Agile Management in Software Development Projects

Mohamed Khalil

November 30, 2015

Abstract

This report presents the agile management style, a management scheme that recently is being commonly adopted for software development projects. I will present a comparison between such a model and quite-famous alternative, the waterfall model, followed by a detailed explanation of the different stages of the agile project from the start to the conclusion. Finally, based on an experience from Ferienakademie 2015 - Course 5, I will present some challenges that might be confronted by the management and the developers. For elaborative reasons, real-life examples from the Ferienakademie's 2015 - Course 5 project "Grand Theft Boat - Sarntal" or from my student job at the chair of Computation in Engineering(CiE) in Technical University in Munich(TUM) will be presented.

Contents

1 Introduction

Agile management is a management style quite adopted by software development teams. It was proposed by 17 software engineers in February 2001 in what they referred to as the $AgileManifesto$[1], in which they stated:

> We have come to value:
>
> - Individuals and interaction over processes and tools,
> - Working software over comprehensive documentation,
> - Customer collaboration over contact negotiation,
> - Responding to change over following a plan,
>
> that is, while there is value on the items on the right, we value the items in the left more.

Generally speaking, agile management is viewed as a break-free from the traditional management style which sticks to a stiff strategic and daily plan aiming to deliver a fully-functioning project in one shot at the end of its duration. In agile, the management, in line with the team, are usually more focused on getting a fundamental prototype of the project's deliverable, or a running beta version of the software, as a first step and then adding a hierarchy of deliverables on top of that as time approaches the project's deadline. Meanwhile, the increments in the project are highly susceptible to modifications or renovation by the team or the customer of the software. Such a flexible model presents a significant deviation from the classical waterfall model presented in the following section.

1.1 Contrast Between Agile and Waterfall Management

The waterfall management style has been quite well-known among the software development community. It basic structure is shown in figure 1. As a first step, the requirements of the software are determined via an interaction between the customer or the sponsor from one side and the management or the developers team directly from another side. Usually, within the team, such requirements are elaborated and presented in deeper technical detail, but the key aspect here is that, at this point of time, the requirements are more of less set and forever, with a minimal probability of alteration in the future.
Later, based on the set criteria, the design phase of the software is initiated with more involvement of the developers and less customer interaction, aiming at setting the overall plan of the software whether technically (such

the tools needed, the program structure, etc.) or managerially (such as division of teams, time plan, etc.). The development phase is an extensive implementation phase where all the planning is executed and the developers progress towards the final deliverable consistently.
At a later stage, the software is got together and tested. It is a quite notorious point of disappointment, bug discoveries and quick fixes by the team, since the code as a whole piece is being integrated together, with high hopes of perfect fit of all the interfaces[1]. Following that, final tune-ups are conducted during the last stage, the implementation phase, to deliver the software in its promised shape to the customer.

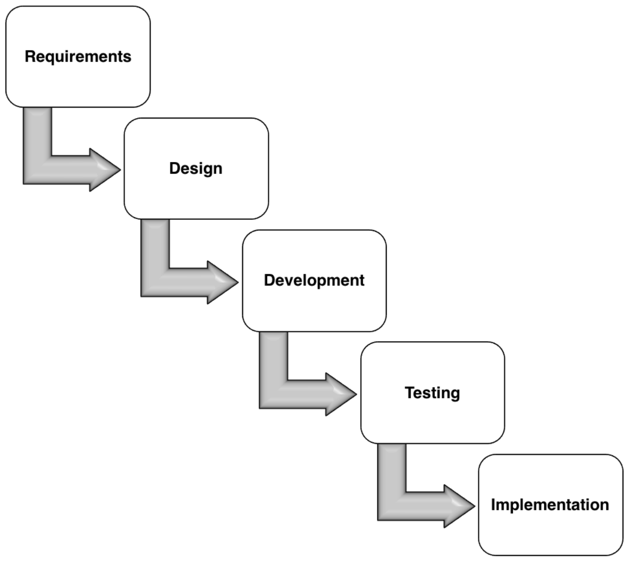

Figure 1: Waterfall model

Now compare this model to the agile model, presented in figure 2, which is supposed to be for the exact same project with the exact same duration. In the agile model, as shown, the project is divided into a plurality of chunks, referred to as *features*. For the sake of analogy and contrast with waterfall model only, it can be perceived that a tiny waterfall holds for every feature, such that the project as a whole is an integration of a series of deliverables, the features, and not a single deliverable at the end. Nevertheless, it is important to note that the features' management doesn't follow a waterfall model in this management scheme. The details of this point are elaborated in subsection 2.3. Another aspect in agile management, which stems from its definition, is the flexible response to the fluctuation of the requirements. Unlike, the well-defined and agreed-upon contract of requirements set at

[1]of course, step-by-step testing is conducted along the way to ensure this point is bug free, still there is a higher risk of code-blocks' mismatch or system tests failure

the first stage of the waterfall, in agile, the customer usually remains in action along with the team throughout the project, continuously providing feedback and adjusting, if needed, the upcoming deliverables' requirements. Also, the team is more ready to react to those changes by relocating its human or financial resources.

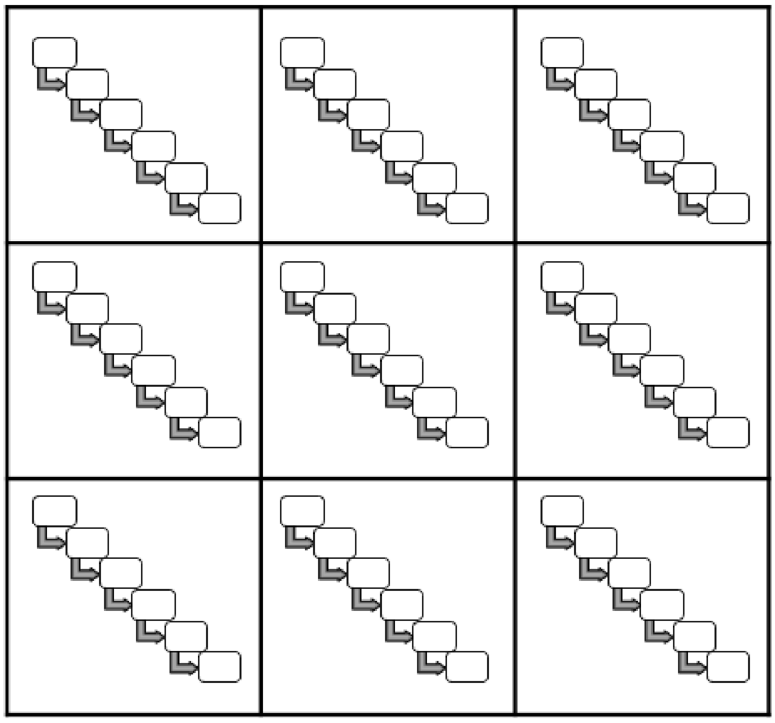

Figure 2: Agile model

Common Aspects between Both Models

It is worth-noting that, although both schemes have some differences, nevertheless, they still share some points in common, which they would also probably share with any other project management scheme. First, a clear vision and a determinate set of requirements need to be set for the project. The milestones for the project might change along the way, however, the team need to be aware of what they are working for, even after it changes. In addition, communication is a vital aspect in agile projects and in any team work generally. Finally, a collaboration between an attentive manager to the on-going changes and the timely progress of the project from one side and a dedicated team from another side is a key towards the success, despite the management style.

2 Stages Of Agile Management

Agile management is characterized by specific aspects concerning its features management, as pointed out in the previous section, as well as its different stages; the envision stage, the sprint and the closure[2]. In this section, an in-depth picture of the agile management aspects will be portrayed.

2.1 The Agile Feature

As stated before, the term *feature* in agile management refers to the deliverable of each chunk of the project. To make it clear to the management and the developers how such a deliverable should look like, the agile feature is characterized by:

- A minimial amount of work of requirements to be delivered, or as quoted from The Agile Manifesto *"minimize the work **not** to be done."* To elaborate further, I would present an example from my student job. In a FE package, we wanted to implement an eigen problem solver (a solver aiming to find the eige modes of a given structure). In linear algebra, there are two main types of eigen problems, a generalized and a standardized. Without going into details of linear algebra, the simplified could be regarded as *simpler*, and more likely to be chosen as a start step and to delay the implementation of the generalized sovler to a later stage.

- An updated input from an involved customer, if available, or an agreed decision by the team on the next incremenet to add to their project. Taking back the same example, it is now the point where the team should decide whether to go for a generalized eigen solver or to show satisfaction with what they have now and switch their attention to something else, a transient problem solver, for instance.

2.2 Stage 1: Envision

This stage is corresponding to the requirements and design stages in the waterfall model, but with a slight difference. In this stage, the management should lay the foundation for the following three broad lines:

- **Project Vision**: What do we do? For whom? Why? The answers to those questions are the guidelines for the project's pathway. In our Ferienakademie 2015 (FA) course, for instance, we were doing a *boat race game incorporating a fluid-structure interaction simulation.* For whom: well, for ourselves. As mentioned in the previous point, there was no "customer" or "sponsor" for our project. Why we were doing

[2]in contrast to five stages of the waterfall

it: first of all, it was the coursework and also because the developers were sort of interested in the course's theme and the project's topic, since it has got the highest votes.

- **Project Charter**: This is the hierarchy of the stakeholders of the project. It should comprise the management, the different teams and their connections to one another and the customer or sponsor, if any.
- **Project Tools**: These are the necessary set of tools needed by the team to perform his task. In the FA course, examples of those were Github for code exchange and Bullet Physics libararies for rigid body motion.
- **Project Norms**: These involve the set of oranizational rules and guidelines to be followed through the project. An interesting point that might be covered by this is the schedule of scrum meetings[3]

2.3 Stage 2: Sprint

It is called sprint because we are delivering a small deliverable, the feature, as soon as possible. This stage is the core of the agile management. To be more precise, after the envision stage is over, the project more or less is about the stack of successive, consecutive sprints and their delivered features. For every sprint, a feature or more are determined as a deliverable. This could be a new one just proposed by the customer, an already planned one from the general plan for the project or a late one from a previous sprint. Usually, a sprint's duration is from 4-12 weeks in a normal working week; it is very feature-oriented. The sprint comes to an end either after a feature is ready to be delivered or the allotted time runs out. In the latter case, the feature has to be re-evaluated and rescheduled in the coming sprint or not. It could even be completely discarded if the customer/ developers decide so.

Useful Sprint Tools

Some techniques might facilitate the realization of the agile model during the different sprints. Among those are:

- **Scrum Meetings**: These are periodical meetings held for the developers to share among the team their task updates. Usually, their talks consist of their progress on an assigned task in the feature, the challenges or difficulties they have faced and their prospective propsed work. These meetings are usually kept short and precise, where the manager should play the role of the observer of the team's progress and guide its next step rather than the moderator of the meeting's agenda or content.

[3]explained in section 2.3

- **Backlog**: This is the to-do list of the project's features. With the continuously changing demands and relatively rapid pace of the project, the manager as well as the team need to be aware of the current status of the tasks in the feature. They also need to be noted when tasks are added, modified or removed from their duties list. Keeping such a tracking system, whether electronically or classically, would perfectly fit for this job.

- **Unit-Testing**: A crucial tool, yet by-passed by some programmers. Unit-testing ensures that each block of the code is actually doing what it is supposed to do. It assures the developer that his code is logically correct. This is performed by creating a simple test case and running the code on it and comparing it with apriori-known solution obtained either by experience, from hand calculations or from a similar tested package (ex: benchmarking a matrix inversion function's output against Matlab's output). This ensure that any usage of an already existing is error proof and that any malfunctioning caused by future modifications in the code is captured, since one or more of the tests will fail.

- **Continuous Integration**: It is very important that throughout the sprint that the programmers take a pause from scripting new code and merging their current progress together. This in turn points out if there has been any overseen error in the interfaces and ensures that the whole code works as anticipated.

- **Pair Programming**: It has been noticed that the presence of two programmers working on the same code reduces the possibility of bug occurence. Pair programming works as follows; one developer is actually coding while the second one is tracking his code to check for syntax errors, bugs, bad programming style, etc. Of course, this might be a luxury if human resources are limited, but it is a recommended action if feasible.

- **Co-location**: For some features in the project, the presence of one person in two single teams can be a great advantage. One example was in FA course. The member responsible for the interface between the fluid simulation code and the rigid bodies code was holding discussions all the day with people from both teams more than any other member in the either teams held such discussions.

- **Re-location**: In other situations, one task might no longer be as human intensive at a given time. In this case, some persons might be shuffled around supporting other teams (pair-programming with other coders for example, or helping debugging and unit-testing). This was clearly obvious in FA when coders from different subteams jonied the rigid bodies and the fluid simulation groups in debugging segmentation faults, while towards the end of the project duration, a new team was built to develop the game logic (originally, members of this team were doing something else before it even existed).

2.4 Stage 3: Closure

After all sprints have passed, the management ought to ensure that all the required deliverables are met and that the project is ready for closure. Accompanying that, a reflection from the entire team over the project's progress is highly encouraged to motivate continuous improvement. And of course, the team should celebrate !!

3 Challenges in Agile

Agile is not as smooth as it might seem. There are always some challenges that can show up, of which the management and the team need to take into consideration to avoid falling into their traps.

3.1 Waterfalling

Now that the difference between the waterfall and the agile models are explained, it can be inferred what could this challenge be. A failure to identify a solid milestone, specially at the beginning, might result in a reselient feature in a notorious manner; meaning that the feature will keep extending and extending beyond its allotted time without any signs of being completed soon. This occurs because the teams will be very focused on the clear project's goal rather than the current feature(s). The management needs to make sure that the output of every feature is clearly conveyed and understood by assigned memebers. In our case in FA, the first milestone of the project lost the agile track and fall into a waterfall because it was unclear for some teams that the goal of the first milestone was to connect the different teams across the project's pipeline, even with dummy data at the interfaces. In contrast, some teams delayed the implementation of the interfaces and rather focused on obtaining elaborate but rather time-consuming real data instead.

3.2 Feature Delays

This is not a standalone challenge, but rather an aftermath of waterfalling. When features are extended, due to waterfalls for example, or are even misplanned on the timeline, they might end up on the stack of the following sprint and so on, leading to a delay in the planned features for the proceeding milestones. In the FA course, due to the Waterfalling of the first milestone, we had to rush through implementing a lot of dummy data through the code in order to wrap up this stage. It was planned that for the second milestone a more elaborate user-input would be implemented, but this in turn was postponed until we, first, resolved all the dummy data with physically sensible ones.

3.3 Parallel Implementation

Another problem that usually results from the delay or non-frequency of continuous intergration is parallel implementation. No, this is not the code parallelization which saves time - it, on the other hand, consumes more time!! What is meant here is the case when teams program perfect but separate code. The problem arises when different code piece need to be fit together in the pipeline. Conflicts such as segmentation faults, incompatible interfaces or a pipeline with a malfunctioning logical sequence could be usually the results of such an issue.

References

[1] M. B. et. al., *The agile software manifesto*. `http://agilemanifesto.org/iso/de/`, 2011. [Online; accessed 20-November-2015].