Managing Defects in an Agile environment
Introduction

Teams often struggle with answering the following question: “How to manage our Defects in an Agile environment?”. They start using Scrum as a framework for developing their software and while implementing, they experience trouble on how to deal with the Defects they find/cause along the way.

Scrum is a framework that does not explicitly tell you how to handle Defects. The straightforward answer is to treat your Defects as Product Backlog Items that should be added to the Product Backlog. When the priority is set high enough by the Product Owner, they will be picked up by the Development Team in the next Sprint.

The application of this is a little bit more difficult and hence should be explained in more detail.

1. What is a defect?

Wikipedia: “A software bug (or defect) is an error, flaw, failure, or fault in a computer program or system that produces an incorrect or unexpected result, or causes it to behave in unintended ways. Most bugs arise from mistakes and errors made by people in either a program’s source code or its design, or in frameworks and operating systems used by such programs, and a few are caused by compilers producing incorrect code.”

There are a lot of synonyms and explanations for a defect. Terms like ‘bugs’, ‘incidents’, ‘faults’ are all used, often for the same context. It is not always clear what we mean with all these terms and how they should be handled in terms of priorities and their place on the Product Backlog. Shortly summarized a defect can be described as unwanted behavior of a program.

Injecting passion, Agility and quality into your organisation.

RON ERINGA
AGILE COACH

After being graduated from the Fontys University in Eindhoven, I worked as a Software Engineer/Designer for ten years. Although I have always enjoyed techniques, helping people and organizations are my passion. Especially to deliver better quality together. When people focus towards a common goal, interaction is increasing and energy is released. This is the beauty of the Agile way of working.

Personally I have about seven years of experience with all kinds of Agile practices, like Scrum, Lean, Kanban, TDD, and Continuous Integration. During my experience I had roles as a Coach, Trainer, Scrum Master, Product Owner, and Agile Project Manager. In addition, I also speak regularly at various Agile conferences and seminars.
2. Why and how can Agile methodologies help to improve my Defect handling?

It is inevitable that at some point in time every software application will contain Defects. Non-realistic teams deny this situation and discover that they were wrong. Realistic teams accept this fact and focus on wondering when these Defects are caused, how long it takes before they are found, and even more important when they are resolved.

Many studies have proved that fixing a Defect found in a production stage is much more expensive (in terms of cost and effort) than fixing it when found in earlier stages of the project (NIST study). These same studies concluded that “software bugs, or Defects, cost the US economy an estimated $59 billion annually, or about 0.6 percent of the gross domestic product”.

Another study at the IBM Systems Sciences Institute states that the cost of finding & fixing a Defect in the maintenance stage is 100 times higher than when it was found and fixed in the design stage.

Combined with the relative cost graphs it becomes immediately clear that there is a lot to gain by running an project in an Agile way. From these graphs you can conclude that Agile helps bringing down the cost of late found Defects and increasing quality from the moment a new piece of software is written.

3. What is a Product Backlog Item?

**Scrum Guide:** The Product Backlog lists all features, functions, requirements, enhancements, and fixes that constitute the changes to be made to the product in future releases. Product Backlog items have the attributes of a description, order, estimate, and value.

**Themes, Epics, Stories, and Defects**

A Product Backlog Item defines all work that needs to be done to make an incremental update to your product. The amount of work related to a Product Backlog Item can differ from a few hours to months. A Product Backlog typically contains small items with high priorities on the top and big chunks with low priorities on the bottom.
Scrum does not prescribe how a Product Backlog Item should look like, since we don’t want to tell people how they should do their work. However there are a number of good practices in Extreme Programming (XP) that can be used to make the definition of a Product Backlog Item more explicit. While Scrum only talks about Product Backlog Items, the Agile community [www.mountaingoatsoftware.com/blog/stories-epics-and-themes] has more implementations for a Product Backlog Item:

- **User Story**: User Stories describe a new functionality from the perspective of the person who desires the new functionality. A User Story should be small enough for a team to pick it up in a Sprint.

- **Epic**: An Epic is a large User Story. It is a label that is used to indicate that a User Story cannot be picked up in a Sprint, since it is simply too large/complex and needs to be broken down into smaller User Stories.

- **Theme**: A theme is a group of User Stories. Sometimes it is useful to speak in groups of User Stories, for example when we are talking to customers about the content of a release.

Just like Themes, Epics, and User Stories (which are typical terms that come from XP) a team can also put Defects on their Product Backlog. This means that Defects can be just another implementation of a Product Backlog item and can be treated as a Product Backlog Item in terms of planning, estimation, and priority.

### 4. Defect, User Story, or Task?

In general we often see four different paths that lead to new work for the team:

1. **New feature**
   This is the most common way to introduce new work to the Product Backlog. One of the stakeholders wants new functionality and the Product Owner has brought up a new Product Backlog Item on the Product Backlog. It often starts as an Epic that will be split up in smaller User Stories if needed. Scrum Teams will use the Backlog Refinement sessions to do so.

2. **Changing functionality**
   User Story was accepted in a previous Sprint by the stakeholders and/or Product Owner, but for some reason they changed their mind. Since it is the stakeholder who is about to change his mind (which could happen at the Sprint Review, when stakeholders gain new insight), a change like this can be translated into a User Story that will end up in the Product Backlog.

3. **Development Issue**
   A Developer is working on a User Story but while he was implementing the Story he introduced an issue. In the most ideal situation he will fix the issue immediately, before he delivers the User Story to the rest of the team. Members in a team that uses Continuous Integration typically deliver their changes a few times a day. It could happen that instead of the Developer, one of the other team members detects the issue after it has been delivered. In this case it is wise to explicitly visualize the issue to make sure it will be resolved (typically they are entered as a task on the Scrum Board and become part of the “Done” criteria for that story). In other words the User Story in this Sprint can only be marked as done (and hence delivered at the end of the Sprint), once all detected issues are solved. A team should never be pressured to finish User Stories in a Sprint with open development issues.

4. **Defect**
   A lot of Scrum teams use the definition “Defect” for a development issue that is found after the Development Team has delivered a Product Increment at the end of a Sprint. Apparently the team delivered an Increment at the end of a Sprint but for some reason they did not detect the error. These kind of Defects are a manifestation of technical debt and an indication that the Development Team should increase their skills to deliver higher quality Increments. This is typically an indication that the Definition of Done, used by the team needs to be improved.

   Since it is often not clear what User Story exactly caused the error it is probably a wise thing to create a new Product Backlog Item on the Product Backlog. This could be a new User Story, but many Agile ALM Tools also provide an option to create a Defect on the Product Backlog. After the Defect has been added to the Product Backlog, it needs to be discussed with the Product Owner and the Development Team.
Before a Defect can be solved we first need to discover the root cause. Finding the root cause of a problem can be a very time consuming job in itself. As a result teams often find it harder to estimate a defect, compared to a User Story.

Defects might be caused by a different team/person than the person analyzing the problem (the typical legacy problem). This often results in Defects being ‘pinballed’ from one person/team to the other, before the right person/team is able to solve it. This behavior is typically seen when Scrum Teams do not have the end to end responsibility we would like them to have (for example one Scrum Team being responsible for prototyping stuff, another for building the product and another for solving Defects). This is also why we should always need to make teams responsible for the complete software application lifecycle.

When the time between the cause and discovery of a problem is substantial there is a big chance that the knowledge about the cause might be missing. This could be caused by a number of reasons like incomplete test cases, stories running over multiple sprints, ‘not-done’ User Stories (incomplete ‘Definition of Done’) or work traveling between multiple teams.

The nature of a Defect typically demands other information to be provided than we normally expect for a User Story. This is the reason that Agile Backlog Management tools often have a different template for User Stories and Defects. Information we typically need for a Defect:

- Who submitted the Defect?
- Who found the Defect?
- What is the unwanted behavior that was found?
- What scenario has caused the behavior to appear?
- Is there any logging to provide background information?
- Is the scenario reproducible?
- What is the impact and severity of the Defect?
- In what software build/Sprint was the Defect found?

In what software build/Sprint was the Defect caused?

5. What is the difference between a Defect and a User Story?

The biggest difference between a Defect and a User Story is that Defects are not about new functionality. User Stories are usually about new stuff that a customer wants (or same stuff in a different way). A defect is a result of a mistake that has been made in the past, but for some reason it was not detected.

Defects are the manifestation of technical debt in your system and they may be directly visible to your customer, which usually has a big impact on the priority (it often means they need to be resolved immediately). Teams are generally more comfortable estimating User Stories than with estimating Defects. This is due to a number of reasons:

- “Low Priority Defects” can be dealt with in a very simple way. They can be discussed during the Backlog Refinement sessions and will be handled like any other User Story on the Backlog.

- “High Priority Defects” are the ones that need to be resolved as soon as possible. These are typically the ones that block the Development Team or (worst case) the stakeholder(s) in their current activities. In the most ideal case it is an easy to fix problem that does not affect the Sprint content, but in many cases these Defects cost a lot of time to invest and resolve. So, as soon as becomes clear that a Defect has an effect on the Sprint content, the Development Team should re-discuss the Sprint content with the Product Owner.

The Development Team and the Product Owner should determine together the priority of the Defect:

- Before a Defect can be solved we first need to discover the root cause. Finding the root cause of a problem can be a very time consuming job in itself. As a result teams often find it harder to estimate a defect, compared to a User Story.

- Defects might be caused by a different team/person than the person analyzing the problem (the typical legacy problem). This often results in Defects being ‘pinballed’ from one person/team to the other, before the right person/team is able to solve it. This behavior is typically seen when Scrum Teams do not have the end to end responsibility we would like them to have (for example one Scrum Team being responsible for prototyping stuff, another for building the product and another for solving Defects). This is also why we should always need to make teams responsible for the complete software application lifecycle.

- When the time between the cause and discovery of a problem is substantial there is a big chance that the knowledge about the cause might be missing. This could be caused by a number of reasons like incomplete test cases, stories running over multiple sprints, ‘not-done’ User Stories (incomplete ‘Definition of Done’) or work traveling between multiple teams.

The nature of a Defect typically demands other information to be provided than we normally expect for a User Story. This is the reason that Agile Backlog Management tools often have a different template for User Stories and Defects. Information we typically need for a Defect:

- Who submitted the Defect?
- Who found the Defect?
- What is the unwanted behavior that was found?
- What scenario has caused the behavior to appear?
- Is there any logging to provide background information?
- Is the scenario reproducible?
- What is the impact and severity of the Defect?
- In what software build/Sprint was the Defect found?

In what software build/Sprint was the Defect caused?

Shortly summarized, managing Defects is often perceived as more difficult than managing User Stories, since Defects tend to have higher priority and are more difficult to estimate.
6. What if high priority Defects are disturbing our Sprint goal?

“Our team has a lot of high priority Defects during the Sprint, which makes us very unpredictable. What to do about this?” I have had this question many times, so let’s elaborate on it a little bit more. In such a situation a team should ask themselves a number of questions.

Prioritize your Defects

“What is the reason that we experience this amount of Defects?” Apparently something happened in the past that caused a lot of Defects. A lot of teams keep building new functionality while not solving this problem.

When too many Defects pop up during your Sprint you are probably in need of a corrective action to increase the quality of your software.

“Are all these interrupting Defects really that high priority, or can they wait until next Sprint?” If you put your Defects on the Product Backlog, discuss them in your Product Backlog, and Prioritize them like you do with User Stories the problem gets the attention it needs. Very often, teams do not really challenge the Product Owner to use the Product Backlog as the mechanism for planning Defects. Instead of putting them on the Backlog they handle them during the Sprint, disturbing their planned Sprint goals.

“But, what if we still have Defects with high priority that need to be fixed immediately?” There is always a balance that a team needs to take care of during a Sprint. There are always unexpected surprises that can interrupt your Sprint goal. If your team knows that they can handle a number of these High Prio Defects without disturbing the Sprint goal there is not a real problem. As long as you make sure these disturbances are in control by the team. A good way to ensure this is to use a swim lane on your Scrum Board that visualizes all disturbances. If you use the Kanban principles for limiting the amount of you can make sure the teams velocity will not be affected. And if the amount of High Prio Defects in a Sprint goes up, that might be a first indication of a possible problem kicking in. So this means you should start planning your Defects through your Product Backlog again. The team should set a boundary to the acceptable amount of High Prio Defects in a Sprint.

Visualizing Defects

When teams are creating too much technical debt they may reach a point where they need more time in a Sprint to work on technical debt related Defects than they create new functionality. This is typically the point where drastic actions need to be taken to get back on track. For such a team it might be a good idea to start visualizing the debt they are creating, so they can see the size of the problem they need to solve.

Example

The example below shows a graph that was actually used to visualize the amount of Defects we had in a team.

This graph reflects the number of Defects for over a one year period. What you clearly see is:

- That the team started with 50 Defects and they kept increasing (building technical debt). After almost half a year the technical debt became too big and the team realized that they needed to fix the situation.
- It took almost 2 months to get the situation under control.
- It took the rest of the year to bring the debt back to a minimum.

Once, the team was back in control of their quality and technical debt, they inspected themselves, and adapted their way or working. What this team did is:

- Introduce a threshold of 40 Defects and did not allow the amount of Defects to go above. Once they went above 40 Defects they talked to their Product Owner, planned more Defects, and less User Stories in the next Sprint.
- Improved their ‘Definition of Done’ with writing more and better automatic tests.
- Added another person to the team with test knowledge and capabilities, so the team had more focus on quality.
Another metric that might help your Development Team and Product Owner in a situation like this is to visualize how much time the team spends on "new features", "changing functionality", "development issues", and "Defects".

This graph shows the Velocity of a new team over a few Sprints:

- The Velocity of the team became stable after five or six Sprints.
- The team is quite stable in its ability to work on new features each Sprint.
- However the team is more and more working on Defects every Sprint.

So although the Velocity is stabilizing, this might be an indication they’re building technical debt, since the balance between Defects and other work is getting worse.

Conclusions

- Make sure that your team is not accumulating technical debt by visualizing it and acting on it. If you already have accumulated technical debt, make a plan to fix it, since your team will become slower.
- Put your Defects on the Product Backlog and handle them in the same way as you do with your User Stories.
- For High Prio Defects make sure they don’t affect your Sprint goal and when they do, act on it, and discuss them with your Product Owner.
- Make sure that the tool you use to manage your Product Backlog can make a distinction between Defects and User Stories in order to be able to present such graphs.

7. What’s the effect of Defects on a teams’ velocity?

Your Velocity should not be affected by the amount of Defects your team produces, provided that the Defects are put on the Product Backlog. Teams that use separate Defect-lists besides the Product Backlog often think their velocity is affected. What really happens here is that the teams Velocity stays the same, but the ability to work on New Features is affected by the amount of Defects they have to solve. A stable Velocity is a very useful metric in a situation like this, because for every Sprint you can predict the balance between your Defects and other Product Backlog Items.

What you clearly see is:

- They only once crossed the threshold and were able to fix the issues within a Sprint (2 weeks).
- They never had the same amount of Defects again, since they were inspecting and adapting.

8. We already have a Defect Tracking system and want to setup a Product Backlog. What now?

Many organizations are already using Defect Tracking Systems at the time they start working with Scrum. What often happens is that the Product Backlog is setup with a different tool, since the Defect Tracking System is only there for Defects. However, if you look at the nature of the work in Defects and User Stories, both should be added to the Product Backlog and both should get focus in the Backlog Refinement sessions.
Making a distinction between the tools makes sense, for obvious reasons, but the big mistake many teams make is that they keep using a different process for prioritizing and planning their Defects.

So besides doing Scrum for “New features” and “Changing functionality”, they still have tons of other meetings where they estimate, plan, and prioritize their Defects.

The trick here is to use one single approach (the Scrum approach is to do Backlog Refinement) for Defects and User Stories, so the teams can keep focusing and are not bothered with two different processes, but still use different tools when this is obligatory. The disadvantage of using two tools for managing one backlog with User Stories and Defects is that it is harder to maintain and more error-prone.

Another solution could be to use the old Defect tracking system for managing the complete Backlog. This could be an option when the old Defect Tracking System supports the team in their day to day Scrum activities.

Did you find this article interesting and do you have the need to learn more on this topic? Send an e-mail to r.eringa@pawareness.nl.