FRESHER VIEW ON KANBAN AND TESTING

DIFFERENT APPROACHES TO TEST IN AN AGILE CONTEXT

Keywords:
Flow-driven delivery
Testing during story Implementation
Rigor in testing and developing

Abstract: *this article we present practices that many of our customers use when implementing a Story or high-level requirements during a sprint or in a flow-driven delivery. This is different from the usual* ***Kanban*** (1) *board, sequencing the various development stages (Ready, In analysis, In dev, In test, Done).*

Those customers need or prefer to add rigor into their delivery flow to augment the quality of their products and solutions. While this approach cannot be used universally, we found that this is quite efficient in some case, especially when tooled with XStudio.

Summary

Introduction
A user story, any story, is a set of predefined tasks
A story is then delivered one task at time
Quality is built during the development, not verified after

Introduction

There are many books and many more videos addressing the "how to deliver using a Kanban style board". This has brought a lot to the software industry when used by teams that were ready to adapt it to their context. Adaptation can take different forms.

One of our customers stated: "*Working with Kanban is like creating a tiny waterfall! If you look at it from a high-level, this is about sequencing waterfall tasks at lower granularity! In our organization, we do not have senior, highly experienced developers... we must MAKE them. So, we need to be much more rigorous and drive them more.*"

So much for agility and flow-driven development... the "Empire strikes back". Well... this is not so. It turns out this customer, as others we found later, does use Kanban board, in a very efficient way for its context.

This paper relates how this customer adapted its organization and how XStudio helped him in doing so.

A user story, any story, is a set of predefined tasks

A user story (2) is basically a high-level functional or technical requirement for your solution. It expresses a desired capability and how the users or system will probably use them.

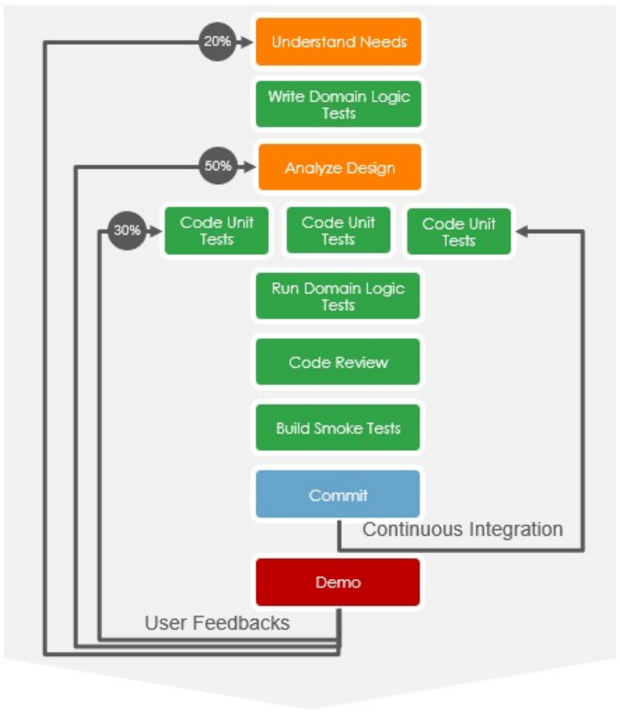

Figure 1: Developer tasks for a story

When a development team addresses a story, there is a set of tasks that a developer MUST do:

Understand Needs

Read the story, get a direct conversation as close to the end-user as possible, rephrase, draw, until you can...

Write Domain Logic Tests

Design some tests to check the story by yourself. Design at least a positive test, a negative test, but don't try covering all test cases. The goal here is to make sure you really understand the need. If you can't write useful test (automated or manual), then you are clearly not understanding the need and you must iterate to the "understand the need" task.

Analyze Design

Make your impact analysis and design. This is time when you need to ensure you know which code, which files and artifacts you should check out, which classes, methods, you need to alter or create and resolve contention with your peers about the data model, should you need to adapt it.

Code Unit Tests

This is when you create and/or adapt the code, the configuration parameters, the artifacts. There you can get several short tasks. You need to ensure the highest quality and coverage by writing and exercising independent unit tests.

Run Domain Logic Tests

Run your functional or technical tests. Use the test you defined earlier to make sure what you implemented is working and answers the needs your understood.

Code Review

Get one of your peers to review your code - don't let "code smells" go. Through-(re)factor your code now and get back to the "Code Unit test" task if needed.

Build Smoke Tests

Build and run your smoke tests on your sandbox environment. Before you commit anything, make sure it builds and has not broken anything major.

Commit

Commit and let continuous integration run.
Watch for automated feedbacks and take corrective actions.
If your solution benefits from automated tests and code analysis, you should get rapid feedback, which is easier and faster to correct while you are still immersed into the functionality you developed.

Demo

Run your exploratory tests and demo.
If you are experienced enough just do it, else do it in pair with a Business Analyst or a SME. **Iterate to whatever corrective action.**

One can say that this is an evident set of tasks any professional developer shall deliver on. Probably... but for many companies, professional developers are not the norm. Many "freshers", out of schools are to be trained before they can be productive, especially in an Agile project.

A story, is then delivered one task at time

The above model can be adapted to any context. What is important for agile project favoring visual management, is how you then make progress and blockage apparent for quick resolution.

You can then use a quite simple Kanban driven flow, for each task:
- **In Progress**
- **Blocked**
- **Done**

That's it!

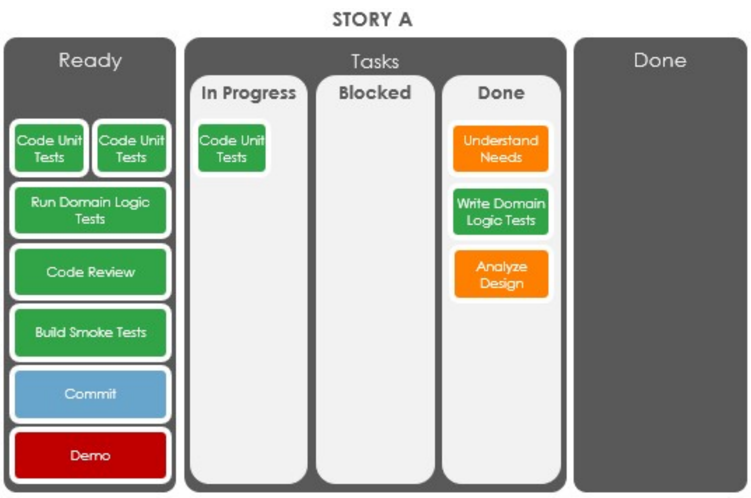

Figure 2: Progression of tasks for a story

You drive your tasks for a specific story, one at a time.
- You keep the advantage of managing Work-in-Progress limits
- You can't have more tasks "in progress" than you have makers in the team
- You can't take a new task on a user story if the prior is blocked
- All blocked task is immediately made apparent, and you can address it without even waiting for the daily stand up

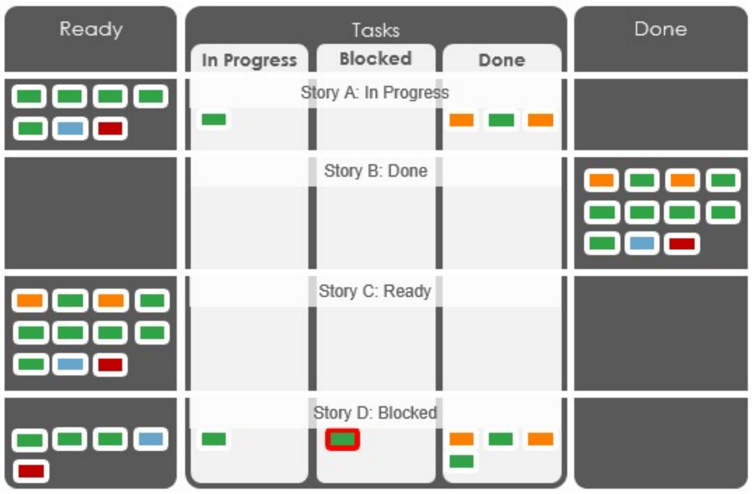

Figure 3: Progression of the stories

Quality is built during the development, not verified after

As you noticed there are multiple tasks devoted to test in this micro-process. Nothing if new here. It is just that you give the habit for the "developers" to do them.

During this process XStudio is used all along.

When a development team addresses a story, there is a set of tasks that a developer MUST do:

Domain Logic Tests

They are defined and designed in XStudio.
They can be manual or automated tests.
They can be shared with a QA team, so that they don't have to redo them.
They can be used later as smoke test.
They can be used by the business analysts or the SME to make sure you understand the needs.
They can be used for Sprint review demonstration.

Unit Tests

They can be scanned by XStudio and automatically imported to be used in regression campaigns on the integration environment.
XStudio greatly facilitates the management of those unit tests **avoiding the developer to describe each and any test.** Results are extracted from the standard outputs format of any xUnit compliant test framework.

Smoke Tests

Smoke tests and regression campaigns are easily defined based on the tests that have been created during the development.

Exploratory Tests

They are used to ensure the quality of the new features for which you may not have the time to create automated tests during the development.
Note that many customers prefer to manage test automation as stories on their own, applying the above process.
With XStudio you can turn exploratory test notes as manual scripted test using XStudio's PDNL syntax automatically.

If you are stringent on quality assurance, you can also manage your code review as test and use the defect module to follow on code review defects.

XStudio is used during all your development process.
It allows building and measuring quality as you go. You can know deliver more easily on your test pyramid, and with efficiency.

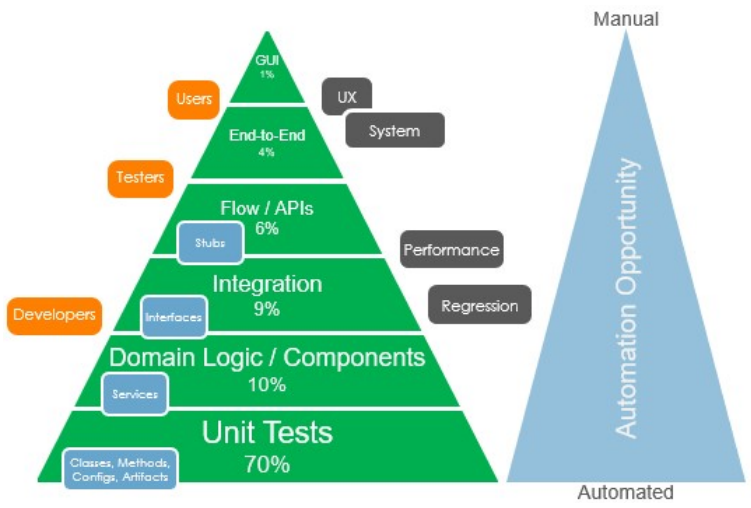

Figure 4: The test pyramid

Conclusion:

NOT ALL SOFTWARE ORGANIZATION ARE EQUALS WHEN IT COMES TO AGILE DELIVERY. SOME ADAPT AGILE PRACTICES TO MATCH THEIR CONSTRAINTS: USING AN EFFICIENT FLOW-DRIVEN DELIVERY, MATCHING THEIR DEVELOPER'S LEVEL OF EXPERIENCES AND ENFORCING QUALITY ASSURANCE PROCESS.
XSTUDIO IS VERSATILE ENOUGH TO ACCOMPANY SUCH PRACTICES, WHILE PRESERVING ALL THE QUALITY ATTRIBUTES OF AGILE METHODS.
TESTING IS AT THE HEART OF ANY EFFICIENT SOFTWARE DEVELOPMENT TEAM. GETTING XSTUDIO AS YOUR CORE TEST MANAGEMENT TOOL WILL TREMENDOUSLY ACCELERATE ADOPTING THOSE PRACTICES.

References:

① Kanban: https://en.wikipedia.org/wiki/Kanban_(development)
② User story: https://en.wikipedia.org/wiki/User_story

Figures:

Figure 1: Developer tasks for a story
Figure 2: Progression of tasks forn a story
Figure 3: Progression of the stories
Figure 4: The test pyramid