A Technical Story

Nick Robinson
Lab49
nick.robinson@lab49.com

Abstract

Across the Agile community there has been a lot of discussion regarding the use of technical stories. While the community seems split into two camps of for and against, the majority of extreme programmers favour to define the system using only the traditional customer focused user stories. In some cases the technical story arguments are academic, but our experience report demonstrates clearly why sticking to user stories has its benefits. Our experience using Scrum and XP has been that allowing technical stories into the process can de-rail the ability to deliver constant, measurable business value per iteration, as well as inhibit the planning process. Once technical stories are allowed, the customer and business can be alienated very quickly, especially when they have no engineering background. Furthermore, when problems occur and the technical stories begin to slip, the customer can be left in a very difficult position, not knowing how to re-prioritise or direct the process. It is therefore our intention to highlight some of the pitfalls that can result from using technical stories.

1. Introduction

Since August 2006, my colleague and I from Lab49 have been working alongside an established development team working on a front-office risk reporting/trading platform, helping it to make the transition to agility. When we first arrived the team already had some exposure to Scrum and eXtreme Programming (XP), so our role since then has been to help improve the understandings of agility and application of the practices and process. From August to Christmas we observed how the team was working (also working alongside them), and offered knowledge and helpful suggestions to improve their process where we thought there was clear value to be realized. However the implementation was not thorough and some discipline with the process was needed, so the team made the collective agreement to begin 2007 with the extra commitment to the process.

Our experience report straddles December 2006 and January 2007, because in the penultimate iteration planning session of 2006 the team decided to introduce two technical stories, and it is the effects of this decision that this report is based.

2. Background

Looking at where the team has come from at the start of the transition to agile development, it is clear it has moved forward in many positive ways towards improving the way it works and in embracing the values of agility. In our experience, teams new to agility will find some aspects more challenging than others, and this was true with our present team with respect to the function and nature of user stories. One of the key observations we made early on in our engagement was that the format and content of their stories were very technically-oriented, rather than customer-oriented. Our position on this was that technical stories can not only be problematic in a customer-centric process, but with a little effort they can typically be converted into vertical, customer-focused user stories, which also has a positive effect on the size of the work behind each of the stories. At the same time, estimations were erratic and arbitrary, and one of the key reasons for this we believed to be the format of the stories.

During the period up to Christmas the team tried its hardest to break down all work into canonical user stories as we suggested. But in the first iteration of December 2006, having heard our suggestions about technical stories, the team decided that two stories simply had to be technical stories and that we should try them. The customer proxy, previously a project manager for the same trading product, was unsure about the use of technical stories either way, since he was still learning along with the team. So the situation was such that it seemed the democracy within the team was good enough for us to try the technical stories and to review how it goes - an innocent child, out of
curiosity and guile, must first touch the fire to realize it burns.

3. The technical stories

The two technical stories were very different pieces of work, so it will be useful to quickly review them here before continuing.

3.1. Add configuration service

The first story was entitled Add Configuration Service, which as a result would allow all of the settings across the whole application to be pulled down from a settings service sitting on the enterprise bus. The existing implementation stored and retrieved the settings from local xml files, but different parts of the system accessed them in their own unique ways. Being a C#.Net project, there are .Net framework classes to support application configuration, but partly due to their ineffective use with test driven development, a number of abstractions had been introduced to avoid direct access to the .Net framework classes. However in our short time with the software we had already experienced at least three variations of settings and how they were accessed – none were consistent.

While the system consisted of no more than five fundamental risk reporting screens, there was still a lot of code and infrastructure needed to bring them to reality; this is not a trivial system. Furthermore, the system had been built from the ground up to be completely configurable, so the number of settings that existed in the system was vast. Thus a story to introduce the configuration service into the application, by placing all of the local settings into the back-end service, to us seemed like a large piece of work.

Add Configuration Service was a profoundly horizontal technical story, which needn’t have been. The actual requirement as specified was to provide some ability for the system administrator to make changes to a trader’s set-up. In this imprecise definition it could be deemed to mean the whole of settings, but even so, this work could still be focused differently. On hearing the initial requirement we dug deeper with our analysis, and the reality was that the customer proxy would have been happy to configure just the two main risk reports, which would have been a better approach. At the same time, technical designs abound, the team saw this as a technical story since they believed it was necessary to put all of the settings into the configuration service in one piece of work, seeing our suggested logical decomposition of the story as offering little value.

3.2. Split hedging screen

A few weeks into the engagement we made a suggestion to introduce some form of layering in the user interface, highlighting the benefits of the Model-View-Presenter and Presenter-First patterns. The first major screen within the system that implemented the Presenter-First approach was the exotics-based hedging screen.

Early versions of this screen benefited greatly from the separation of concerns we had been successful in injecting, but as more features were added, the design began to rot. The project technical lead made some subtle but useful changes to the way the screens were being constructed at runtime, which had the side-effect of improving much of the layering design across many of the screens. However the one screen left to update to the new paradigm was the hedging screen.

![Figure 1 – Hedging Screen Layout](image)

Figure 1 shows the composition of the hedging screen as seen by the user. Section A is the main risk report used to support the hedging decision process, section B is a summary of the data shown in A, and C is a panel full of options used to change the data shown in A and B.

The objective with the split hedging screen story was to implement the new framework developed by the technical lead in the previous iterations to support reflection-based screen definition and hydration, much like Microsoft’s XAML.

The nature of this story was different to the configuration settings story, and initially it seemed like it could be split across the sections A, B and C. However, no behavior was changing and no user requirement was driving the story. To be fair in our appraisal, the hedging screen had caused some pain to the team previously, particularly related to code-rot...
and difficulties working with the code. But at this stage in the project no more hedging features existed on the product backlog – to the customer the hedging related work was “complete”.

Our position however has remained the same – there was no customer requirement that was driving the work to update the old design of the hedging screen with the new framework design introduced to the rest of the code-base. We believe it makes sense to wait until new functionality comes along in the hedging area so that these design enhancements can be effected as part of the normal engineering activities. Since the code-base is an organic, ever changing process, there will often be areas within it that are not as perfect as the team might like; we wouldn’t recommend creating stories to deal with perfecting them.

4. Working with the technical stories

By the third iteration planning game after the technical stories first arose, both still remained incomplete. For the previous two iterations the stories consumed much of the development resource, and during this four week period (our iterations are two weeks in length), little business value had been delivered. Furthermore, with both of these stories still very incomplete, the situation seemed to have no positive outcome.

In the planning session that these stories arose, we explained that they were horizontal in nature, and both clearly did not look as if they could fit into a single iteration if they remained defined as technical stories. At the same time, another problem we identified was that the stories could not be easily tested or accepted, again highlighting a problem in their structure.

In all normal cases the structure of our user stories have been using that suggested by Mike Cohn [1], along with the INVEST acronym (I for Isolated, N for Negotiable, V for Vertical, E for Estimable, S for Small, T for Testable), while our story estimates are rooted in work by [2] and [3]. Based on these simple guidelines, it was clear the technical stories needed addressing.

4.1. Intoxicating positivity

There is a lot to be said for the positive excitement of developer confidence, and this was pervasive during the first iteration in which the technical stories were explored; the confidence was so high the team was adamant it could complete them within the iteration. In fact the up-beat spirit was such that even the customer proxy was also in favour of getting the technical stories out of the way, since from a certain perspective they seemed justified. Unfortunately by the time the second planning meeting arrived, both technical stories had lost their allure as they had become an Achilles heel for the team, with all the hallmarks of a mini-death march.

As is often the case with a large code-base, cross-cutting changes often uncover issues with the code that can cause major problems to the team’s ability to make small incremental updates. In fact, it’s probably true that this approach inhibits any ability to apply emergent design, which can negatively increase complexity and further slow the team down. Both technical stories seemed to uncover more problems as each day passed by, and the configuration settings story was forcing re-writes of whole system areas due to how the sweeping changes were being made.

4.2. Losing traction

The technical story work was begun with a team velocity of 24 story points, but by the end of the first iteration, this figure had been reduced to just 11.

Nonetheless, in the second planning meeting the team spent a long time impressing how close they were to finishing the technical stories, stating that so much progress had been made in the previous iteration that it would be a waste not to finish them now. At the same time, it was clear everybody was now beginning to see the problem with these technical stories. Seizing the opportunity, we asked if we could split the stories to make them vertical, but due to the amounts of inertia behind these stories, it again seemed like they must run their course unfettered.

Lost in the details of software engineering and with palatable frustration and disappointment, the customer proxy acquiesced to the teams request to complete the flagging stories. So with the customer proxy’s backing, the stories were scheduled into the second iteration as the highest priority work and the team pulled together to make a heartfelt commitment to deliver.

Of course, this story, which is not a technical story, doesn’t have a happy ending. During the third planning meeting both technical stories remained incomplete and even worse was the fact our velocity had almost flat-lined, with the team completing only one story point of work. The configuration settings story seemed to be never ending, and without many ideas on how to finish the very incomplete hedging story, they were both set to take up most of the resource for the next iteration.

Unfortunately the customer proxy could not be present for much of the planning meeting, but for the time that he was, he exhibited frustration and exasperation at the incomplete stories. It was not ideal
that the customer proxy was unable to commit to full attendance, and with the need to rush off he quickly accepted that the technical stories needed to be concluded, and selected them for the third iteration; then he left. The result was that the complete third iteration was assigned to technical stories, and there was no real velocity to speak of from the previous two iterations, since our ability to deliver business value had almost completely disappeared, sucked into the black hole of technical stories.

With a little business value delivered over the past four weeks, and no end in sight, the second day into iteration three dealt a swift conclusion to what had long turned into an untenable situation, when the customer proxy requested an emergency meeting after the morning stand-up. As soon as the morning stand-up completed, the customer proxy leapt into an emotional monologue, expressing his deep frustration at the situation with the technical stories. Not only did he not understand the stories, but he explained he was experiencing extreme levels of pressure from the business because high priority work from the iteration before Christmas had still not been scheduled, and he genuinely didn’t know what to do next. Flabbergasted and deflated, he explained how he put his trust in the team, and we had said that these stories would be completed weeks ago, yet both remained incomplete with lots of work still left to do. Furthermore, other stakeholders are looking at the plan, seeing that these technical stories are assigned top priority iteration to iteration, and it is only the developers who understand them, but nobody else wants them. It was impossible not to feel the customer proxy’s pain; it wasn’t a good situation.

To re-align the team, the only viable solution at the time was to completely cancel the configuration settings story by backing out of some of the functionality, and then parking it. The split hedging screen story was split into a number of vertical user stories, but they weren’t ideal and didn’t conform to the best practice guidelines. However, to the team this seemed like the most appropriate re-dress. But parking work often means leaving it in the code-base, and this comes with its own problems. If the effect of this is that the team can return to delivering business value, then such a strategy seems constructive. The problem with this approach is that the unfinished work is waste, and having it in the software, waiting to be completed is something an agile team must learn to avoid.

Nevertheless after all of this, only a few iterations were lost to the overall experience, and this is not such a great deal in the scheme of things, since the project has run for two years or so, and has a long future ahead. The final situation certainly was not perfect since the team had a lot of work to do in rebuilding the trust of the customer and the business, but the experience has been rich with insights into a key aspect of agility that should be useful as similar challenges arise.

5. Conclusion

User stories are fundamental to a customer-driven process, and they are successful because they keep the requirements capturing simple, both to apprehend and work with. Technical stories on the other hand change the focus from customer-driven to technical-driven, and in some cases this trade-off based on the context might be warranted. However we believe that for both novice and experienced teams, keeping the process as close to the users frame of reference, which is often non-technical, not only maintains an equal playing field between all of the team participants, but it also helps anchor the team to the principles of agility – collaborating and working with our customers.

A valid question asked by reviewers of this paper has been where there contributing factors such as technical debt that played an important role in the failure of the technical stories?

With respect to the team and its dedication to the project before we arrived, the code did indeed suffer from rot in many places. However we feel the code quality was no less perfect than many other software projects that are worked on every day around the world, not constructed using agile practices of test driven development, simple design, refactoring, etc. But at the same time, a number of smells did exist in the code-base, and most certainly inhibited progress iteration to iteration.

We feel there are two key points to keep in mind that help avoid the need for technical stories. Firstly, recognize that while user stories are simple in their structure, it takes skill in working with them. To work with user stories requires some level of analysis skills, as well as being able to take a step back from the technical world and getting more into the world of the customer. We feel that because this is a skill to be developed, when pain arises the option is can be to just create technical stories. But this is the greatest opportunity to learn, and the extra effort can reap great rewards for all of the team. The configuration settings story could have easily been decomposed to cover the immediate requirements, which had already been understood to be the need to configure two of the core risk reporting screens. Technical debt or otherwise, the story simply didn’t need to be technical. Furthermore, the pinch point using approaches such as XP comes when a team is forced to re-evaluate its mental models for how to construct software – the paradigm shift from
big up-front design, to the practices of simple, evolutionary design need to happen. The not wanting to break the story up was primarily motivated by not apprehending how the horizontal work could actually be built incrementally.

The second key point we like to remind ourselves of is the principle of this is what we do. Keeping code clean, shaping it, evolving it, testing it, re-building it, within reason should all be part of our daily practice in carving code out of the digital oak. Many theories abound as to why the hedging story was pushed so hard to be left as a technical story, but ultimately we believe it comes down to not feeling comfortable with leaving the code-base in a transitory state, with the two user interface designs existing side-by-side; the hedging story was nothing more than a wish to refactor the code. Sometimes its not appropriate to spend time and money on something that is going to add little value, until it has been truly perceived that way – conjecture of the future possibility results in decisions based on no tangible foundation. Agility is all about putting off decisions till we have the best information to work with the challenges, and this in our view reaches through the whole process, right down to working with the code. With code-magnets and other techniques, as engineers we can evolve and clean our code-base over time without having to stop or impact our ability to deliver business value.

If the code-base is truly unusable, then another option we have come across is to spend a number of iterations working as expected, followed by a one to two week period in which the team is left to clean up the code-base. This is extremely useful, because the customer doesn’t need to be engaged in this part of the process, and so when the iterations resume proper, the customer-driven process continues as before.

In the end, each team, project, environment and culture will be different, and what will be appropriate for one situation might not be appropriate for another. In the end flexibility and pragmatism will always prevail, and as long as the velocity can be maintained and the customer sees true business value being delivered iteration to iteration, then that’s ultimately what this is all about.

6. References

