Table of Contents

Fixing Defects Does Not Mean You Are Addressing Technical Debt 3

Fixing Defects Does Not Mean You Are Addressing Technical Debt

I am not sure entirely how it happens. As part of training for Scrum Teams, we introduce the concept of Technical Debt so we can reduce the harm we are introducing into our code through shortcuts, and to start a discussion about investment required to bring down the technical debt associated with our fielded products. We talk about Ward Cunningham's original definition of Technical Debt. We talk about how to reduce creation of more Technical Debt through Definition of Done. We talk about ways to reduce Technical Debt, by characterizing it, and making it visible so it can be dealt with.

The problem is that the idea of “addressing Technical Debt” becomes synonymous with “fixing defects” over time. Or at least it does in shops I've worked in. The thinking is “if we are fixing defects we must be reducing our Technical Debt” and we feel good that we are doing this. Further it leads to thinking that we can “take on” Technical Debt as a business decision and that is OK as we can plan for defect fixes in the future, determine how much it is and so on.

Lets be perfectly clear.

> **Addressing Technical Debt is not the same as fixing defects.**

Technical Debt is the poorly structured code (the 30,000 line function that grew over time), the code that people don't want to deal with (the module that no one or “only George” can touch without fear of breaking it), the code that obscures its intent, the code that is so rigid it cannot be changed, the code that is not covered by automated tests, the code that cannot be easily read, the code that still produces defects even after we've fixed the end users view because we haven't looked into the root cause, and so on. In other words it is the cruft we've built up in our code base because of shortcuts and poor decisions we have made in the past.

Addressing Technical Debt may help you in the future by reducing the number of defects. For example, if you re-factor code that is the source of a lot of support calls so that it is easier to work with, you may end up with fewer support calls on that area of code. But the reverse is not necessarily true. For example, by fixing the defect you may in fact introduce another coding short cut and so increase Technical Debt rather than reducing it. This is especially true if you take the view that you then want to manage new defects rather than simply getting rid of it while the context is fresh in your mind.

More importantly if you make changes that improve your ability to make changes and reduce the number of defects you have then you also have more time to work on items that actually produce value.

Creating Technical Debt should not be intentional. We shouldn't normally create situation where we manage bugs we have just introduced. We should write the code so there are no bugs. In particular if you leave the defect in the code that you've just worked on then this just means you've done poor work. And calling it a more formal name like “Technical Debt” won't change the fact that you knowingly did poor work in an area. As Uncle Bob Martin says "A mess is not a technical debt. A mess is just a mess."

How do we go about changing this perception? Perhaps the simplest approach is to get a group of people

in a room and have them brainstorm what they think “Technical Debt” is. If your experience is like mine, you'll get a number of people saying something like “fixing defects.” This allows you to work a more nuanced discussion about defects, technical debt, and approaches you have toward improving quality of the product.

We also need to make sure we start dealing with Technical Debt we have in our fielded products. We need an incremental approach to addressing existing technical debt, but something that allows us also to plan a bit better how we take this on. In the book The People's Scrum Tobias Mayer offers a slightly different take on an incremental approach to paying off technical debt based on some work with a team in New York City:

- When a team member comes across bad code during a Sprint they don't fix it unless it was crucial for the feature they were actually working on. Instead they note the file, line numbers, features that the poor code touched and track this as “Technical Debt”. (Note: the team did this with sticky notes, but I expect there are lots of ways of doing this).
- During Sprint Planning as the team discusses potential stories to be worked on, team members quantify the known danger areas in the code using information tracked as “Technical Debt.” We now use the policy above (“never add more debt”). Coding around existing, known problems adds debt. Taking this feature on in a Sprint would require that the technical debt is addressed as well. This would be discussed as part of the overall plan and the estimate for a user story increased based on this known technical debt information.

If the product owner still wants the feature (it may have got very expensive in comparison to initial estimates) the team would add the Technical Debt items as (sub-)tasks on the user story.

TechnicalDebt, BlogEntry, Defects

From:
https://www.hanssamios.com/dokuwiki/ - **Hans Samios' Personal Lean-Agile Knowledge Base**

Permanent link:
https://www.hanssamios.com/dokuwiki/fixing_defects_does_not_mean_you_are_addressing_technical_debt

Last update: **2020/11/03 06:55**