Process Critique: Iterative User Stories

Introduction

This critique explores the introduction and experimentation of iteration into user stories, and its effect on discussing acceptance criteria and research-based tasks (referred to as spikes). This trial took place within a student project with the UTS Software Development Studio and Honesty Box, in which our team designed and developed a proof of concept browser extension to demonstrate the possibility of a software only implementation of Honesty Box's existing hardware package. This project was highly explorative in nature, as neither party was sure of the limitations of this approach, resulting in deliverables that were more focused on research than design. This critique follows this project's requirements gathering, my research into the possibilities of how-to best document and distribute our teams workload, the various efforts we made into user story writing, and their effect on the overall outcome of the project.

Problem Definition

Our original user requirements came from a user story mapping session we had with the Honesty Box team at their offices. The session followed Jeff Patton's (2014) process, where as a group we wrote every task a user may take between the time they get their internet to understanding its quality. After which, they were grouped into a timeline and refined into stories and epics.

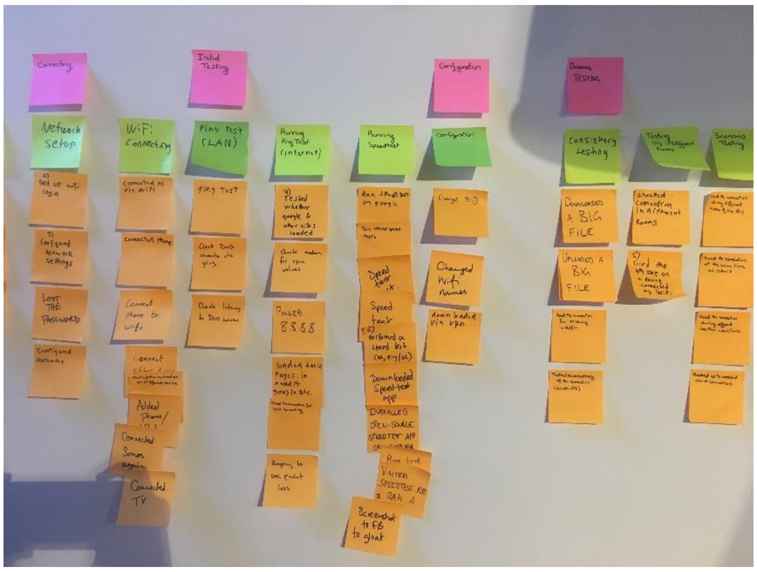


Figure 1 - Part of the timeline map showing the transition from initial to ongoing testing

We found there was a big chunk at the start we described as one-off tasks, with repeating or situational tasks becoming more prominent further in the timeline. We also found that many of the later tasks were just more complex variations on existing processes. Once transformed into user stories scoped to our project, we realised many of the requirements couldn't be accurately estimated or broken up until we had a clearer picture of the technology available and our platform.

While spikes and dev tasks could be written to document what work we were doing in the early sprints, they weren't conducive to writing verifiable acceptance criteria or providing solid deliverables, fitting the agile goals of delivering working software as "the primary measure of progress" (Beck et. al, 2001). Many of the traditional definitions of acceptance criteria (i.e. merging in code, deploying, a user-facing feature etc.) weren't available for these initial tasks, making it harder for us to be assured what we had worked on was enough to consider the task 'done'.

Moving forward, we also saw that much of the work we would be delivering would be improvements on the simple foundations we were creating, rather than completely new user stories. In certain situations, it's hard to write these stories as anything other than 'do X, but better this time'. Because of this, the management of the project was influenced by research into the best way to break up our features into deliverable iterations.

Literature Review

The traditional definition of 'done' when it comes to user stories, from Jira creators Atlassian, is simply "the user can complete the outlined task" (Rehkopf 2019), which works for well-defined instances, but does not consider degrees of quality. Depending on who you ask, agile either assumes that you get whatever you set out to do write the first time, or you fix up any past imperfections as you go in other user stories. This principal is explored in the concept of technical debt, where due to time constraints in a project, problems accrue that can cause usability issues, or extend development times. If not dealt with correctly, "mess builds, the productivity of the team continues to decrease, asymptotically approaching zero" (Martin, R.C., 2009). This idea can be applied to all iterative approaches to software, where developers often will make UI improvements as they go, but never document them as explicit stories that add user value.

Due to the user centred and explorative nature of this project, many of the initial requirements were either non-functional or hard to quantify (i.e. how to best get a user to install the software with ease). Many of these requirements fit closer to user-centred design, and as such much of the literature with iterative user stories focuses on user experience or interface (UX/UI). There has been a considerable effort into the development of a hybrid user-centred design and agile software development system (UCASD), an element of which is including the iterations of non-functional requirements into agile user stories (Brhel et al. 2015). This review outlines an issue with having the designers and software development as separate entities, and lists the difference in documentation as a leading issue. Da Silva et al. (2011) brings up the idea of "parallel sprints", in which a user centred design team works in parallel but separately to the software development. While this project is one team with no strong differentiation between UI and developers, a potential hybrid that documents iterative changes as an equal but separate tracking system is a possibility. A goal of iterative user stories is to keep them in line with existing documentation, however this does not mean we can't hypothetically treat them as an extra category (i.e. user story, dev task, bug, iterative story).

The paper most in line with the goals of my research focused on iterations in UI design (Ferreira et al. 2007). This paper analyses four projects on their ability to iterate on their UI, and discuss how many developers don't treat UI with the same mindset as they do user stories, instead focusing "on what we might call "UI stories", where these form coherent elements of a large design". The paper also brings in useability testing as a metric for acceptance criteria. While in a large project with a QA or testing team this is excellent, we are limited to just ourselves and occasionally a review from the client. However, we can still use the ideals of usability testing to drive how we think about acceptance criteria. One of the project rejects the need for user stories altogether for iterative development "They were too elaborate to be like user stories, they have too much implementation specific design already in them, so we kind of let them be and phased them out and reformulated concrete tasks that would be easy to manage within iteration". This project is outlined in the paper to be less agile leaning than the others, however it is possible the reliance on agile developers needing to fit everything to the box of a user story could be seen as more of a limitation than a mindset. Adapting these UI concepts into other areas, user acceptance is more of a discussion with the team than a concrete set of instructions.

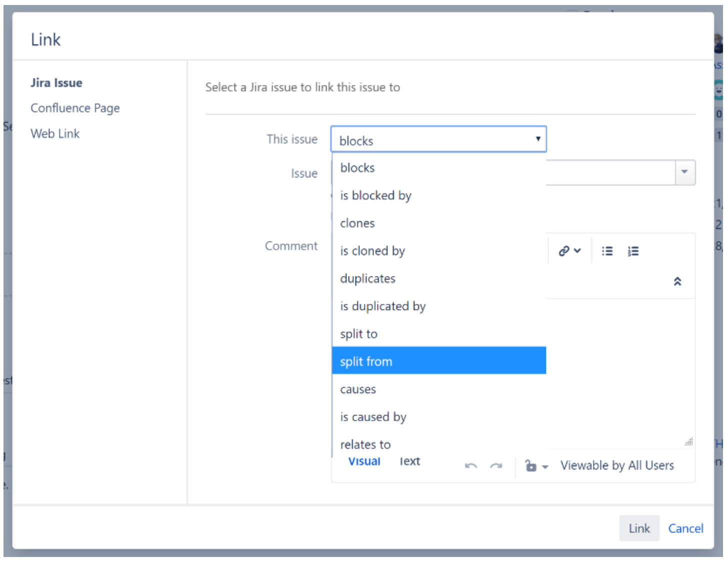
Implementation Plan

Based off the problems we faced in this project, and the suggestions given by the literature, our plan for developing iterative user stories had to meet these requirements:

- Have a historical log of previous iterations of development to the feature.
- Iterations are not subtasks, as they need to have concrete deliverables that add stakeholder value as all user stories should.
- Each iteration has its own acceptance criteria, ownership, and point estimation, essentially acting as it's own complete user story.
- This documentation is meant for the whole development team, not just UI/UX designers.
- Acceptance criteria can be a discussion, not definitive requirements. I.e. the outcome of a research iteration can be a prototype of the next, solidified iteration.
- Iterations must relate to one feature and work consecutively. If two iterations of the same feature can be done concurrently, then they can be considered separate user stories.
- There must be a way to log the card as complete, while still leaving them open for future iterations.

Effectively what we required was a 'done for now' column in our sprints, meaning they had achieved a deliverable level of work for the time being, but would move the card back into the backlog as the next iteration, letting us come back to it with new requirements when needed. Of course, this would be in a currently non-existent system, so for our project we stuck with Jira and worked it to fit as close as would effective.

A proposed work around within Jira's existing setup was linking iterations of cards to each other. However, despite it being available, we instead went with manually writing the iteration number in the title, with notes in the description outlining any major links, as Jira's linking was neither a streamlines nor visible enough system to be of much use to us.

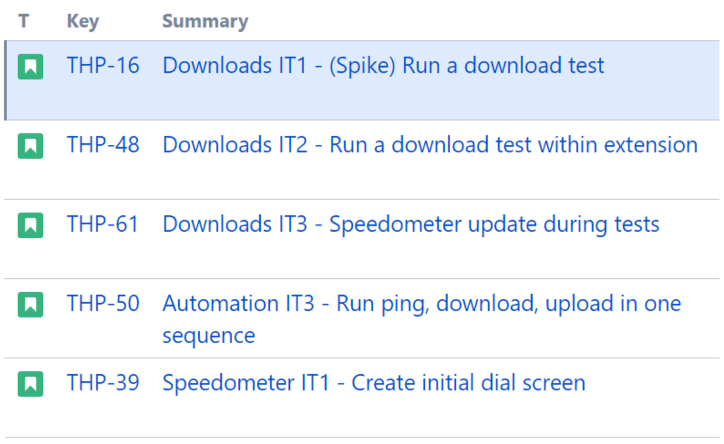

Select a Jira issue to link this issue to

This issue: blocks

Figure 2 - The card linking modal within Jira

Outcomes

Adapting our user stories to match the final iterative format was not an immediate process, as much of the project had already been completed before we solidified. It was much more a shift in mindset than a direct effect on our documentation. We noticed that many of our existing user stories, especially the ones in the UI and test epics could be retrofitted into this format, demonstrating how this project fit this iterative concept. As an example, many of our spikes from the first sprint were eventually rephrased into iteration 1 of the user stories that came later.



T	Key	Summary
🟩	THP-16	Downloads IT1 - (Spike) Run a download test
🟩	THP-48	Downloads IT2 - Run a download test within extension
🟩	THP-61	Downloads IT3 - Speedometer update during tests
🟩	THP-50	Automation IT3 - Run ping, download, upload in one sequence
🟩	THP-39	Speedometer IT1 - Create initial dial screen

Figure 3 - Example of how iteration cards have been documented

While I believe the idea behind our final approach taken would allow for a clearer understanding and documentation of iterative features, due to this concept only being introduced towards the back half of our project, as well as the limitations of Jira in documenting this, we found that retroactively going back through our project was documentation for its own sake, and not beneficial beyond an example level. We did, however, discuss many of our stories moving forward in this iterative manner, which was a great improvement to our sprint planning and product discussions.

References

Patton, J., 2014. User story mapping: discover the whole story, build the right product. " O'Reilly Media, Inc.".

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R. and Kern, J., 2001. Manifesto for agile software development.

Rehkopf, M. 2019, User Stories | Examples and Template, viewed 05/10/2019, <https://www.atlassian.com/agile/project-management/user-stories>

Martin, R.C., 2009. Clean code: a handbook of agile software craftsmanship. Pearson Education.

Brhel, M., Meth, H., Maedche, A. and Werder, K., 2015. Exploring principles of user-centred agile software development: A literature review. Information and software technology, 61, pp.163-181.

Da Silva, T.S., Martin, A., Maurer, F. and Silveira, M., 2011, August. User-centred design and agile methods: a systematic review. In *2011 Agile Conference* (pp. 77-86). IEEE.

Ferreira, J., Noble, J. and Biddle, R., 2007, August. Agile development iterations and UI design. In *Agile 2007 (AGILE 2007)* (pp. 50-58). IEEE.