Agile development and functional testing: friend or foe?

Author: Tom Vercauteren, Functional Tester, SD Worx, Belgium

Abstract

Agile Development methodologies (in our case: Scrum) often assume that the customer and developers work together so closely that there is no need for a functional analyst or tester to be present on the team. In real life, our customer does not always know what he wants, so you do need an analyst, and our developer occasionally makes a mistake, so you do need a tester. This document explains how you can fit “functional testing” into the Scrum methodology, without too much overhead, or deviation from the basic principles of Scrum. This method was successfully tested in at least 2 projects.

What is Scrum?

For those of you who have not yet heard of Scrum, this is what you need to know: Scrum is a time-boxed methodology, that uses time-frames of typically 2 to 4 weeks.
During this time, called a Sprint, a team of developers programs one or more “User Stories”. These “User Stories” are things that the Product Owners wants to be able to do with the new software.

For example: During Sprint 3 we will develop these User Stories:

- The customer wants to check the balance of his account
- The customer wants to withdraw money of his account

That’s all we start from. As the Product Owner works closely with the Developers, all further details will be filled in while development is in progress.

All team members are present at a short daily meeting during which all issues, and To-Do’s are discussed. This is called the “Daily Standup”, because we don’t use chairs…

For more (accurate) information, look around on stickyminds.com

User stories

Acceptance Criteria

As a tester, all we get is the User Stories, and when they’ll be done.

See the example above: During Sprint 3 we will develop these User Stories:

- The customer wants to check the balance of his account
- The customer wants to withdraw money of his account

We know that "Sprint 3" ends in 4 weeks, so that's when these User Stories will be "done", which means that we could theoretically go live with them. This also means we need to have them tested by then!

As a tester, I need more information than "The customer wants to withdraw money of his account" (and so does the developer), so we start asking questions:

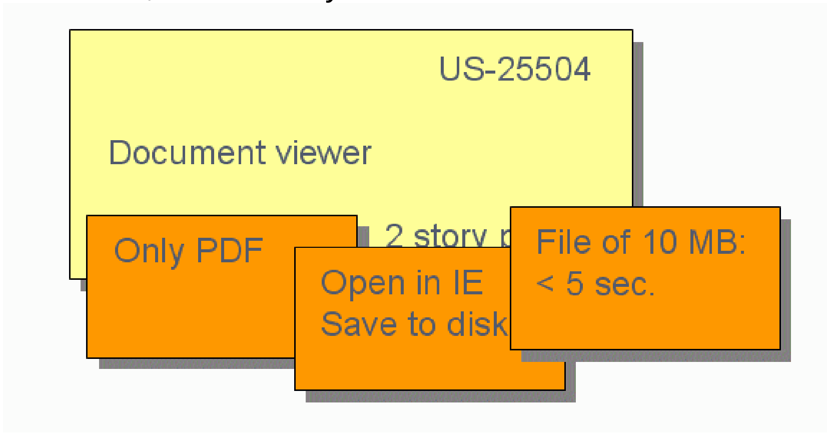
- What if there is no money on the account?
- How can we be sure that it's his own account he is withdrawing money from? How is this secured?
- Where can you withdraw money? Only at the ATM of your own bank, or at any ATM?
- …

The answers to these questions ALWAYS surfaces during the Sprint, whether it is the tester, the developer or the analyst that asks the question. The problem is, it's almost NEVER formally captured…

This information is called "Acceptance Criteria", and it should be added to the user story it belongs to!

We use yellow sticky notes to write the User Stories on, and we attach orange sticky notes with this extra information to the original note.

In the end, a User Story looks like this:



And at the end of the Sprint, we add this information to our documentation.

Test Plan

1. A User Story should be tested as soon as it is developed.

2. Up to the last moment, new information about the Story can be discovered

This means that even at the last moment, you never have a valid “Test Plan”. There is never time for:

- writing a rigorous test plan
- creating in depth, fully detailed test scripts
- running a complete set of regression tests (manually)

You can only keep up with the work by working Agile yourself!

- Only write the documentation that is absolutely necessary!
    - Don’t: “press button X, fill in field Y, …”
    - Do: “withdraw money from account that contains no money”
- Use “exploratory testing”, e.g. write down your test cases as you execute them.
- Use the Acceptance Criteria.
- Regression Testing needs to be automated.

Sprint Backlog

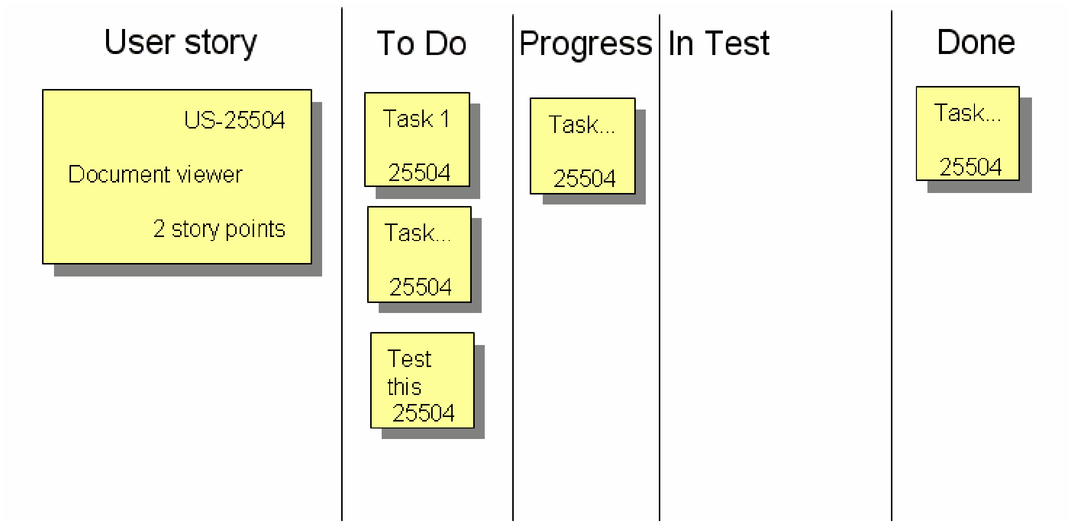
Bugs

A Sprint Backlog gives us an overview of the work we have to do during this Sprint. Each User Story is broken up into tasks, and these have a status: to do, in progress, done, …

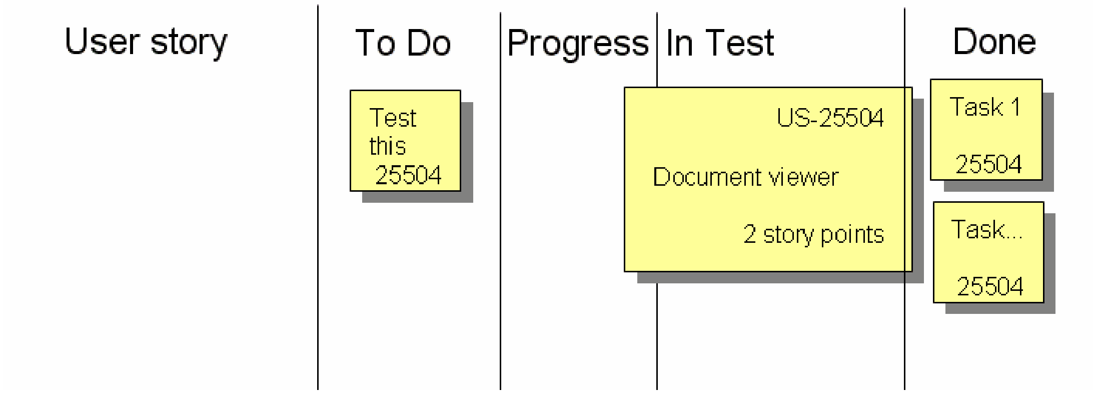
Not all User Stories are going to be tested:

- A “Spike” (a short – often technical – research) will not be tested
- Technical User Stories (stuff like “set up the test environment”) will automatically be tested once the functionalities are tested.

Eventually, our Sprint Backlog looks like this: For User Story “25504”, we have 5 tasks, 3 “to do”, 1 “in progress” and 1 “done”



The fourth column, “In Test” is not standard Scrum.
When a User Story has completed all tasks, except “Test this”, then all development is done, and the Tester can do his thing.
At that moment, the sticky note of the Story is moved to “In Test”:

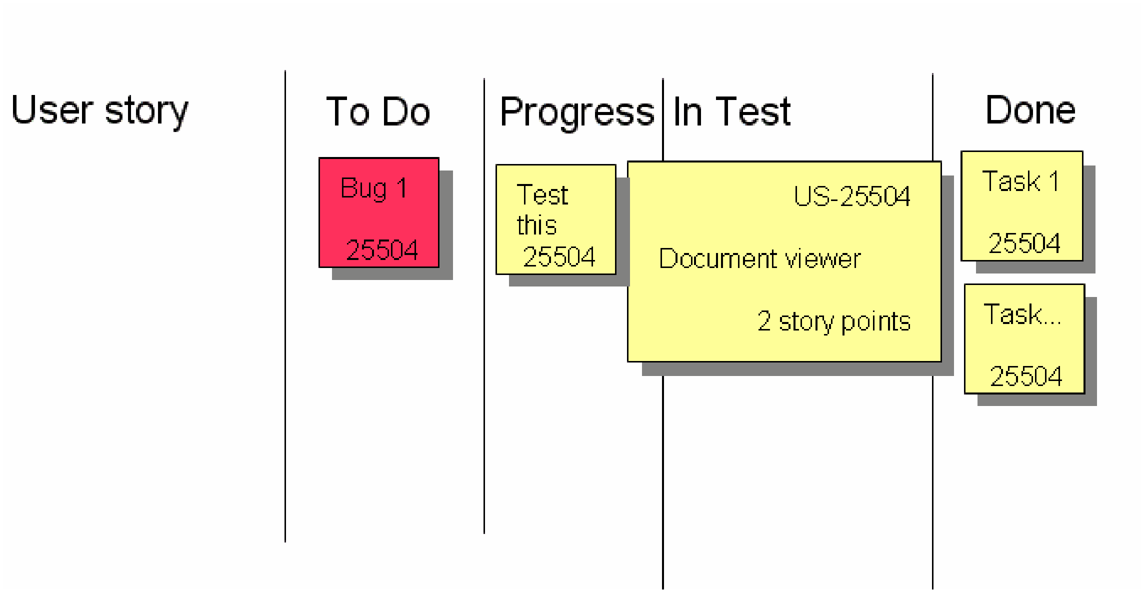


At that moment, the Tester should:

- move the task “test this” to “in progress”, as soon as he starts testing.
- add “red sticky notes” for all the bugs that he finds

Bugs are reported during the Daily Standup, and the team decides one of these:

- to fix this during this sprint
- to add it to the product backlog, for later fixing
- that this is not a bug



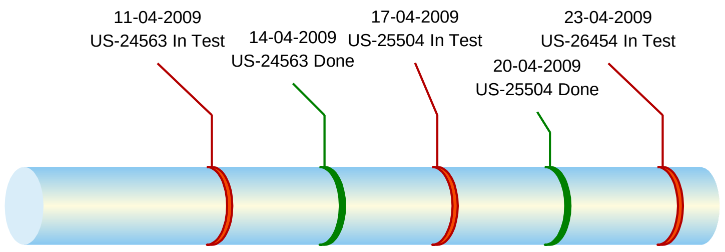
As soon as a bug is fixed, the developer moves the note to “in test”, and adds some comments: “in what release is this fixed?” and “what was the problem?”.

As soon as the tester verified that the bug is fixed, the note can be moved to “done”.

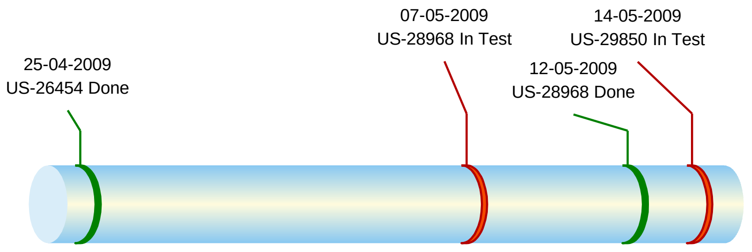
When all tests have been executed, and all bugs are fixed, the User Story is “done”.

Release Management

The trouble with Scrum is that most of the time, the last User Story that we committed to develop, is testable only a few days before the end of the Sprint.
In the example below, User Story 24563 can be tested by April 11th, and is successfully tested by April 14th. Story 26454 is testable by April 24th, but our Sprint ends on April 25th…

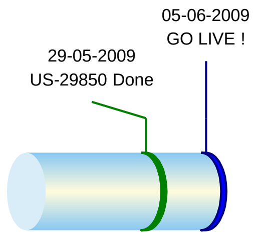


No problem, we just continue testing during the next sprint:



The trouble is: when can we go live? When is our product stable enough?

We need to add a short sprint, during which we only fix bugs:



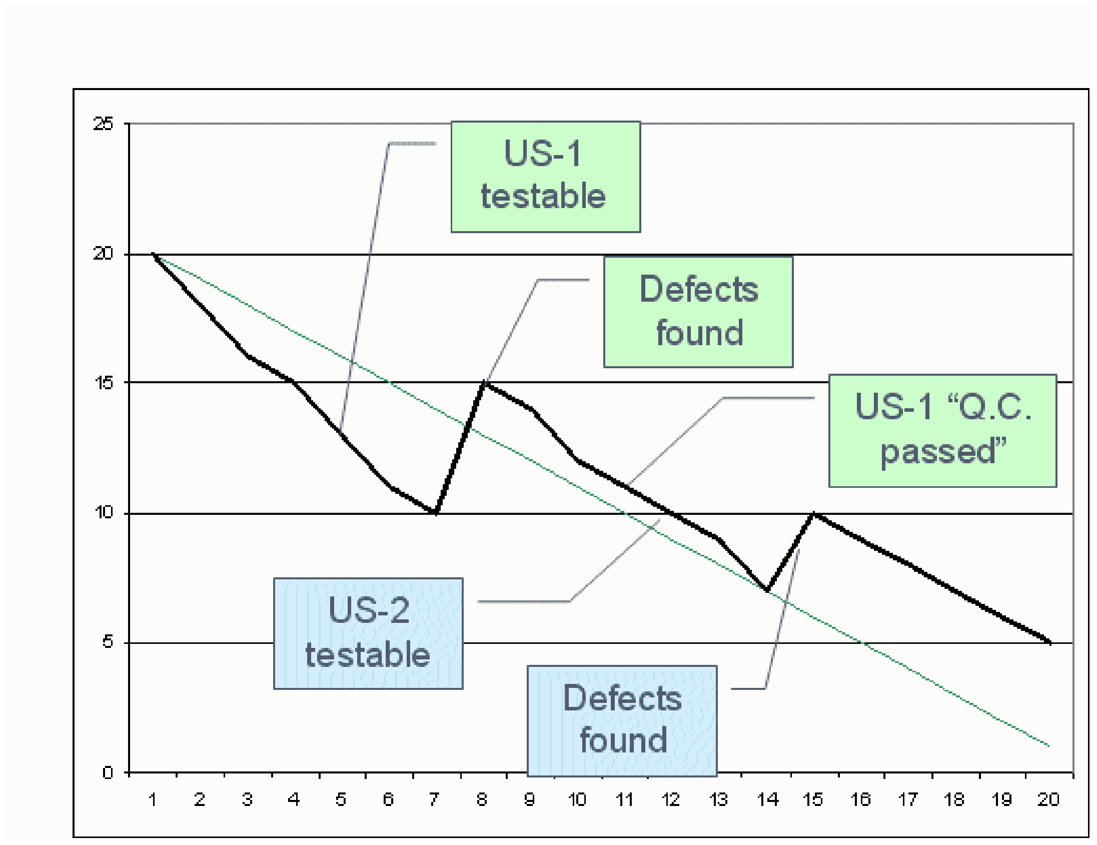
The added bonus of this short sprint is that while the tester is hard at work, the developers have time to finish the documentation.

Daily stand-up

All team members should be present at every daily stand-up meeting. That includes the tester. If, like me, you happen to be a “shared tester”, and you’re working on 2 small teams (a tester can service on average 5 developers, so I ended up with 2 teams of 3 developers), that means you have two daily meetings of 15 minutes each. If for some reason you can’t attend each meeting, the team should agree on a fixed weekday, on which attendance is mandatory (for the tester, the developers, the customer representative, the project manager, …).

Sprint burn down

Whenever a bug is discovered, you add a “red sticky note” to the task list. Unfortunately, this causes the “Sprint Burn Down chart” to go up:
(This chart visualizes the number of tasks still remaining in the sprint. The green line is the “expected number of tasks finished each day”)



As you clearly see, the black line never hits the bottom by the end of the sprint. This can be a bit demotivating, so we fixed this:

- at the start of the sprint, the team determines the task that need to be done, and create yellow sticky notes for each of them.
- for 15% of these yellow notes, we add blanc "red sticky notes" and we already count them when we draw the green line.
- whenever a bug is found, a red note is used. We found that the presence of these red notes actually encourages developers to start using them too, so this adds transparency to our process, which is what Scrum is all about.
- if we have red notes left at the end of the Sprint, we did a good job, and just chuck them in the "done" column.

The Demo

At the end of each Sprint, there is a meeting where whatever is finished in the product is demonstrated to the entire team. This is not the same as acceptance testing! I actually saw one product owner sitting there with his hands in his pockets muttering "Yeah, it works great. This will do fine by me. Let's go live with this…"
No "real user" had actually even seen it, and he never even actually pressed the button…

It is my opinion that the product owner should conduct a formal acceptance test, preferably the day before the demo (so he can present his conclusions there). You might want to cover this issue by having the product owner formally sign off on acceptance of your product.

No user interface?

Some user stories have to be tested, while they don't actually deliver a user interface. Today, more and more software is developed as a service, and later on someone may or may not slap a user interface on it. If your team does not deliver a UI, you should ask them to create a "test application for testing purposes" at the very least. Do remind them that this test-UI should NOT do any validation! If a function expects an integer, the tester should be allowed to hand it a text, just to see what the function does with it!

Definition of DONE

When is something done?

A task is done when it is tested by the developer. He'll typically have automated unit and integration tests in place.

A user story is done when it is functionally tested (to the best ability of the tester, using mostly exploratory testing), and the team agrees that there are no significant defects left.

A defect is done when it is resolved by the developer, and the tester re-tested it, found no remaining issue, and closed the defect.