving the best advice I have. You get to decide if it's true for you.