

4.5: Cleanup Stories

Technical Debt is so important that we must manage it directly – its management won't happen by accident or as a byproduct of development. Since Technical Debt can't be seen except by looking at the code, its presence must be made visible to the outside somehow. To do this we introduce the Cleanup Story.

Table of Contents

<i>How Technical Debt Arises</i>	1
<i>Cleanup Stories</i>	2
<i>Discussion</i>	3

If you recall from our chapter on Technical Debt, Technical Debt is an evil, invisible, killer of our product. It can't be tested for, and consists of things like incomprehensible code, bad design, incorrect or insufficient documentation, a shortage of unit tests, and so on.

Technical Debt is so evil, so insidious, that it needs to be treated differently from almost everything else that happens to us in software. There are a number of ways that we create, or find, Technical Debt, and we'll discuss them in this chapter.

Normally, when we find Technical Debt, we often ignore or skip over it because we don't have time to "fix it right now." But we must do something – not ignore it – and that something is the Cleanup story, which is the main topic of this chapter.

How Technical Debt Arises

There are a number of ways that Technical Debt happens. The most obvious is that we take shortcuts when doing work, or don't do something as well as we should have. We're supposed to follow our "doneness" Agreements to prevent this from happening, but sometimes the Agreement just isn't good enough, or else we don't do it quite as well as we should. Stuff happens. Maybe it shouldn't, maybe we don't want it to, but it does.

Sometimes we create Technical Debt on purpose, because we have to do some work and we don't have the time to "do it right." Some people say this would never happen, but I don't believe it. Imagine the Product Owner saying: "we need to deliver this

feature by tomorrow or it will cost us millions of dollars, and I don't care if you just have to hack it in!!" What would you do?

That's a rhetorical question, I hope. Of course you would just hack it in – you would intentionally create Technical Debt – because doing the “right thing” is not always the right thing to do. There could be a higher business reason that overrules quality, right? Of course there could be, and we wind up with Technical Debt. It would be nice if we didn't have to do this, and some people don't, but if you do, you've got Technical Debt.

Sometimes we just “find” Technical Debt when we are wandering around in the code working on something else. We don't know how it got there, but there it is – that's the reality of the thing. It could be simply that our concept of “good enough” has changed since we wrote the code we're looking at, but the fact is that we've got Technical Debt now.

In any case, we've got to do something about it, because we need to live our values – we need to respect our code and make the fact we found the Technical Debt visible. After all, it might not be found again until it's too late. So what do we do?

Cleanup Stories

The solution is the Cleanup story, which is a special kind of chore. Remember that chores are things we do that do not provide Business Value; they usually help us maintain our velocity. There are chores about things we need in our environment, and there are chores about things we need to do to the product. These last stories/chores are usually to reduce Technical Debt, and are the ones we refer to as Cleanup stories.

So let's say we found, or created, some Technical Debt. There is this piece of our code that's stinking up the place. We've got to live our values, and the two that are the most important here are respect and visibility; we must respect our product, and make the Technical Debt visible so that we can do something about it.

Basically, a Cleanup story is a story that apologizes to our code base about something bad that happened, and promises to fix it. For example, let's say that we hacked in some code by not unit testing and doing some fairly bad design – so that it needs refactoring. Then what we would do is add a story to the Backlog named “Fix up module XYZ” in which we would explain what unit testing needs to be done and what code needs to be re-factored.

Or say that you were wandering around and saw some terrible code in module XYZ that needs to be redesigned. Then you would add a Cleanup story named “Redesign module XYZ” where we would explain what we found and what needs to be done.

Now, you don’t have to admit that you created the Technical Debt even if you did, you just have to document that it’s there, and what you think needs to be done to resolve it. You don’t have to be a martyr or suicidal...

This is what Cleanup stories do; they make it visible on the Backlog what we have to do to bring the code quality back to where it should be. The Cleanup story always belongs on the Back Burner, so that's always right in front of us when we do our sprint planning, forcing the Product Owner to ignore it *on purpose*. It never gets put in the Fridge or Freezer or dropped out of the Backlog – It's always right there in the Back Burner making us feel bad. Its purpose is to guilt us into doing something about the problem, forcing us to repay the Technical Debt.

The basic idea here is that we must force the Product Owner to make tough choices; does he want to fix the Technical Debt, or does he want something else with Business Value instead? If the Product Owner is incapable of making these decisions, maybe the team should consider using something like the “70/30 rule” which we discuss in Chapter 3.?, which would allow the team to make the decisions about the Technical Debt stories.

Discussion

There are a number of issues concerning, and related to, Cleanup stories. The first is the definition of “done”. The idea is that the definition of “done” is good enough so that it prevents the production of Technical Debt, but that might not be the case. If you are finding Technical Debt even though you have met your definition of “done”, you need to retrospect on how to improve your definition of “done”.

As we'll discuss later, storytypes are used to help catalog the definition of “done” for various types of stories, among other things. So as we capture and improve our definitions of “done”, we need to make sure we update our storytypes so that we can document and spread the word about our findings.

In other words, our first choice is not to create Technical Debt. In order to do this we follow our definition of “done”, and continuously inspect and adapt the results to make sure that our definition of “done” is good enough. However, if Technical Debt is created or found we need to add a Cleanup story to the Backlog to document it and request it be fixed.