**Project Two**

Brian Kittrell

Southern New Hampshire University

CS 250: Software Development Life Cycle

Cory Thoma

October 14, 2022

Scrum-agile teams consist of a Scrum Master, a Product Owner, one or more Developers, and one or more Testers. The Scrum Master works closely with the Developers and serves as a non-authoritarian manager or project leader for the team, keeping notes, guiding the development of the project, and watching the progress and how it aligns with the requirements. The Product Owner serves as the other half of the project management side of agile and works as a liaison between the client and the team, refines product requirements, reviews deliverables and test results, and keeps track of progress. These two positions are important because a team without some kind of guidance can go off course quickly, which may result in ignoring or missing product requirements, and the end goal of the project is to produce a product that meets the client’s requirements.

The Development Team represents one or more dedicated programmers tasked with implementing the design and requirements so that the software product can be tested, and the Testers carry out that testing. The importance of both are obvious–the software must be made, and feedback must be received–but many may not recognize that having separate Developers and Testers eliminates the potential for bias in the process. Developers can be a prideful bunch at times, so having a separate group of Testers can be invaluable for finding the real problems in the software before it comes to the client to go live.

The team undergoes a cyclic repetition through the agile methodology. First, planning, analysis, and design are undertaken to determine what needs to be done in the current cycle, who will do it, and what the deliverable looks like. Implementation occurs next, where this part or module of the product is coded. Some very basic testing can–and arguably should–occur in the implementation phase to make sure the software functions, but the true test comes from the testing phase that follows. The product is tested thoroughly by the Testers, and the code from the current cycle is integrated into the product. Any changes, error corrections, and debugging takes place in the maintenance step, and many teams have a brief reflection between the end of the current cycle and the beginning of the next to incorporate lessons learned, improve processes, and settle grievances that occurred during the cycle.

Given the circumstances and requirements of the project from the course, the waterfall method or the agile method could have been successfully employed to develop the product in question. However, I also recognize that we did not produce a complete product during the course, as many of our user stories were not implemented in the design before the end of the term. Had we included all of those additional complex modules and functionality, the scope of the project would have quickly outgrown the waterfall method. In smaller projects with small teams or a solitary developer, waterfall makes sense and can be more efficient. In fact, many developers I’ve worked with in the past and I sometimes prefer the waterfall strategy when dealing with inexperienced clients. The extensive planning phase of waterfall in small projects with clients new to software development can make the process smoother down the line when the planning phase is thorough.

In projects with the intended scope of the SNHU Travel product, an agile approach is almost always better. Agile is less sensitive to changing requirements, and the requirements did change throughout this project, and dealing with changes in requirements throughout the process is where the agile method really shines. The process also addresses the need for change when issues are found by the team. For example, the background color and text color in the project weren’t conducive to a positive user experience, and in my email to the Product Owner as a Developer, I addressed this concern, offered my opinion, requested the opinion of the Testers, and made numerous suggestions to improve the end user experience when the service was to go live. Had we been operating under the waterfall methodology, these requirements would be defined in the design documentation, which is a problem for Developers because they know returning to design during the implementation phase could result in huge delays. It is often better to complete the project and treat future changes as a new project than to return to design at that late of a stage, but in agile, design is just another piece of the weekly or biweekly cycles, and requirements can and will change throughout the process.

Another strength of agile is user stories and how they can be used to bring about the exact results. For example, let us assume that two statements could be made to describe the same requirement:

Design a user interface with three buttons to produce the desired activity: New, Save, Load.

As a user, I would like the ability to open new files, save my work to the local drive or the cloud, and load my files from any drive so that I can keep my work portable when I’m traveling.

The first statement is vague and nondescript, but the second statement as a user story gives specific requirements and needs to be fulfilled by the software. By phrasing these statements in a certain way, we get intent, cause and effect, and targeted use cases, which help the software meet the requirements of the end users.

Finally, the tools that supported the strengths of the agile process were tools that facilitated fast and easy communication, such as sending and receiving emails. Although we did not use many tools during the course, instant messaging tools, especially those geared toward groups such as Slack, would be immensely valuable to a team pushing toward the finish line. Various bug- and issue-tracking software, such as Jira, would be helpful to keep things organized along the way, and Continuous Integration/Continuous Development tools, such as GitHub and many others, help keep the codebase available and stable as programmers work and changes are integrated.

The principles that I followed the most throughout the process were a commitment to technical excellence, welcoming changing requirements, and dedication to simplicity. Technical excellence is obviously important in any technical project, but it can lead to unwarranted pride and elitism if team members begin to assume that they know everything there is to know. On the contrary, technical excellence is about learning and performing the processes necessary to produce good software. I’ve learned to welcome changing requirements in the agile process because they should be seen as opportunities for improvement rather than burdens to bear. Lastly, the dedication to simplicity is a strong guiding principle because it focuses on the shortest route to the goal and reduces the urge for engineers to overengineer. I’ve fought that latter desire before, and it can be a hard thing to overcome. But for the benefit of the project and its timely conclusion, overengineering should be avoided, and the shortest route to the goal often results in the most efficient and less error prone path to the destination.