Agile Development Processes (VT2014) Post-mortem Report

Romain Colombo

Agile methodologies (Scrum, XP...) and their practices are rather designed for professional teams who work on a full-time basis on one (or few) project(s). They somehow assume that the team progresses simultaneously on the tasks and members are able to interact almost every time, every day, possibly with warm communication channels. However, the settings of this project were quite different. We are student with little experience in programming. We had different schedules and other commitments made difficult for us to work all together on a regular basis. Given these constraints, applying a particular methodology “out-of-the-box” would be certainly ineffective, and furthermore, this might be true for any project. We had to try several practices, evaluate and adapt them with the intent to make these practices compatible with out process in an Agile spirit.

After this project, it became clear that it is not worth implementing a practice without understanding what principles it tries to benefit. I have structured this report around three main topic (“understanding the goal of the project”, “team collaboration and project organization” and “technical perspectives”) which represent significantly the Agile principles we tried to implement through different practices.

Understanding the goal of the project

The requirements of this project were first (deliberately) not well defined. The customer expected an application used by a navigator of a pair programming team that “helps them to work more efficiently”. “Make it easy to use and performant”. Being Agile is particularly helpful for this kind of situation where the customer expresses a need only with unclear requirements.

Brainstorming (first team meeting)

Starting a project is tough. In a regular project, a project manager would plan tasks and distribute them to the team members. In order to be Agile during the first meeting, we tried to involve everybody, reflect collectively, get a common understanding and share our visions. This was done for project purpose but also for team building. We used a whiteboard to draw mock ups of application screens. This helped us to discuss smoothly about the problem and to get a broad view of the assignment (what is relevant/coherent or not). This leads to write the first user stories.

During the entire meeting, the recurrent question was “does the customer really want/need this?”. Indeed, we came up with our vision of the assignment which might differ from the customer one. In order to mitigate this issue, we decided to only focus on “safe” stories for the first sprint (login, repositories listing...) and make a prototype in order to show our understanding of the problem.

UI/Interaction prototype

The aim of making a low-fi prototype in our Agile-based project was two-fold:

- Get a common vision of the targeted program and help to define more precisely the splitting of the tasks ("I will implement this screen, but linking it to the previous screen is part of your story")
- Share with the customer the complete view of the understanding of the team as early as possible, stimulate reactions (since requirement are vague) and get an overall feedback before the implementation

Making a prototype is time-consuming and does not give direct value for the customer (i.e.: working software). However, it was a relevant artefact for the first customer meeting because it allowed us to have a short and effective discussion around a tangible example instead of having a "small talk" on vague concepts. This really fits the "customer collaboration over contract negotiation" value of the Agile manifesto. Later on, we used this prototype as a "requirement document" but we did not focus on maintaining it up to date. Indeed, writing down the eventual differences from the prototype in the details of a story was sufficient. However, if the project had required deeper modifications, we would probably have needed to update the prototype in order to avoid confusion.

Acceptance criteria

Sprint lengths were short, technical implementation was challenging for most of us and we had to deliver working features by the end of each week. Consequently, we avoided to write down too many details about the stories and only relied on our shared understanding (discussion during meetings, prototype…).

Relatively quickly, this raised some issues: misunderstanding of the story, duplicate work or missing parts. Indeed, even though everybody agreed and understood the goals/boundaries of each story right after the planning meeting, this did not seem to be as clear few days after. We mitigated this trouble by writing detailed acceptance criteria along the user stories and this was successful. On the one hand, this required a bit longer work during our meetings, but on the other hand, this avoided wasting valuable time during the rush of the end of the sprint.

Thus, relying only on direct communication might impact the project productivity. An Agile team has to continuously think about how much documentation is needed to produce according to the situation and not just try to always avoid this part of the project process.

Team collaboration and project organization

Planning meeting, planning poker, point estimate

In a Scrum-based project, the Product Owner is usually responsible to plan a consistent list of stories for the next sprint, according to the customer's priorities and the estimations of the team. Since we did not have a dedicated planning meeting with the customer, we discussed roughly about the next sprint priorities with him during each acceptance test sessions. Then, our team met alone in order to write collectively the user stories, prioritize and estimate them. These planning sessions took about two hours for each sprint.

The estimation of stories was both interesting and frustrating. We used planning poker with point estimation. Playing this game is particularly fun, especially when the planning session tends to be long, since it breaks the monotonous flow of the meeting. The debates were framed by the game rules: the members which provided the least and the most points gave their justifications before a new turn. However, we used a set of three points (one, two, three) simply because it was the option provided by our virtual task board (Pivotal Tracker). This particular set of point did not make sense for us because, depending on the opinion and the experience, a “one”, a “two” and a “three” could be raised for a same story. Using the Fibonacci sequence instead (1,2,3,5,8,13…) would be interesting to try out next time since it lets to be precise for the little stories and roughly estimate the big ones (which are inherently uncertain). It is probably for this reason that we did not always feel the interest of this practice and we tended to use it less by the end of the project.

In addition, another advantage of the estimation debates is that they helped us to identify which team members were most comfortable in implementing a particular story.

Task board

In Agile methods, the task board is not anymore a tool dedicated to a project manager in order to keep track of a plan. It can be changed continuously, provides details of the upcoming stories and a brief overview of the following ones, and more importantly, it is owned by the whole team.

We used a virtual task board (Pivotal Tracker). Comparing to a physical “homemade” task board, we have to either adapt ourselves to its usage or learn how to customize it. It took time to “tame” it and to start using it in an efficient way. However, since we did not have an assigned room, using a virtual tool was appropriate.

When it comes to involve all the team member in the project organization, using such shared artefact is crucial. It allowed us to be more self-organized by pulling the tasks from the board instead of having a centralized assignment.

However, when we wanted to implement the detailed acceptance criteria, Pivotal Tracker was not suited. We had to use the Github issues feature for writing them. On the one hand, using several tools can be uncomfortable, but on the other hand, this let the task board clear and concise by only displaying the essential information.

“Daily meeting”

The daily meeting is usually set up every day at the same time in order to bring the team member up to date, raise questions/issues and coordinate the project. To keep this meeting short, it is sometimes done “stand-up”.
Despite we were aware of the value of this practice, our constraints made impossible to implement it. Often, members who worked the day before could not be present the next day, and the ones who were present were not able to work the day before. Given the lack of coordination during the sprints, we started quite naturally to post short messages (“daily minutes”) on our Facebook group. A Facebook post is actually really suited for this purpose: messages tend to be concise, comments can be used for reacting to an issue, the “like” feature plays the role of acknowledgment by the other members and screenshots can be easily attached to get a quick overview of the work done. Even though our practice has not the benefit of a warm communication channel, it was quite effective. People present at a meeting started by reading the previous “minutes” of those who were absent.

So, the “daily minute” was our adaptation of the traditional daily meeting for an “unsynchronized semi-located” team.

Sprint retrospective

One of the key principles of the Agile manifesto says “at regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly”. From the second sprint, we implemented sprint retrospective meetings after the acceptance test sessions in order to discuss about how we were working together and how we could improve our process.

It was suited to set up these meetings between two iterations because the team members could have a free mind. These meetings took typically less than one hour and the topics were summarized as bullet points in a Wiki page. This let us quickly refer to it at the start of the following sprint retrospectives. They were structured around three questions: “What went well?”, “What did not go well?” and “How to improve?”. The range of topics went from details like “the standardization of our branch names” to more general issues about our process.

The Scrum master of the current sprint was the facilitator of the meeting. He/she wrote down the key elements of the discussion and tried to stimulate the debate among the members. However, by doing so, the team members were more likely to talk directly to (and to look at) the facilitator instead of having an overall discussion with all the team. We did not succeed to mitigate this issue. It would be interesting for a future project to try to stand up the facilitator and ask someone else to write down the ideas.

I think these retrospective meetings were very useful for us as it is a core practice in the Agile methodologies. It seemed easier and easier to conduct them while the team members got to know better each other. However, this may have worked well with us because the real purpose of this project was to try out Agile practices. Getting the commitment of the team might be more difficult with other settings. That is why it is ineffective to implement a practice without understanding what it tries to solve.

Technical perspectives

Testing

Some Agile methodologies, as XP, encourage writing automated test and running them as early and often as possible. This aims to deliver higher quality software and allows implementing effectively other practices like “integrate often” or “collective code ownership”.

In spite of our willingness to try many practices, we were not able to implement a structured and systematic test methodology. By the end of the project, we wrote some functional tests with the framework Robotium, but this was more individual initiatives (and curiosity) than a collective decision.

I think we did not feel the need to use a strong testing methodology because our work was mainly about presenting dynamic data (UI) retrieved from a high-level API (third-party code). Moreover, we were not accustomed to the Android environment, so it was not very clear what we should test in order to secure our code. The different features of our program were written in different “Android activities” and the interface between them was simplistic. Introducing a bug in one of the activities would have been directly visible in this activity and would not have impacted the other ones.

However, even though we did not get serious trouble due to the lack of automated testing, this could have become a technical debt if we had continued the project, and consequently complexify the design.

Simple design

The simple design practice consists in only implementing code that is needed to make a feature working in the simplest way. This avoids preparing the current design to receive future features that will be -eventually- implemented later on.

I am not really sure that we consciously use this practice at the beginning. Instead, we were influenced by the short length of the iterations (and we had to deliver valuable features) and by the fact that we were beginners in Android development (at start, we were not able to think ahead anyway).

We figured out later that we were practicing simple design during the planning meetings when we got a better overview of the project goals and when we already had a base architecture: “We can implement story A like this, it’s a bit more complex, but it will be easier to integrate story B during next sprints”, “Well, I’m not sure that we will implement story B, let’s focus on A”.

Quick iterations were an incentive to choose the simplest way. In a project with longer sprints, the team would need to be more vigilant. However, this project was a really good example for us, because we realized that many of our “good to have” stories have not been implemented finally. If we had taken time to “prepare” the design for them, we would have probably produced less “top” stories.

Conclusion

Agile is a set of values and the different methodologies are frameworks that implement these values with various practices. There are many methodologies because each product and each project setting need to be managed in an appropriate way.

This project was the opportunity for us to try out different practices, evaluate and improve them. Combined together, we made our own methodology which tried to fit our constraints and ourselves as individuals. We came up with a relatively satisfying result comparing to the time we were involved in the project (about the third of a full-time team).

I consider this project as a first step towards Agile practicing. The assignment was short enough to focus on our process but our little technical experience let us evaluate the benefits of these practices. In a future -more complex- project, things will be probably different. The need of coordination might be higher and consequently “team collaboration” practices will have to be enhanced (communication, synchronization, colocation). The technical part might need to be more rigorous (testing, coding standards, refactoring). From a managerial point of view, the process should be also measured (velocity, burndown charts).

What stays in my mind from this experience is that Agile puts people first. It undertakes the fallible human nature by constantly question the process in order to improve it. Agile tends to benefit to all the parties involved (stakeholders, managers, technical staff) and is adaptive through collective intelligence. From my point of view, this makes the digital world even more exciting.