Theoretical vs. Practical Software Development Techniques

Jasper Nicole Morrow

Follow this and additional works at: https://louis.uah.edu/honors-capstones

Recommended Citation
https://louis.uah.edu/honors-capstones/195

This Thesis is brought to you for free and open access by the Honors College at LOUIS. It has been accepted for inclusion in Honors Capstone Projects and Theses by an authorized administrator of LOUIS.
Theoretical vs. Practical Software Development Techniques

by

Jasper Nicole Morrow

An Honors Capstone
submitted in partial fulfillment of the requirements
for the Honors Diploma

to

The Honors College

of

The University of Alabama in Huntsville

05/01/2017

Honors Capstone Director: Doctor Richard Coleman

[Signature] 05/16/17
Student (signature) Date

[Signature] 5/16/17
Director (signature) Date

[Signature] 5/16/17
Department Chair (signature) Date

[Signature] 5/17/17
Honors College Dean (signature) Date
Honors thesis Copyright Permission

This form must be signed by the student and submitted as a bound part of the thesis.

In presenting this thesis in partial fulfillment of the requirements for Honors Diploma or Certificate from The University of Alabama in Huntsville, I agree that the Library of this University shall make it available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department, Director of the Program, or the Dean of the Honors College. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this thesis.

Signature

Student Name (printed)

Student Signature

Date
# Table of Contents

Abstract ........................................................................................................................................ 2  
Introduction ................................................................................................................................ 3  
An Overview of Software Development Techniques ........... 3  
  Agile ......................................................................................................................................... 4  
  Scrum .................................................................................................................................... 5  
  Kanban.................................................................................................................................. 7  
Applied Software Development Techniques ...................... 8  
  Agile ......................................................................................................................................... 9  
  Scrum ................................................................................................................................... 11  
  Kanban.................................................................................................................................. 14  
Conclusion ................................................................................................................................... 16  
Bibliography .............................................................................................................................. 18
Abstract

There are many different approaches to the process of software development. Each has its own advantages and disadvantages, and every team of developers chooses to use whichever best suits the needs of both the team and the project they are working on. The styles of software development that I will focus on will be general Agile, Scrum, and Kanban. I will be reviewing how each of these styles of development are intended to be implemented in theory, how these are each implemented in companies across the world, and how these are implemented in a smaller classroom setting with a team of students, and some smaller, local companies.

General Agile is a very flexible method of software development that is more suited to a project that may have changing requirements. This style creates multiple iterations of a product over a period of time, with a backlog created around the needs of the customer. Each of these iterations of the product will be a working product. Rather than working on smaller parts of the product at a time, this focuses on having fully functioning versions of the product released at incremental periods of time. This style puts the customer’s satisfaction first, and is very suited to finicky customers whose needs may be subject to change.1

Scrum is currently the most popular subset of agile, which is normally used for much larger, more complex products. This approach does not change, in the respect that the responsibilities and things like meetings never change. Scrum uses fixed-length iterations to complete a project. These are called sprints, and any team member will be assigned user stories to complete during this sprint, which is a small task that compiles the bigger epic that the whole team is to complete during the sprint. Daily during these sprints, there is a standup where the team members will discuss: what they completed yesterday, what blocks they had, and what they plan to do that day. At the end of the sprint, there will be a sprint review to discuss what happened during the sprint and what the team wishes to change or keep the same during the next sprint. This style is very structured and allows for lots of feedback from both ends throughout the process.1

Kanban is another subclass of Agile, which is one of the easiest methods to switch to or add overtop of another method. This method specifies exactly how to produce software. This system allows for team members to pull more work from the “card" if they finish with their work early, and new work cannot be added to the card until space is freed up by another job being finished. This method puts efficiency first, and matches the abilities of the team members very well.1

Throughout this paper, I will be comparing the ways these three techniques are meant to be implemented to the ways that they are actually implemented in both a student and professional environment.


Introduction

Many software development companies are struggling with different styles of development techniques, trying to find the perfect match for their company, their customers, and their product. With so many different techniques out there today, it can be hard to consider which one might be best for a given situation. With that in mind, I have decided to review the theoretical and practical applications of three prevalent techniques to help clear up some of the confusion about these techniques, while also showing how they’re actually being successfully or unsuccessfully implemented in the real world. The three techniques I will review are Agile, Scrum, and Kanban.

An Overview of Software Development Techniques

A software development technique is simply a tool for assisting software development teams in the planning, coding, and testing of their larger software projects. They provide a framework with which a team can tackle a very large project with relative ease, and ensures that the work is done by the deadline. There are many different kinds of techniques a company or team could choose to use but the three main techniques I will cover are Agile, Scrum, and Kanban.
Agile

Agile is the mother software development technique that spawned some of the other current techniques. It was originally created by a group of software developers that agreed that these twelve principles are necessary to the developmental process: the top priority is satisfaction of the customer, requirements shall always be open to change, working software must be delivered frequently, developers and businesspeople must work closely together each day throughout development, the project must be built around motivated individuals with an environment that is friendly to the creative process, conveying information to or within a development team is most efficient via face-to-face interactions, the primary measure of progress is working software, the pace of software development should be constantly sustained, technical excellence and good design enhance agility with software, simplicity is essential to the product, self-organizing teams make the best software, and the teams must regularly reflect on how to improve at set intervals. These twelve principles comprise the backbone of the current Agile development technique, which is a very flexible, yet organized form of software development that allows for last-minute changes, yet ensures the product is done by the deadline. There are six steps to the Agile development cycle: step one is the

---

requirements analysis, where the business requirements are identified, and the team does proper research on the product and its use; step two is planning, where the project is broken down into smaller parts; step three is design, in which the team comes up with the look of the product, such as the GUI; step four is the actual implementation, in which the team actually creates the features that they planned to have in the product; step five is testing, where the team tests the code and finds and fixes any bugs or problems with it; the last step is deployment, where the product is actually delivered to the customer. This cycle focuses on customer satisfaction as well as efficiency, which is why Agile is widely used in software companies and teams today.

**Scrum**

Scrum is a software development technique that spawned from Agile, with many similarities, but with more structure planned into the development cycle. Unlike Agile, Scrum is heavily centered on a set of three roles, which keep the development process running smoothly. The first role is the Product Owner, who is responsible for building the vision of the product and making sure that the team understands what the finished product needs to be and also breaks the product

---

down into steps in the backlog to give the team goals to meet. The next role is the Scrum Master, who controls the scrum process rather than the team, working with the team and the Product Owner to overcome challenges and arrange meetings, suggesting any changes to the sprint pacing they see necessary. Then, of course, there is the Scrum Team, which will normally have approximately five to seven members who work together to complete the stories in each sprint until, and even after, the product is completed. The Scrum structure is centered around six main steps: making the product backlog, sprint planning, refining the backlog, daily meetings, sprint review, and sprint retrospective. The backlog is a list of features to be included in the final product that is made by both the product owner and the team together, which is used to help keep the team on track throughout the development of the product. During sprint planning, the team decides what items in the backlog they can achieve in that sprint, and move them to the backlog for the sprint to be completed. After the sprint is completed, the team does backlog grooming to reassess the backlog and refine it to better suit the product as the development continues, such as removing items that are no longer relevant or necessary to the product. Daily sprint meetings last about fifteen minutes, and in them the team discusses what was completed the day before, what needs to be completed upon that day, and any roadblocks that are stopping development. There is a sprint review meeting at the end of each sprint, where the team presents all of their completed work for the sprint. There is also a sprint retrospective at the end of each sprint to open up a discussion about how well the scrum process is working for them, including a talk about the good, bad and possible improvements for the
overall scrum system in place. This method provides the same flexibility that Agile provides, but also gives a firm structure to ensure that the development process runs smoothly.⁴

**Kanban**

Kanban is another, easy-to-apply method derived from Agile, which places emphasis on efficiency and is based on a manufacturing line in which a new task is not placed on the task board until another task has been completed. This method is very good for matching the pacing and abilities of a team, and focuses on smaller changes rather than large chunks of project. Typically, a board, be it physical or digital, is used to keep track of the items to be completed for the project to be complete. These boards typically have three columns: to be done, in progress, and completed, though they could have other varied columns depending on what works best for the team. The items to be done are placed on Kanban cards in columns on the board depending on what stage of progress the item is in, and a team typically cannot have a new item placed on their board until another item is completed, to prevent overloading and to keep older items from being overlooked in favor of other items that were more recently placed on the board. This technique is based upon five general principles: visualizing workflow, which allows for a better

understanding of the larger picture of the final product goal and makes it easier to identify possible issues at the beginning of the process so that they can be discussed ahead of time; managing that flow, by constantly improving upon it to help the team produce results as quickly as possible; limiting the work in progress on the columns on the board, which removes the need to prioritize tasks, which improves the speed and flexibility of the production system; making policies explicit, such as explaining to the team what it means to be truly finished with an item, whether it be finishing the raw code, or testing and revising; and continuously improving the system through small changes consistently throughout the process. Kanban is a very easy development technique to use that focuses on giving only as many tasks as a team can handle, but also the maximum number of tasks a team can handle, maximizing the efficiency with a very simplistic system.\(^5\)

**Applied Software Development Techniques**

The actual implementations of these development techniques are different from team to team, and some implementations are more efficient than others. I have analyzed the implementations of these styles in a few companies, as well as in a classroom setting to determine how well each style actually works in a professional setting.

setting and offer possible improvements on each style of development. Though the sample size is small, I feel like these generalizations apply to many of the companies that develop with these styles of techniques, and that any company that uses them would benefit from considering this analysis.

**Agile**

Agile is one of the more commonly used techniques, and I shall discuss the implementation of Agile from T&W Operations, Inc. from the viewpoint of an ex-employee as well as the implementation from my own experiences working on a software development team in a classroom setting.

The software developer I interviewed worked for T&W Operations, Inc., which is a software company that works closely with customers from all levels of government, from 2006-2015; he was on multiple teams during his time with the company, and has therefore given an approximation of the combination of all of their experiences with the company. Contrary to the principles of Agile, the team was not specifically in direct contact with the customer throughout the process, and would only directly contact certain customers at varying times throughout the development process. The number of team members on any of the teams he was placed on varied from two to six members, and the members would meet only once a week for about ten minutes to discuss the product and their progress on the work they were assigned. The team was not assigned any roles, and the whole team shared every role in the development process, such as developers, designers, etc. If a team member was to be ill or missing for other such reasons for a period of time,
rather than reorganizing the team and redistributing the workload, the team would meet with the customer to discuss the setback and would push back the date that the product would be finished to give the other developers more time to make up for the loss of a member. However, the team was very good with changes to the requirements to the product and the process was very flexible, as the requirements could sometimes change literally once every hour. Overall, I would say that this experience was not very true to what Agile is supposed to be like, and some of the efficiency of development was lost due to their loose use of the development technique. If the teams had met more often with the customers, like is typical in an Agile setting, the requirements could have been discussed more clearly, and would not have had to change as often as they did, saving the team a lot of work that was lost in scrambling to change the code to meet ever-changing requirements. The team should have also met together more frequently to discuss the work that each member was doing. Instead of having to push back the product date each time a developer was out sick or had urgent family issues, they could have efficiently redistributed the work amongst the other team members if they had been meeting more often because the whole team would have been aware of the whole process. However, the teams were still very able to complete the project and all of its requirements, so the Agile process isn’t entirely failing them.⁶

---

In my personal experience with Agile, my team played around with Agile for a period of time to attempt to organize our senior project. In line with the Agile process, my team often discussed progress with the customer, and were able to ask for clarity on any of the requirements and give status updates approximately once a week for about ten minutes. My team was comprised of four members, and everyone also shared responsibilities during the project. However, we did have some members that were designated by the team to have a higher level of responsibility due to their expertise in one area or another, and these roles would change when necessary as dictated by the team together. If one of our team members was to fall ill or have a family emergency, we would simply redistribute the workload to the other members without pushing back any of the deadlines as a result. However, our team was not faced with any changes in requirements as brought on by the customer, but rather we had many changes brought upon by ourselves. The team did not respond very well to these changes, and any substantial change in our understanding of the requirements would set the overall progress of the team back, causing us to work longer hours towards the end of the project. Our implementation of the Agile process was fairly true to the theoretical Agile process, though our team was not as flexible to change, which caused us many headaches towards the end of development. If I could suggest any changes, it would be to have the team fully discuss and understand the requirements before continuing development to prevent the need to change the workload so late in the game.

**Scrum**
Scrum has become a more popular technique in recent years, and I will discuss my experiences with it both in a professional setting, from my time at Intergraph PP&M, as well as in a more relaxed setting in my senior software development project.

Intergraph PP&M has, as of recent, become a company that is very centered around the Scrum method of software development. I worked at Intergraph for a brief internship in 2016, where I was a member of a small Scrum team of about eight members, including the Product Owner. However there were teams as small as three members to teams as large as ten members, though from discussions with my coworkers, the five to seven member teams do the most efficient work without wasting any team member’s abilities or time. All the team members shared the same responsibilities, excluding the Scrum Master and Product Owner. The Scrum Master, as expected would run the meetings, and was responsible for keeping digital track of exactly what work needed to be done, and taking down any notes he felt were necessary to be recorded on the overall Scrum process. The Product Owner would help the team develop a backlog, and broke down the requirements for the team, and they would only be present in sprint reviews to discuss with the team what was accomplished. We, of course, had the typical daily stand-up, sprint reviews and sprint retrospectives. The daily stand-up featured the entire Scrum Team and Scrum Master, and lasted about fifteen minutes, where the team would discuss what was completed the day before, what needed to be completed on that day, and the team also discussed any roadblocks they had in their coding. When discussing any blocks, they would keep the discussion brief and continue talking through their issues after
the meeting was over. The building was designed to have many open spaces and places to sit as groups to encourage the developers to communicate more, so after the meeting the developers working on the problem together could talk to each other by their desks or move to a different space to discuss alone where it was more private. The sprint review included our entire team, along with the Scrum Master, Product Owner, and our supervisor, an employee who was overseeing many teams to ensure that the product was being made to the satisfaction of the customer. The sprint retrospective was run by our Scrum Master, where the team would discuss what they felt did and did not work for them as far as the Scrum process was concerned. They were asked to rate the overall sprint, tell the Scrum Master what did and did not work, and give suggestions for things to be implemented in the next sprint. If any of the developers had to go away to be with family or became ill, there was no additional stress added to the team because the process was so organized that they could just assign other members the user stories that would have been assigned to that member originally. Having larger teams also helps with this, as it allows the one member’s work to be more distributed so that the impact on the team is not large. Many developers say that there is no company that actually implements Scrum properly, but Intergraph seems to implement it almost perfectly, with great results. There really aren’t any improvements I could suggest on the way that they implement Scrum, and I feel like it’s working perfectly for the company.

My software development project group also vaguely implemented Scrum at times to try to get our project organized. We only had a team of four, but we still assigned Scrum Master and Product Owner roles to members within our team.
Specifically, one team member ended up being responsible for each of those two roles, though these roles were only vaguely adhered to. The Scrum Master would ask for progress from each member whenever we would host meetings, and the Product Owner would break down the project into smaller user stories and put them in the backlog to be selected by team members themselves. We did not have retrospectives or reviews, but we did have “daily” stand-ups where we would meet after every class period, or “work day” to discuss where everyone was in their work. During our meetings, we also would discuss what work was completed, what work needed to be done, and what problems we were experiencing, as expected of a daily stand-up. The biggest drawback we experienced from this method was the lack of responsibility amongst the team. The Scrum Master wasn’t holding members accountable, and the team wasn’t actually meeting together to discuss the project and the work that was being done. Some members would end up doing work that others had already completed and the updates to code would clash with each other and crash the program. Our implementation of Scrum wasn’t very ideal but if we had implemented the roles properly and communicated better, I firmly believe this would have been a successful method for us.

**Kanban**

Kanban is not as much of a commonly used subset of Agile, but some companies, such as DevSpace Huntsville still use it. My senior project team also used Kanban the most of all of the techniques discussed, and I will review the implementation in both of these cases.
DevSpace Huntsville may not be a software development company, but it still has to use software behind the scenes to make its websites work, and for that they use Kanban. DevSpace uses an online Kanban board, which can be accessed by each of their board members. Any member is able to add, remove or move any of the Kanban cards from the board, and the members select work for themselves as they please after a discussion with the other board members. They use the three simple columns: to do, in progress, and completed. As new requirements arise, cards are added to the to do board; as work is selected, it is moved to in progress; when work is completed it is moved to done. The rule of not adding work until work is completed is not followed, and in this case it leaves the board muddled and sometimes items are neglected. There are also not solidly set deadlines because it is a non-profit company rather than a development company being paid by customers, so the work is simply done at an “as-you-wish” pace where members simply pick up work where they can or where they want to. This is a very imperfect form of Kanban, but it is working well enough to keep them above water. A suggested improvement would be to manage the flow of the cards more closely and to have some sort of system to assign work to ensure that someone is doing it.\(^7\)

My senior software development team used Kanban as its preferred method of distributing work, since it was the easiest to implement and required very little setup. One member was in charge of adding items to the Trello board as

---

\(^7\) Christopher Gardner, "Practical Software Development," e-mail interview by author, May 14, 2017.
requirements were made clear, and for a brief part of the project, he would assign
cards to members, but as the project continued, members simply selected cards that
they wanted to do and assigned themselves to it. Our team also did not follow the
rule of only adding cards as they were finished, but rather added cards as new
requirements were discovered. The team also initially set certain cards to be done
only in a certain timeline. Some early developmental cards were to be done in the
first few weeks, and then GUI was added to be done at a later time, though this fell
apart near the end of the project when the team reverted to simply posting every
job in the to do column and grabbing them as we could, trying to tackle many cards
at once. While this method did work for us, it would have been better implemented
if we had more organization about the way we were posting and grabbing cards
from the board. This was our most successful method simply because it was much
easier for us to break down the work and get it done when there were small, visual
cards in place to help keep us organized.

Conclusion

All three of these techniques have their benefits and drawbacks, and each
one works best in different kinds of situations. Agile works best in situations where
communication with the customer is frequent and the team is good enough at self-
organizing that they don’t need the added help of assigning certain roles to
members to keep the team on track. Scrum works best when there is a lot of time to
devote to starting up the development process, when there is an environment in
which the team can freely and frequently communicate, and when there is a need to continually work on development of the product even after it is completed. Kanban works best for companies that want to get the work done as quickly and efficiently as possible without much discussion throughout the team.

Overall, implementations of these techniques normally fall flat of the ideal, so it is hard to determine which technique I would suggest to use for general software development, but in experiencing first-hand the well-oiled machine that was Intergraph PP&M’s scrum teams, I would have to say that I think Scrum is the best method for development. The biggest problem that is encountered by teams appears to be lack of organization, and Scrum offers a good structure to keep the teams on track. There are multiple members at different levels of control in place to help manage the team and ensure that everything is working smoothly. The Product Owner makes sure the team understands the work that needs to be done, and the Scrum Master ensures that the team is on task, while the team itself ensures that the work is being properly distributed. The addition of the retrospectives lends itself to constant improvement on the system itself as well. If anything doesn’t work out, it can be simply tossed in the next sprint, and the team is constantly open to implementing new ideas to make the process run even smoother. If it can be properly implemented, I would highly suggest attempting to use Scrum as the go-to software development technique, as it seemingly has the best results.
Bibliography

