2018

University of North Carolina Wilmington

Master of Science in

Computer Science and Information Systems

Proceedings

https://csbapp.uncw.edu/mscsis

A COMPARATIVE ANALYSIS OF AGILE PROJECT MANAGEMENT METHODOLOGIES

James Gray

A Thesis Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Department of Computer Science
Department of Information Systems and Operations Management

University of North Carolina Wilmington

2018

Approved by

Advisory Committee

Jeffrey Cummings	Devon Simmonds

Elizabeth Baker
Chair

Accepted By

Dean, Graduate School

TABLE OF CONTENTS

ABSTRACT

A Comparative Analysis of Agile Project Management Methodologies. Gray, James, 2018. Thesis Paper, University of North Carolina Wilmington.

Agile methodologies are becoming increasingly popular within software development teams. As companies adopt these methodologies to improve their processes and deliver higher quality products, it is imperative that students have a firm understanding of how Agile project management methodologies function, as well as how to effectively work in those settings to remain competitive in the job market. This paper reviews current research into Agile project management methodologies, in particular Scrum, Extreme Programming and the Hybrid model. There is a wealth of research on when and which methodology to implement, the best scenarios for each methodology, and teaching existing teams to use these methodologies. However, research is scarce in regard to when and where these methodologies could be incorporated into academics. Two experiments were performed comparing two agile methodologies during each experiment. During the first experiment development teams worked under the guidelines of Extreme Programming and Scrum. The second experiment conducted had the development teams working under the guidelines of the Hybrid model and Extreme Programming. The performance of these methodologies was then compared using velocity, defects detected and time to completion. By comparing the performance of these three Agile methodologies, it was determined where these methodologies would best fit into an academic curriculum. Extreme Programming was found to best fit the learning objectives in early courses in Computer Science or Information Systems. Scrum was determined to fit best into the objectives of intermediate level curriculum, while the Hybrid model was found to be well suited for senior level capstone curriculum.

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1: INTRODUCTION

There is ample research on when Agile methodologies should and should not be implemented. This research also extends into how to implement Agile methodologies and which Agile practices work best within existing teams. However, information and resources are scarce on how to incorporate the practice of Agile training into academics or which Agile methodology is best suited for academic curriculum. Throughout a computer science or management information systems curriculum, Agile methodologies are referenced and taught frequently; however, students are seldom required to actually work in Agile environments in the classroom as part of learning best development practices. This lack of practical application and exposure might be due to limited opportunities for students to meet outside of class hours in conjunction with possible departmental limitations. Yet, with the rapid adoption and proliferation of Agile methodologies throughout the software development industry, the absence of this practical experience and subsequent cursory knowledge puts students at a competitive disadvantage.

This research will be a comparison of the performance of selected Agile methodologies and a field study in how students work in practice using these methodologies within their curriculum. The research question asks which of the three most popular Agile methodologies can be most effectively applied within the academic classroom. The goal is to give students the development experience they need to be more successful and competitive in a professional workplace setting through applied learning based on employer desired skill sets.

Agile Project Management can be thought of as set of methodologies whose purpose is to aid in the productivity and efficiency of a project, while promoting better decisions and thinking through problems more effectively as a team (Stellman, 2014). Agile itself is often referenced not as a stand-alone method, but an overarching philosophy or mindset to guide project teams.

According to the "Agile Manifesto," there are twelve key tenants to Agile Project Management and Development. These twelve tenants are centered around delivering working software frequently. They emphasize not just accepting but embracing the changing requirements and environments, as well as collaboration and communication between developers, customers, and management. The tenets also include reflection on how the team can become more effective and then adjusting accordingly. (Stellman, 2014)

Figure 1: The Agile Methodology - The Official 360Logica Blog 2017

Currently, there are a plethora of Agile methodologies being implemented by project teams across the world. Although these methodologies all seem similar on paper, in practice each methodology has its individual strengths, weaknesses, and optimal settings. Agile methodologies have proven to be well suited for software development projects where there is a high technical risk. Projects like these typically have uncertain requirements, and the project team is expected to be flexible and anticipate new requirements during the project's life cycle. Communication between stakeholders and project team members is paramount in Agile methodologies due to the constantly changing project requirements.

The goal of this research is to contribute to both computer science and information systems development education through developing a strong understanding of how both Extreme Programming and Scrum work in practice in an academic setting. It will also give valuable insight into how students learn, work, and perform under a strong teamwork paradigm. This will allow for the exploration of the implications in regard to accelerating students' comprehension and preparation for the work force.

CHAPTER 2: REVIEW OF LITERATURE REVIEW AND ANALYSIS

Among software development approaches, Agile methodologies have become the most widely used model in the industry (Musa, 2017). Agile is typically defined as an iterative approach that delivers software incrementally at the end of time segments, often referred to as sprints. Many would say that Agile is much more than just a way to break down a project into manageable chunks; it's a "set of methods and methodologies that help your team to think more effectively, work more efficiently, and make better decisions." (Stellman, 2014, pg. 2). Stellman (2014) continues to explain that Agile is a mindset of a team, one in which planning, design, and process improvement are opened up to the entire project team, giving all members a say in how development practices are defined. This literature review examines published evidence describing the impacts and differences among the Agile methodologies of Scrum, Extreme Programming and the Hybrid model on a project teams' performance. In addition, this study investigates whether or not a specific widely used Agile methodology is more well suited for implementation within an academic course setting.

2.1 Scrum and Extreme Programming (XP)

Agile methodologies have greatly increased success rates in regard to development, quality and speed to market (Rigby, 2016). With the dominance of Agile methodologies in the workplace, it's necessary for students to have a firm grasp on Agile concepts to better prepare them for industry standards. The two Agile methodologies that are considered to be the most popular and widely used in software development teams are Scrum and Extreme Programming. (Rubin, 2013; Musa, 2017). Both emphasize customer satisfaction and quick development over all else. These processes utilize iterative development to release models which ensure that the customer is provided with the product they desire, when they need it.

2.2 Scrum

Scrum is an empirical framework that is used to manage and address complex product development while productively and creatively delivering products with the highest value possible (Schwaber, 2016). The origins of Scrum are frequently debated. Some attribute it to Hirotaka Takeuchi and Ikujiro Nonaka from their 1986 Harvard Business Review article "New New Product Development Game (Nonaka, 1986). Others argue it was Jeff Sutherland and Ken Schwaber who introduced the methodology in 1995 during the annual Object-Oriented Programming Systems, Languages & applications conference (OOPSLA, 1995). In 2001, Ken Schwaber and Mike Beedle presented the methodology in their 2001 book "Agile Software Development with Scrum" (Anwer, 2017).

The three mainstays of Scrum are rooted in transparency, inspection, and adaptation. Transparency refers to the fact that all meaningful aspects of the process should be visible to all team members involved in development, with all team members developing a common understanding of what is being observed. Inspection relates to keeping track of and having an understanding of Scrum artifacts to be certain there is no deviation from the sprint's goal. Adaptation ensures that, if a deviation is detected that would result in an undesirable product, the deviation can be resolved or adjusted to curve further deviation (Schwaber, 2016; Anwer, 2017). A graphic describing the workflow process of Scrum is presented in Figure 2.

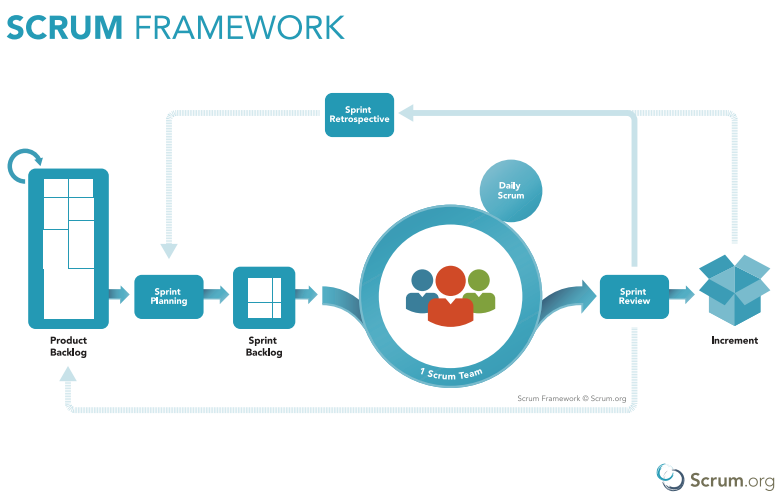

Figure 2: The Scrum Framework (Scrum.org, NA)

2.3 A focus on Scrum Teams

To understand how Scrum teams operate, it is necessary to have an understanding of a typical Scrum team's composition. A Scrum team is comprised of three parts, which include the Development Team, the Scrum Master, and the Product Owner. Development Teams are typically small, usually between three and no more than nine members. This optimal size range is based on the given rule that if the development teams size is less than three, team interaction and progress in sprints will be hampered (Blom, 2010). Smaller teams are more likely to encounter constraints in skill which can potentially render them unable to deliver at the end of a sprint. Conversely, teams larger than nine can lead to confusion and simply require too much coordination (Schwaber, 2016). Blom (2010) states the ideal situation is a team between 5-7 people.

Development teams are self-organizing, meaning that it is ultimately their decision on how to transform the product backlog into a functional, incrementally releasable system (Steghöpher, 2017). In addition, there are no formal titles for the members of the development team aside from that of developer. Sub-teams are not recognized within the development team

(Schwaber, 2016). Although Schwaber (2016) specifically states that it is not to be the case, it's notable that in practice this is typically not followed, testers and business analysts tend to work as sub-teams.

The Scrum Master acts more as a facilitator than a Project Manager on a Scrum team. The Scrum Master's role is to ensure that the team is following Scrum practices, theory, and rules (Blom, 2010). Scrum Masters coach the development team in self-organization, aid them in creating organizational value, removes blockers or impediments to the team's progress, and shields the project team from outside interferences (Schwaber, 2016). For the Product Owner, the Scrum Master assists in finding ways to effectively manage the Product Backlog, establishes the value of clear and concise Product Backlog items to the Scrum Team, and ensures that the Product Owner properly arranges the Product Backlog to maximize value (Schwaber, 2016).

The Product Owner is held accountable for maximizing the value of the Scrum Team's product, as well as the work of the Development Team (Stegöpher, 2017). Regardless of organizational differences, the Product Owner is solely responsible for managing the Product Backlog. Managing the product backlog is comprised of: (1) ordering the product backlog in a way to best achieve goals and missions; (2) making sure the product backlog is transparent, visible and clear to all members of the Scrum team; and (3) making sure the Development team understands the product backlog and its components to ensure a mutual understanding of the tasks at hand (Schwaber, 2016).

Scrum relies heavily on a time period referred to as the sprint. Sprints are sequential time allocations with a fixed duration, depending on the size and complexity of the task at hand. Schwaber (2016) states that during a sprint, no changes are made in regard to scope that would jeopardize the sprint's goal, and scope can and will be adjusted as the product owner and

development team explore and learn more about the project. In a typical Scrum setting, a short daily meeting is held known as a "Daily Scrum" or "Stand Up" meeting, which usually last no longer than 15 minutes. Upon completion of a sprint, a "Sprint Retrospective" meeting is held to discuss what went right, what went wrong, and the lessons learned by the team from the activities involved in completing the sprint (Schwaber, 2016).

The values of Scrum are courage, commitment, respect, focus, and openness (Stellman 2014). These values help the self-organizing teams to work and adapt to situations effectively. Throughout a sprint, focus is a crucial element for the team. During a sprint a team member must be completely dedicated to the tasks they have taken on to ensure the success of the sprint. Additionally, team members are expected to retain a level of mutual respect of each other throughout the project's life cycle. The value of respect has more to do with trusting other team members to complete the tasks they've taken on and delivering the highest quality of work possible (Stellman, 2014). This leads directly into the value of commitment. Through being committed to the project, the team is able to achieve the sprint and project goals without the need for bureaucratic assignment of tasks. Within Scrum, project teams' openness is highly valued, ultimately meaning that any other member of the project team should know what another team member is working on at a given point in time (Stellman, 2014). Team members are expected to have the courage to take a stand for the project and work through tough problems as a team (Schwaber, 2016).

Scrum artifacts, such as the product backlog and sprint backlog, represent work and value. They are designed to promote transparency within a project, allow for inspection, and if needed adaptation, as well as promoting a mutual understanding of all team members (Schwaber, 2016). The product backlog, the main source of requirements for a product, is a list of items,

features, enhancements and fixes that are needed for the product to meet its requirements (Schwaber, 2016). Product backlogs are expected to grow with a project as requirements are discovered or change through the course of the product life cycle.

The sprint backlog is made up of product backlog items that are to be completed for a sprint. It can be viewed as a prediction done by the development team concerning what functionality will be delivered in the next increment and attempts to pinpoint all work deemed necessary to meet the sprint goal. Sprint backlogs are expected to be updated throughout a sprint as work is completed, requirements change, and other functionalities are found to be irrelevant. It typically includes plans for delivering the product's increment and achieving the sprint goal as well (Schwaber, 2016).

2.4 Extreme Programming(XP)

Extreme Programming, often referred to simply as XP, is similar to Scrum in many structural ways. However, unlike Scrum, many of the practices within XP are specific to programming and attempt to assist developers in creating higher quality code (Stellman, 2014). Beck (2003) explains that there are five core values of XP and twelve primary practices. The core values of XP are communication, simplicity, feedback, courage, and respect.

XP's core values are similar in nature to those of Scrum, but have some differences in terms of how a development project is undertaken. The first of the five core values, communication, centers around an effort to keep the right channels of communication open. By keeping communication between team members flowing, documentation is kept at a minimum, and it ensures that all team members are conscious of the work being done by other members. This is achieved through imposing practices such as unit testing, pair programming, and task estimation. Simplicity, the second of the five values, focuses on the promoting the effective

design of simple and direct solutions to problems. Feedback, the third value, assists in keeping the project on track and can span time periods ranging from minutes to months. Having the customer be on site throughout the duration of the project benefits the team greatly in the form of immediate feedback. The fourth value, courage, is centered around the idea that each team member should be focused on making the best possible decisions in regard to the project, even if that means having to abandon and start fresh on a solution that was designed with great effort by the team. Courage also puts forth the attitude of acknowledging your personal shortcomings and being open with the team. By embracing this attitude and not being afraid to fail, the development team will see an increase in productivity and produce a higher quality product (Beck, 2003). Finally, respect, the fifth value, emphasizes that all team members are valuable to the project. It's difficult to be told a solution you labored on is designed poorly. Working in a pair programming environment can be especially challenging and requires a great amount of respect for your fellow team members (Beck, 2003; Angioni, 2006; Anwer, 2017; Stellmen, 2014)

Extreme Programming works in what are referred to as cycles instead of sprints. Cycles range anywhere from minutes to days, weeks or months, although a typical cycle is usually a week to three weeks (Beck, 2003). At the start of a cycle, the customer will identify the stories that will be implemented during a given iteration. The development team works throughout the cycle implementing the requested features derived from user stories and using feedback cycles to fine-tune the product according to the customer. Cycles end with the customer running functional tests to determine the success or failure of product's iteration (Beck, 2003). A graphical depiction of the XP process is in Figure 3.

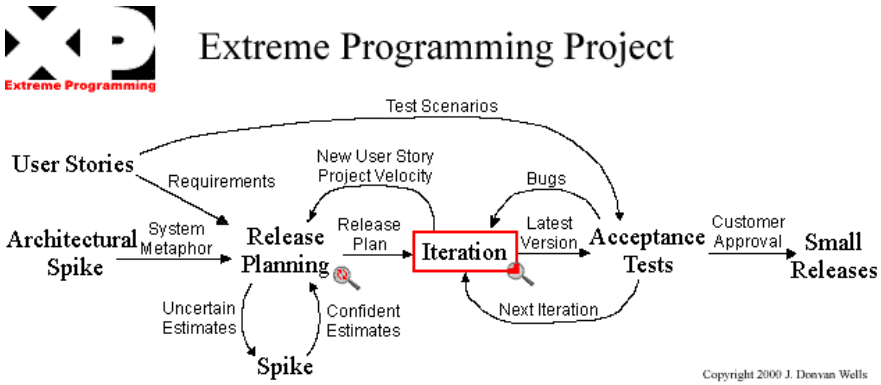

Figure 3: The Extreme Programming Project (Extremeprogramming.org, 1999)

The twelve practices that XP teams espouse are the planning game, small releases, metaphor, simple design, testing, refactoring, pair programming, collective ownership, continuous integration, 40-hour week, on-site customer, and coding standards (Stellman, 2014). During the "Planning Game", requirements are collected and stored in story cards which are used for future planning. Time required to complete tasks, the team's resource allocation and working hours are all defined during this phase of the practice (Wood, 2013). Small Releases refers to the consistent increase in value of incrementally releasing working parts of a larger system. The architectural design of how the system should work with the design of the system is referred to as the "metaphor". XP emphasizes simplicity; simple design exemplifies this by stating that the design and development of basic required functionality is the most important (Beck, 2003). Continuous testing assists in providing frequent feedback through extensive unit and acceptance testing. Collective ownership of the project means that not only all team members have access to all parts of the project, but that failure or success does not rest solely on the efforts of on individual from the team. Through continuous integration developers are encouraged to make frequent and systematic pushes to a shared knowledge base or repository to avoid integration conflicts. Finally, XP emphasizes the need for reasonable work weeks (the 40-Hour workweek)

and on-site customer presence. This makes sure that the development team doesn't burn out from work overload and that project progress is kept on track by the customer dedicated to working directly with the project team (Beck, 2003).

The practices of XP are very developer centric. The objectives of XP are based in the development of simple code and collaboration within the development team through pair programming. Pair programming, as the name suggests, is literally two developers working at one keyboard together. During this process, they are intended to learn from each other, offer immediate feedback on poor code, and ultimately produce a higher quality product (Wood, 2013)

2.5 Comparing Scrum and Extreme Programming

Table 1 illustrates key attributes of Scrum and Extreme Programming and compares and contrasts the two methodologies. When considering application in a classroom environment, it is important to take note of some of the key attributes of each methodology, including optimal project size, utilization of a project manager, and methodology focus.

In early courses, the methodology focus of curriculum is more geared towards engineering and learning best practices, areas where Extreme Programming excels. As students work their way into upper level curriculum, this focus seems to shift to managing time and student productivity on larger group projects, while maintaining best practices and implementing what they had learned beforehand. Project size and scope also typically grow as students work their way into upper level courses. With the increase in complexity of a project, the need for guidance from a professor/Project Managers becomes more apparent. These are areas in which Scrum excels and could be used to further engage the students.

Attributes	Scrum	Extreme Programming
Development Approach	Incremental and Iterative	Incremental and Iterative
Team Size (only developers)	Multiple teams of 3 to 9 developers	2-10 developers
Project size	Medium or small	Small
Project Manager	Can be utilized	Not utilized
Sprint/Cycle duration	One month or less	Days – 3 weeks
Requirement definition	Product owner	On-site customer and user stories
Methodology Focus	Geared towards management and productivity	Geared towards software engineering
Communication method	Scrum meeting - oral	Standup meeting - oral
Development Flexibility	Cannot change during a sprint	Can change during a cycle if needed
Quality Assurance	Not defined	Test first approach
Coding Standards	Loosely defined based on team	Well defined
Code Ownership	Not defined	Whole team
Documentation	Very little	Very little

Table 1: Comparison of Scrum and XP

2.6 Hybrid Model

Extreme Programming and Scrum represent two extremes of agile methodologies. Scrum focuses primarily on the management of a project team, while Extreme Programming is most concerned with engineering practices. It has been noted that Scrum fails to provide instruction regarding the proper engineering of a software product, whereas Extreme Programming lacks the benefit of management practices (Mushtaq, 2012). The shortcomings of these and other Agile

methodologies have given rise to hybrid methodologies, often blending two or more agile practices to form an amalgamation of the best parts of the chosen methodologies (ScrumAlliance, 2013). Within the software development industry, most companies do not employ a strict Scrum or Extreme Programming methodology. Instead, they use some form of hybrid between the two. The 2013 "State of Scrum" found that the majority of the participants in their surveys used at least one Extreme Programming practice in conjunction with Scrum (ScrumAlliance, 2013). These hybrid methodologies utilize Extreme Programming's engineering practices and Scrum's management practices.

There is no right or wrong way to make a hybrid of these two methodologies. Most organizations have created their own version of Extreme Programming to work alongside Scrum (ScrumAlliance, 2013). These practices are often pulled directly from Extreme Programming's practices and implemented throughout a sprint, with an example shown in Figure 4.

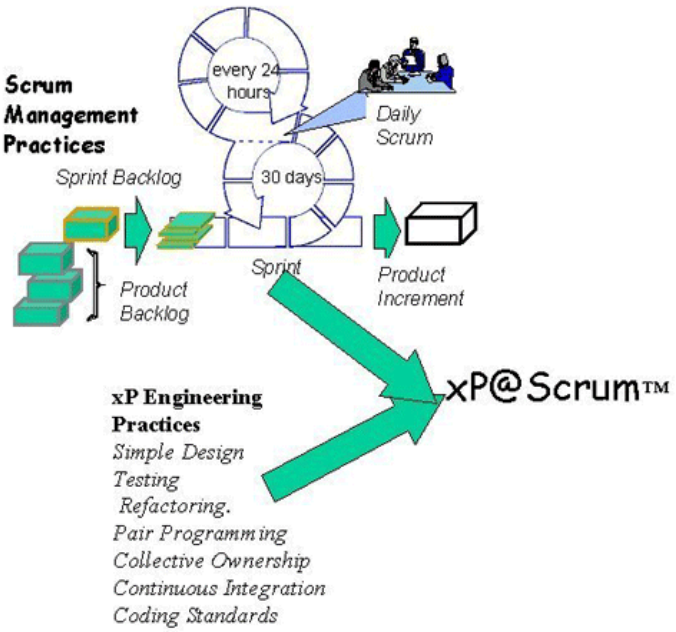

Figure 4: Scrum with Extreme Programming (Mar, NA)

Others have looked into implementing phases into sprints for Scrum teams, such as the

planning phase, design phase, coding phase and testing phase, as seen below in Figure 5 (Mushtaq, 2012). Daily Scrums are still held, and most, if not all, Scrum practices are carried out as usual. Both of these types of hybrid methodologies are intended to remove limitations of a pure Scrum or Extreme Programming methodology and enrich the overall project/process/development process (Mushtaq, 2012).

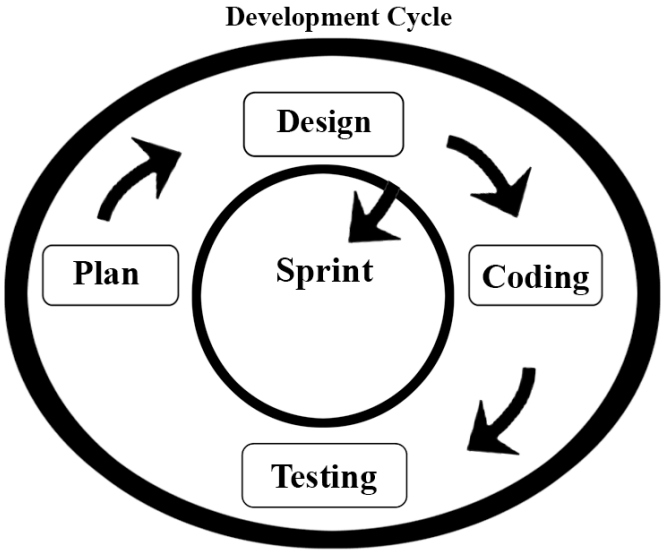

Figure 5: “The Proposed Hybrid Model” (Mushtaq, 2012)

CHAPTER 3: METHODOLOGY

The experiments performed compared Agile methodologies in order to determine which methodology performs best and is most easily integrated into academic coursework. The findings from the experiments performed give valuable insight into where these methodologies would best fit and provide the most value to students and faculty. This methodology section will give insights into how the study participants were handled, how data was collected, and the experimental method used for the experiments. The experiments discussed in this section took place over two semesters with one experiment being performed each semester.

3.1 Context of Product Manager and Project

The College of Health and Human Services at the University of North Carolina Wilmington expressed interest in a mobile application aimed at assisting their nursing students and clinicians. The application they were interested in developing pertained to Pharmacogenomics, an area of study which aims to predict how individual genetic variability impacts drug absorption, metabolism and activity. While one treatment approach may work well for one individual, the same approach may not be effective or may cause adverse drug effects in other patients. The application, which is called “PGx”, needed to be usable and accessible to both iOS and Android users. This presented the perfect opportunity to experiment with the Agile methodologies while the student groups worked in parallel on the same application.

For the second experiment, a directory style application was developed for the Cape Fear Garden Club. This application, like “PGx”, needed to be accessible to both iOS and Android users in order to provide an easier way to contact members and update personal information for the club. While the Cape Fear Garden Club application had a higher technical overhead, it presented a great learning opportunity for the student developers. The developers had to

incorporate MySQL database connections and user authentication into the application.

For both experiments, the User Interface and functional requirements were set before the semester and development began and did not vary during the course of the experiment. This was done to more accurately simulate how the Agile approach would play out in an academic environment, such as a software engineering course. The students participating in the experiments were learning mobile programming from scratch as part of a DIS, but had at least two semesters of programming experience before participating in the experiments.

3.2 Data Collection Process & Tools

For the data collection process, GitHub, Trello boards, Slack channels, and personal notes were maintained in respect to each development team. The development groups were tracked through their contributions to GitHub, the completion of tasks represented in their respective Trello boards, and a combination of Slack communications, as well as developmental notes, taken during sprint planning meetings, Scrum meetings, and communications facilitated via Slack.

During the retrospective meetings, after two cycles were completed, the project team took time to discuss problems, lessons learned, tasks that might have complications or bugs remaining, and the overall status of the project. Selected tasks were to be completed within that one to two week period with the expected result of each cycle being a working piece of software that can be demonstrated to stakeholders. Tasks that the team failed to implement were discussed in these meetings to determine what went wrong and to attempt to remove technical barriers before moving the tasks to the next sprint or cycle's backlog.

Progress was measured based primarily on the successful delivery of project product at the end of a sprint or cycle and the state of the final deliverable. Completion of tasks via Trello,

as well as pushes to GitHub, were also taken into consideration when measuring the progress of each group. These tools were then used to determine how closely the development teams met their sprint goals, project products, and ultimately their deliverable. With respect to completion of a task and completeness of said task, a numerical score was assigned to each groups effort ranging from one to ten, ten indicating completion much before schedule, seven being moderately ahead of schedule, five being on schedule, two indicating completed behind, and one much later than forecast. By tallying these scores at the end of the project, a clear view of how each methodology performed was attained.

3.3 Definitions of Scrum, XP and Hybrid Model used for Experiments

For the group working under the Scrum methodology, Scrum was defined to the team as follows: a project management methodology used in Agile software development that is based on an iterative and incremental delivery model, breaking work into fixed time periods known as "sprints", monitoring progress and re-evaluating plans in weekly or bi-weekly Scrum meetings with the end goal of delivering working software at the end of each sprint. Artifacts will be created using user stories based on customer requirements to produce a product backlog and sprint backlog.

For the group following the Extreme Programming methodology, the following definition has been chosen: a project management and software development methodology that is purposed around improving software quality and adaptability through common understanding of fundamental values and disciplined application of best practices. The core values of Extreme Programming are simplicity, communication, feedback, and courage. The overall process will consist of a planning phase in which requirements are elicited from the customer/backlog and

functionalities expected of the project are defined. Using these requirements and functionalities, user stories are then created each containing its value to business and its priority. The user stories are then further divided into micro-activities known as “tasks” that are ideally to be performed in a period of 2-3 working days, taking 2-3 weeks to be completed. Stories are only considered complete once they successfully pass acceptance tests which are functional tests used to guarantee the functionalities specified within the user stories. Similar to Scrum, Extreme Programming promotes frequent releases at the end of development cycles of 2-3 weeks at which customer feedback can be used to adapt the project to the customer’s specific needs and a working piece of software can be delivered.

The Hybrid model implemented features from Scrum’s management practices and Extreme Programming’s engineering practices. This provides an environment that supports an evolving product, as well as implements quality and functionality incrementally (Mar, NA). From Extreme Programming, simple design, test-first coding, continuous integration, and some pair programming were used, while from Scrum, sprints and planning meetings, daily Scrums, product backlog and owner, sprint backlog, and the Scrum Master were used.

3.4 Developer Teams

For the two experiments, the participants were broken into groups of two, , with one experiment being held each semester. The students were comprised of information technology, computer science and information systems majors. The students chosen had little or no knowledge of the programming languages they were to use prior to team formation. The students that participated in the study voluntarily agreed to participate for DIS (Directed Independent

Study) credit within their major. The student groups were instructed to complete training in their chosen language to become more proficient before the official start of the project.

The groups of undergraduate students were formed to each develop a mobile application using the Android and iOS platforms. Due to the varying levels of their proficiency in application development, the experiment was treated as a Quasi-Experiment. Quasi-Experiments lack the level of random assignment typically found in true experiments. Because this experiment is of a nonequivalent control-group design, the groups being tested were not randomly selected, and their individual skills may not be equivalent; in short this eliminates the concern of maturation, history and testing (Mitchell, 2013). The iOS development teams developed their application in the Swift programming language through Xcode, while the Android groups developed their application in Java using Android Studio. For the first PGx development project experiment, the teams were working under the guidelines of Extreme Programming for the Android system and Scrum for the iOS system. For the second Garden Club development project experiment, the iOS team was working under the guidelines of Extreme Programming and the Android group was using the Hybrid methodology. For this experiment, the same student groups were working in parallel on a directory application. This was done in order to avoid the effects of maturation in respect to familiarity with the two methodologies. Table 2 shows which methodology each development team was using for the two experiments.

	1st Experiment	2nd Experiment
iOS Development Team	Scrum	Extreme Programming
Android Development Team	Extreme Programming	Hybrid Methodology

Table 2: Methodology Assignment

3.5 Experimental Method

The PGx (iOS) group used a modified version of the Scrum methodology to better fit the developer time constraints as students. The project team worked in development iterations called sprints. At the beginning of a sprint, a sprint planning meeting was held to determine the course of action and which items from the sprint backlog were to be completed during that sprint. Instead of daily Scrum meetings the project team had weekly and bi-weekly Scrum meetings. During these weekly/bi-weekly Scrum meetings, each member provided information on what had been completed since the previous Scrum, what was to be worked on leading to the next Scrum, and if there were any blockers/roadblocks preventing progress.

Throughout the experiment with the Scrum-based project team, I acted as the Scrum Master and Product Owner for the project team. As the product owner, I was responsible for maintaining and prioritizing the product backlog containing features and requirements that needed to be built (Stellman, 2014). As the Scrum master, my duties to the project team included working with the team to overcome blockers that had been identified, holding the sprint review and the sprint retrospective meetings. During the sprint retrospective, we discussed lessons learned, successes, and failures in order to improve the project team's future sprints and the quality of the software developed.

For the PGx (Android) and the Garden Club (iOS) group following the Extreme Programming methodology, the weekly cycle practice was followed. However, cycles were set to a two-week period for our experiment, and working days were shortened from the typical 8-hour time frames to fit the students' availability. Similar to the Scrum methodology, each cycle began with a planning meeting. During these planning meetings, the project team evaluated progress made, picked stories for the next iteration, broke those stories down further into tasks

and established a developer to work on said task. Time estimates were also given by the developers working on specific tasks.

Depending on the source, Extreme Programming either appears to have no guidance or explanation of how roles typically found in other project teams would behave in an Extreme Programming environment (Anwer, 2017). Within the Extreme Programming groups, I acted as the "Tracker" and "Coach" whose role is to track the critical activity of measuring the team's progress, as well as supporting teamwork, team processes, and team cohesion. This was accomplished by tracking and communicating with the development team to identify what has been done, how much time it has taken, how much time is still required, and what if any blockers are present (Beck, 2003). While neither of these roles are necessary to an Extreme Programming team, they can be helpful to teams attempting to implement the Extreme Programming practices for the first time, as well as maintaining high levels of performance (Senapathi, 2017).

The Hybrid methodology group utilized Scrum's management paradigms and Extreme Programming's engineering paradigms. This allowed for more accurate tracking of the group's progress, while allowing them the freedom to work on what they deemed necessary to meet product objectives. The team moved in the following phases during a sprint: the planning phase; coding phase; and testing phase (Mushtaq, 2012). The design phase has been omitted, as all functional requirements and design work was completed before each project started. During the planning phase items within the backlog were planned for development and organized sequentially based on priority and dependencies. The coding phase was where all development occurred and centers around coding standards, code ownership, pair programming, and continuous integration (Mushtaq, 2012). Finally, during the testing phase, code written was

tested through unit tests to determine if it passes acceptance tests. The roles for this group were the same as those followed in the Scrum group.

The results of the first experiment were used to determine which methodology to compare against the Hybrid model. Extreme Programming was ultimately chosen to be compared to the Hybrid model in the second experiment. This was due to its higher overall performance in the first experiment comparing Scrum and Extreme Programming.

CHAPTER 4: RESULTS

4.1 First Experiment – Scrum vs. Extreme Programming

Scrum focuses mainly on the project management aspects of development, leaving much to be desired when it comes to the engineering aspects of software development. In contrast, Extreme Programming has the engineering practices, yet it lacks almost all of the management aspects. XP relies heavily on the customer and is not scalable to larger projects (Mustaq, 2012). Several shortcomings and benefits were identified with each methodology during this experiment. By using velocity, defects detected per sprint, and time to completion of projects metrics, insights were gained into the benefits and shortcomings of both methodologies.

One measurement to assess each methodology's use was velocity. Velocity was determined by taking the number of achieved story points per sprint/cycle from completed tasks. Only completed stories counted towards the development team's velocity. In sprint 1, there was a total of 10 story points planned, sprint 2 had 15 planned story points, and sprint 3 had 10 points planned, for a total of 35 story points. The Extreme Programming group was able to score more story points in the first two sprints/cycles by working ahead. They received 7 points in the final sprint because it was all they had left to complete. The Scrum group received 8 story points by completing the UI in the first sprint, 10 points in the second sprint by creating the dosage calculator for the application, and 12 points in the final sprint by completing tasks that had lagged behind. The Scrum group averaged a velocity of 10 story points per sprint, while the Extreme Programming group had an 11.6 average velocity shown in Table 3.

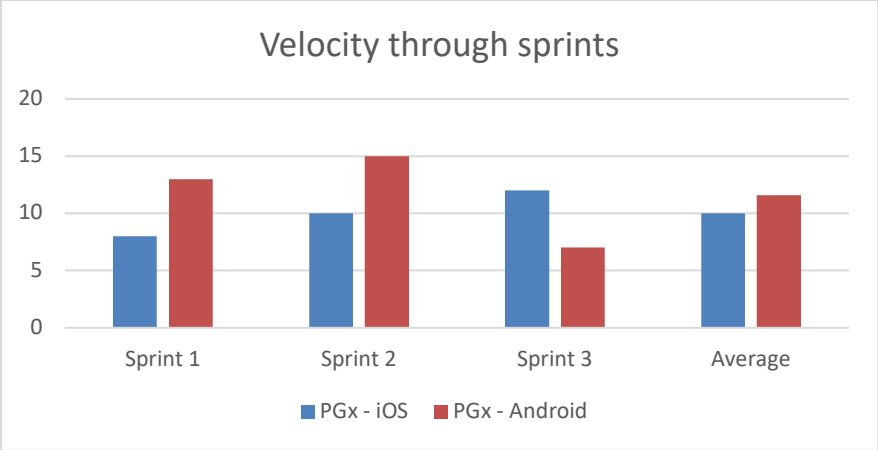

Table 3: Velocity per sprint/cycle (1st experiment)

While sprints are not meant to be restrictive, it would seem that the Scrum team worked at a slower pace initially to stay within the scope of the current sprint. The Scrum group tended to take as long as possible to meet their product deadlines to finish closer to the end date of a sprint. The product deliverable iterations that were developed by the Extreme Programming group were more polished than the Scrum group's as well. This could be attributed to test first programming and Extreme Programming's focus on engineering practices. Through establishing test cases prior to development, the Extreme Programming group was able to produce code with fewer bugs, resulting in higher quality code, as seen in the defect comparison in Table 4.

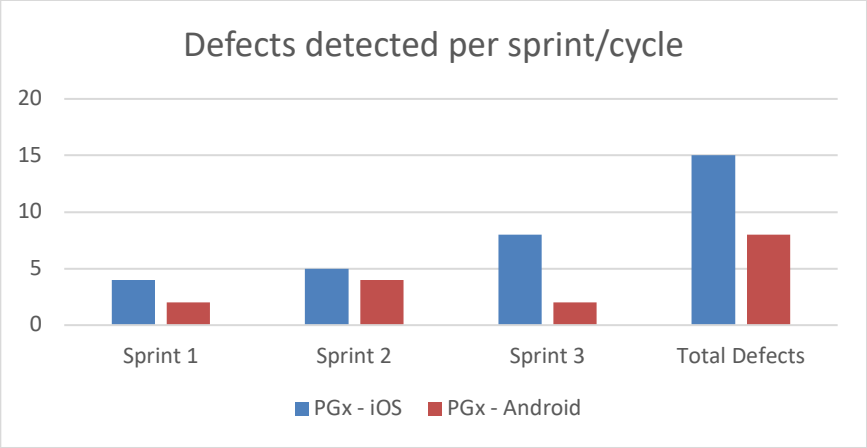

Table 4: Defect comparisons (1st experiment)

The Extreme Programming group was able to finish the application ahead of schedule and reached their final deliverable sooner than expected, even after fixing bugs/defects. In regard to the scoring of completeness of tasks and time taken to complete tasks, the Extreme Programming group again pulled ahead (see Table 5). The Extreme Programming group's average was 8.3 with a total score of 25, while the Scrum group had an average of 5 and a total score of 15. These results indicate that the Scrum group was consistently on schedule, while the Extreme Programming group was ahead of schedule.

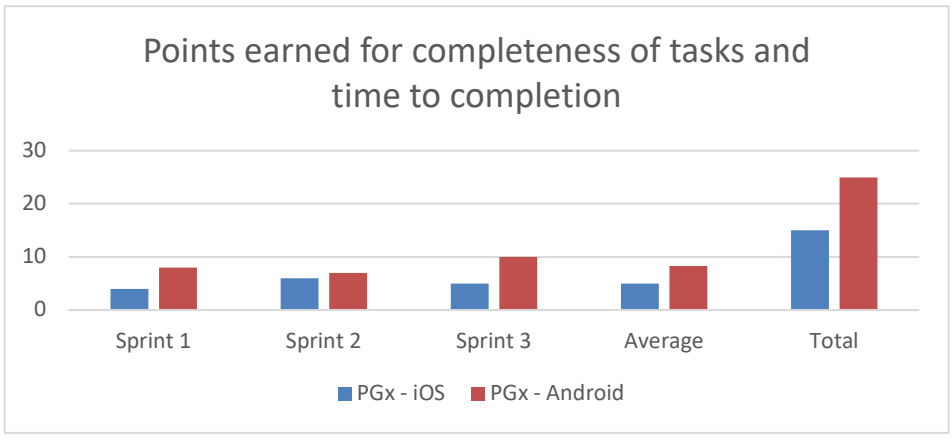

Table 5: Time to completion & completeness of tasks (1st experiment)

The group using the Scrum methodology overall had a more structured project, which allowed for quicker and easier tracking of the group's progress. Over the course of the

experiment, it was noted that the Scrum group was consistently meeting their product deadlines very close to or on schedule, but never ahead of schedule. The Extreme Programming group not only completed their product deadlines, but they would frequently work ahead on features and tasks from future cycles. This was attributed to the Extreme Programming group's higher velocity and their ability to maintain it. Although velocity is not taken into account in Scrum, it is notable that that it took significantly longer for them to build and maintain a steady velocity as seen in Table 3.

4.2 Second Experiment – Extreme Programming vs. Hybrid method

Extreme Programming was used in the second experiment because the methodology outperformed Scrum in all metrics during the first experiment. This experiment focused on Extreme Programming and the Hybrid Scrum-Extreme Programming methodologies. It was hypothesized that by using the Hybrid methodology, the development team would benefit from the pros of both Scrum and Extreme Programming, while negating the cons of both standalone methodologies.

In regard to velocity per cycle, defects detected per cycle, and overall completeness of features and time to completion, both development team's scores were very close. Story points available in the second experiment were as follows for the three sprints. The first sprint consisted of developing the UI and preparing it to receive and fill information from the database. The second sprint was focused on creating the MySQL database and user authentication. The third and final sprint's goals were to populate the UI with data from the database, as well as adding in additional functionality for calls, text, emails and updating user information. There were 7 Story points planned for the first sprint, 15 for the second, and 32 for the third. Both development

teams completed their tasks for the first sprint on time and obtained all story points available. During the second sprint, the database was created and hosted on a server; however, user authentication fell through due to unforeseen complications for both teams. Because of this, both teams earned the 10 points for work completed. The Extreme Programming team earned an additional 8 points in this sprint by populating the UI with user information, while the Hybrid group earned an additional 10 points by completing the Database connection. The totals for sprint 2 were 18 story points for Extreme Programming and 20 for the Hybrid methodology. This left the final sprint with 22 available story points for the Hybrid team and 24 for the Extreme Programming team. In the final sprint, the Hybrid group achieved 8 story points by completing the population of the UI, and the Extreme Programming group achieved 19 points by implementing the call, text and email functionality and establishing the connection to the database. The average velocity of the sprints was 11.6 for the Hybrid group and 14.6 for the Extreme Programming group as seen in Table 6.

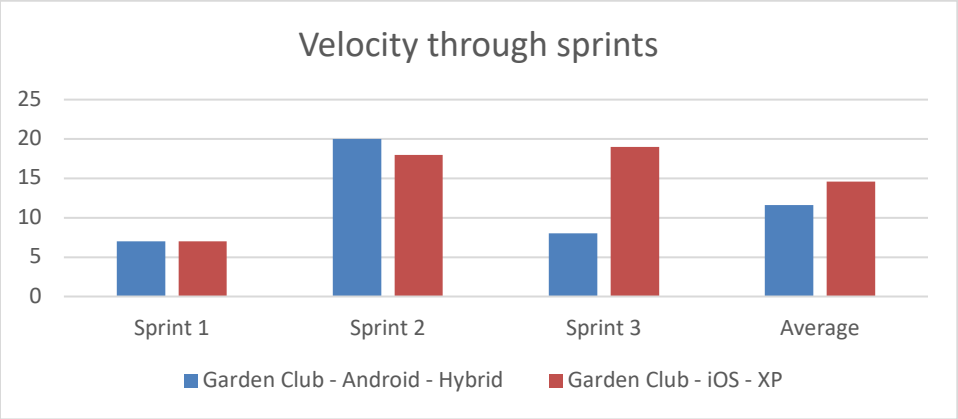

Table 6: Velocity through sprints (2nd experiment)

Defects detected per cycle were low for both development teams. However, the team following the Hybrid methodology had fewer defects than the Extreme Programming development team. This can be seen in Table 7. The Hybrid development team also failed to

implement several features by the end of sprint 3. It is likely that the Extreme Programming development team would have fewer defects if both teams had successfully implemented the same features.

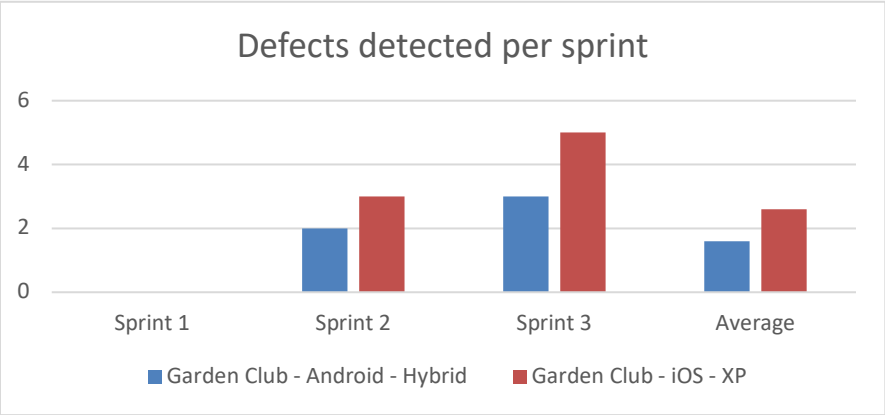

Table 7: Defects detected per sprint (2nd experiment)

In regard to time to completion and the overall completeness of tasks, as shown in Table 8, both development teams received a 10 during sprint 1. In sprint 2, the Extreme Programming group scored 7, while the Hybrid group scored in at 8. The scores for the second sprint did not count the implementation of user authentication, but did take into account each group's additional task completed within the second sprint. For the final sprint, the Extreme Programming group scored 7 by completing all base functional requirements with few defects. The Hybrid group scored a 3 by failing to implement the text, call and email functionality. The Hybrid group averaged in at 7.3, while the Extreme Programming team averaged out to 8.6, with total scores of 22 and 26 respectively.

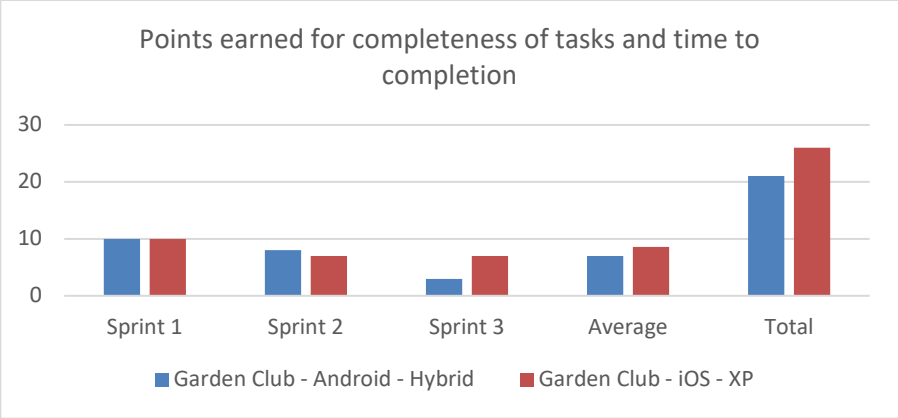

Table 8: Time to completion & completeness of tasks (2nd experiment)

As anticipated, following the first experiment, the development team using the Extreme Programming methodology again completed the majority of their tasks on time if not ahead of schedule and frequently worked ahead on upcoming features to be implemented. The development team working under the Hybrid methodology also completed the majority of their tasks on time, working ahead more frequently than the pure Scrum team of the first experiment.

CHAPTER 5: ANALYSIS AND DISCUSSION

5.1 First Experiment

Both groups in the first experiment completed their application and met the majority of their work product deadlines, as well as producing the final deliverable. They were able to complete an application that was forecast to take two academic semesters to complete in less than one semester. However, the Extreme programming group was able to complete their tasks and ultimately reach the final deliverable faster than the group following the Scrum methodology. This was primarily attributed to the fact that under the guidelines of Extreme Programming the developers were able to shift from one task to another during a cycle, while the group working under Scrum tended to only work on what was defined in a given sprint.

At the end of the first experiment, Extreme Programming was the clear winner as indicated by the results. This finding, in addition to Mathekour et al's (2006) work on Extreme Programming in the classroom, supports the idea that Extreme Programming is an effective classroom approach to teaching Agile Methodologies. This occurred not necessarily due to superiority of the methodology, but because the projects undertaken were better suited for Extreme Programming's attributes. The project size was small. There were 2 developers that were loosely managed and able to organize themselves. However, this came at the cost of being more difficult to track their progress. However, this is not to say that Scrum did not perform well during the experiment. The Scrum group completed the application on time, and since Scrum focuses more on management, it was easier to keep track of their progress and impediments throughout the experiment. A reason for the lower results could have been the possibility of a misconception/miscommunication within the Scrum group that they were only to work on what was defined in their current sprint.

5.2 Second Experiment

The second experiment drew to an end with the group following the Extreme Programming methodology completing their project first and producing a higher quality product overall. The Hybrid group scored lower overall in velocity and completion of the project. However, this group had fewer defects than the Extreme Programming group.

The Hybrid group completed the tasks given during a sprint, but during that process also worked on features of their choosing which they deemed necessary for future sprints. The Hybrid team benefited from the additional freedom and engineering practices of Extreme Programming while retaining the ease of tracking progress of Scrum. Part of the reason the Hybrid team lagged behind the Extreme Programming team was due to their solution to unforeseen technical barriers. While the Extreme Programming group decided to house the database within the application as a temporary solution, the Hybrid group decided to complete the connection to the database so that when the technical barrier was resolved, the application could continue working as planned. This ultimately pulled time away from activities that were to be completed in the final sprint resulting in their overall lower completion score.

In a true classroom setting, the likelihood of encountering such barriers is far less likely. This is due to projects being well scoped, and the students having a firm understanding of the technical conditions. With this being taken into account, the Hybrid methodology would likely out perform a pure Extreme Programming approach.

CHAPTER 6: PEDAGOGY - FINDINGS IN PRACTICE

Due to the differences in focus, project size, and execution of these methodologies, each are likely to be better suited for different points throughout an academic curriculum.

6.1 Scrum

With its management focus, Scrum would allow for greater control and transparency of students' progress and be a better overall fit for intermediate or high-level courses. However, for this approach to work in a classroom setting, there needs to be a substantial amount of planning done beforehand. A project would need to be defined and have both its non-functional and functional requirements outlined for prospective students. These would then need to be broken down into manageable time frames to fit into an academic semester as sprints.

An application where this methodology could be successfully implemented would be an intro to object–oriented programming course. For example, if the students were to create a game as their course project, the development of the game could be split out into sprints with each iteration implementing more functionality. The initial meetings of the group would be to determine the projects objectives, as well as estimation of time required for the phases. Following this meeting, the sprints would focus on developing the proper class structure of the game, creating the GUI and then adding the ability to save the game, loading saved state data and other optimizations that may be needed. Instead of daily Scrum meetings, bi-weekly meetings could be held during course time or by office appointments. The courses lectures could also work in conjunction with the focus of any given sprint. The first third of the course lectures would focus on object-oriented design, followed by GUI elements and then writing to files, loading files or whatever else might be necessary.

The methodologies of Scrum would pose some difficulties to implementation in an academic setting due to the frequency of meetings and involvement from Scrum Master and Product Owner. This could be alleviated by working sprint artifacts into a lesson plan for standard class projects. However, this would still require a significant amount of extra work and is not feasible for projects where students have a large amount of creative liberty.

6.2 Extreme Programming

Extreme Programming's engineering focus would be an excellent fit for early courses in a Computer Science or Information Systems curriculum. The methodology's focus on engineering, best practices, and test first coding would give students a strong foundation to build upon. A setting where students are required to participate in a learning lab would be an ideal scenario to apply the practices of Extreme Programming. The scope of assignments would typically be smaller and better suited for the Extreme Programming methodology. Assignments could be broken down into relatively straight forward User Stories which the students would then complete during lab sessions. In learning lab environments, students could be paired by the professor to complete programming assignments through pair programming (Williams, 2002). At set intervals throughout the course, students could alternate who was at the helm, as well as rotating partners within the class. By doing so, weaker students would have an opportunity to learn from other students. This would also alleviate some of the burden from professors as more well-versed students could help instruct their partners through problems. Since it would be an in-class lab, the need for an on-site customer, the professor, would also be solved.

Extreme Programming would be easier to implement than the Scrum methodology into this particular academic setting. With progress being more difficult to track in Extreme Programming, having the students follow more stringent assignment guidelines in a lab setting

would allow instructors to easily gauge students' progress. If students were to finish a User Story ahead of schedule, they could begin working on the next leg of the assignment or attempt to add additional functionality for extra credit. Ultimately, this would give students insight into working with other developers, increase their understanding of core concepts, reduce the burden of professors in teaching lab environments, and potentially allow students to produce higher quality assignments.

6.3 Hybrid

The hybrid model which attempts to blend the best aspects of both Extreme Programming and Scrum would be well suited for senior level courses and capstones, especially in situations where students are allowed a great amount of creative liberty in their decision of what to develop, such as an upper level software engineering course. This methodology would give students the ability to work freely at their established pace, while allowing professors to track student's progress effectively. Standup meetings could be held weekly or bi-weekly depending on the availability of class time. Sprints would be focused around delivering specific functionality that all student projects should have with other sprint tasks being generated by the student groups. For example, the goal for a sprint could be to have the UI or database connection completed for all groups, with the other sprint tasks being project specific for each group.

Within student groups, one student could act as the product owner during any given sprint and be tasked with managing the group's product backlog. The students could rotate this role after each sprint so that every student would have a chance to play this role. This coupled with the practices of Extreme Programming, such as test first programming, unit testing and pair programming, would in theory allow for greater transparency into students' progress while encouraging the students to follow best practices and produce better products.

CHAPTER 7: CONCLUSIONS AND FUTURE WORK

While the overarching question of this research seems to have been solved, many more questions and possibilities have emerged. Throughout the two experiments conducted the Extreme Programming development teams outperformed both the Scrum and Hybrid development teams. The Scrum and Hybrid teams by no means did poorly, but were lagging in several areas. These findings, however, do not conclude to the superiority of the Extreme Programming methodology. Rather, they highlight the highly situational nature of each of these methodologies. This research gives insights into where and how these three Agile methodologies could be implemented effectively within an academic curriculum. Additionally, by incorporating these methodologies into an academic curriculum, students walk away better prepared to adapt to the corporate work environment.

In future studies further experiments can be held in more controlled environments to identify the most profound advantages, to identify the specific areas where these advantages are most notable, and the specific components of the methodologies that have the largest impact on students' grasp on concepts.

REFERENCES

Agile Alliance. (2017). *What is Extreme Programming (XP)?*. [online] Available at: https://www.Agilealliance.org/glossary/xp/#q=~(filters~(postType~(~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'xp))~searchTerm~'~sort~false~sortDirection~'asc~page~1) [Accessed 29 Oct. 2017].

Angioni, M., Carboni, D., Pinna, S., Sanna, R., Serra, N. and Soro, A. (2006). Integrating XP project management in development environments. *Journal of Systems Architecture*, 52(11), pp.619-626.

Anwer, F., Aftab, S., Muhammad, S. and Waheed, U. (2017). Comparative Analysis of Two Popular Agile Process Models: Extreme Programming and Scrum. *International Journal of Computer Science and Telecommunications*, [online] 8(2), pp.1-7. Available at: https://www.researchgate.net/publication/316845761_Comparative_Analysis_of_Two_Popular_Agile_Process_Models_Extreme_Programming_and_Scrum [Accessed 30 Oct. 2017].

Beck, K. (2003). *Extreme Programming explained*. Boston: Addison-Wesley.

Blom, M. (2010). Is Scrum and XP suitable for CSE Development?. *Procedia Computer Science*, 1(1), pp.1511-1517.

Mar, K. and Schwaber, K. (2018). *Scrum with XP*. [online] Pdfs.semanticscholar.org. Available at: https://pdfs.semanticscholar.org/a84d/52f2841c8b92f1acae85b403eac8db1350c3.pdf [Accessed 20 Dec. 2017].

Mathkour, Hassan, Abloalsamh, Hatim, Assassa, G., Al Dossari, H.. (2006), "Use of Extreme Programming in Software Engineering Education: A Pilot Study", The Engineering research Journal, Minoufia University, Vol 31, No. 1, 2008, pp. 39-48 [Accessed 28 Dec. 2017].

Mitchell, M. and Jolley, J. (2013). *Research design explained*. 5th ed. Belmont, CA: Wadsworth, Cengage Learning.

Musa, F. and Tariq, M. (2017). Agile Methodology: Hybrid Approach Scrum and XP. *International Journal of Scientific & Engineering Research*, [online] 8(4), pp.1405-1409. Available at: https://www.ijser.org/researchpaper/Agile-Methodology-Hybrid-Approach-Scrum-and-XP.pdf [Accessed 1 Nov. 2017].

Mushtaq, Z. and Qureshi, M. (2012). Novel Hybrid Model: Integrating Scrum and XP. *International Journal of Information Technology and Computer Science*, [online] 4(6), pp.39-44. Available at: http://www.mecs-press.net/ijitcs/ijitcs-v4-n6/IJITCS-V4-N6-6.pdf [Accessed 23 Dec. 2017].

Rigby, D., Sutherland, J. and Takeuchi, H. (2016). *Embracing Agile*. [online] Harvard Business Review. Available at: https://hbr.org/2016/05/embracing-Agile [Accessed 24 Nov. 2017].

Rubin, K. (2013). *Essential Scrum*. Upper Saddle River, NJ: Addison-Wesley.

Schwaber, K. and Sutherland, J. (2016). *Scrum Guide | Scrum Guides*. [online] Scrumguides.org. Available at: http://www.Scrumguides.org/Scrum-guide.html [Accessed 28 Oct. 2017].

Scrumalliance.org. (2013). *Why Scrum? | State of Scrum Report - Scrum Alliance*. [online] Available at: https://www.Scrumalliance.org/why-Scrum/state-of-Scrum-report/2013-state-of-Scrum [Accessed 15 Jan. 2018].

Scrum.org. (2018). *What is Scrum?*. [online] Available at: https://www.scrum.org/resources/what-is-scrum [Accessed 26 Nov. 2017].

Senapathi, M. and Drury-Grogan, M. (2017). Refining a model for sustained usage of Agile methodologies. *Journal of Systems and Software*, 132, pp.298-316.

Steghöfer, J., Burden, H., Alahyari, H. and Haneberg, D. (2017). No silver brick: Opportunities and limitations of teaching Scrum with Lego workshops. *Journal of Systems and Software*, 131, pp.230-247.

Stellman, A. and Greene, J. (2014). *Learning Agile: Understanding Scrum, XP, Lean, and Kanban*. 1st ed. Sebastopol, California: O'Reilly Media, pp.2-241.

Takeuchi, H. and Nonaka, I. (1986). The New New Product Development Game. *Harvard Business Review*, (64), pp.137-146.

The Official 360logica Blog. (2017). *The Importance of Different Agile Methodologies Included in Agile Manifesto - The Official 360logica Blog*. [online] Available at: http://www.360logica.com/blog/the-importance-of-different-Agile-methodologies-included-in-Agile-manifesto/ [Accessed 25 Nov. 2017].

Williams, L. (2010). Agile Software Development Methodologies and Practices. *Advances in Computers*, 80, pp.1-44.

Williams, L., Yang, k., Wiebe, E., Ferzli, M., Miller, C. (2002). Pair Programming in an Introductory Computer Science Course: Initial Results and Recommendations, OOPSLA Educator's Symposium 2002, Seattle, WA, November 2002.

Wood, S., Michaelides, G. and Thomson, C. (2013). Successful extreme programming: Fidelity to the methodology or good teamworking?. *Information and Software Technology*, 55(4), pp.660-672.