Investigating the Benefits of Introducing Process-Oriented Life Cycle Development Models to Improve Students Appreciation for Agile Methods

Omar Ochoa, Miralda Rodney, Massood Towhidnejad
Department of Electrical, Computer, Software & Systems Engineering
Embry-Riddle Aeronautical University
Daytona Beach, FL 32114, USA
{ochoao, towhid}@erau.edu, rodneym@my.erau.edu

Salamah Salamah
Department Computer Science
Univeristy of Texas at El Paso
El Paso, TX 79968, USA
isalamah@utep.edu

***Abstract*—In this paper, we explore the benefits of teaching heavyweight process life cycle models to young software engineering students to better prepare them in the use of agile methods. The benefits of agile methods compared to process-heavy models include the ability to respond faster to the changing needs of customers and the short feedback loop between customers and developers. Since agile methods depend significantly on the competence of the individual, teaching traditional approaches (e.g. waterfall) can be advantageous to the understanding of students about what the major activities associated with software development are, by providing a well-defined structure that naturally aligns with the novice student. Towards this goal, we conducted a survey with different types of software engineers, ranging from novice undergraduates to software engineering professionals. The survey provides data on the exposure of traditional and agile process, the level of experience in using traditional and agile process models, and the perceived most optimal learning sequence. Based on the analysis of the collected data, we argue that teaching traditional software engineering process-oriented approaches prior to introducing agile methods, is highly beneficial to students' understanding and optimal use of agile techniques.**

Keywords—Software Engineering Education, Agile Methods, Traditional Waterfall Lifecycle

I. INTRODUCTION

With the emerging adoption of agile methods for software development in industry, there is a pressing need to align the educational requirements of students in computer science and software engineering programs with those of future employers. The use of agile practices is significantly on the rise within software development organizations. The benefits of agile methodologies compared to process-heavy models include the ability to respond faster to the changing needs of customers' changing business needs and the short feedback loop between customers and developers. Much of the heavy process activities used in traditional methods are substituted in agile approaches by relying on developers to form self-organizing and motivated teams who deliver technical excellence. However, since agile methods depend significantly on the competence of the individual, teaching traditional approaches (e.g. waterfall) can be advantageous to the learning of the student by providing a well-defined structure that naturally aligns with the novice student. This paper explores the benefits of exposing students to the traditional methods as they are mastering software engineering concepts before immersing those students to agile methods. Some literature supports the importance of adherence to the process, a key concept, early in software engineering education to build a foundation that will ease a students' ability to transfer the knowledge to new situations [1]. However, to examine this concept further, this study focuses on exploring the effects of the learning sequence of the different life cycle processes to improve student's appreciation for agile methods by conducting a survey with different types of software engineers, ranging from novice undergraduates to software engineering professionals. The data collected from the survey responses leads us to posit that it is more beneficial to the learning of a student to be exposed to traditional approaches before learning about agile methods.

The paper is structured as follows: Section II provides the motivation for this work. Section III gives a brief background on development methods and lifecycles, and provides a distinction between those considered traditional or agile. Section IV describes some similar work in assessing the learning sequence of processes. Section V presents the approach and survey questions. Section VI highlights the results and analysis of the survey conducted. Section VII presents the conclusions of this work. Section VIII concludes with providing lessons-learned from this research and future work.

II. MOTIVATION

The current research assessing the use of agile methods in software engineering education as well as professional organizations shows that several problems have been reported from projects, course work, and industry. A study found that high quality results are produced by software engineers with technical expertise, which does not reflect the typical software team process as a typical software team does not apply agile practices as thoroughly [2]. In that work, the authors noted from their study on the use of agile in industry that there are

978-1-5386-1174-6/18/$31.00 ©2018 IEEE

too few software engineers with the skills necessary for the employment of agile development and that current software engineering curricula do not prepare the students with these skills. To remedy this, it is proposed that a course dedicated to agile principles should become a part of typical software engineering education. As a result of the study and based on the pyramid of agile competences described, before agile software development methods can be taught, students need a *strong foundation* in engineering and management practices [2]. Another paper states that additional research efforts will be needed to help software engineering education community determine how to incorporate agile practices in software engineering courses [3]. The paper states that a sequence must be chosen for dealing with the agile methods and more traditional processes. The authors argue that the options are 1) agile and traditional processes are to be taught concurrently, 2) agile software development can be taught as a stepping-stone to more formal processes, or 3) traditional development processes can be taught to give students grounding in the more formalized processes before being exposed to the lighter weight approaches. A conclusion that can be drawn from this work is that the traditional development process should be taught *first* to provide students with a *strong foundation* before introducing agile.

Another paper showed that theoretical communication of agile practice does not lead to their adoption in projects [4]. It is important to provide the correct environment in which the process is emphasized, and students are not distracted by the technical difficulties associated with the software development process in a project. The authors' goal is to align the learning and teaching activities to the learning outcomes by providing the frame of reference through theoretical materials and practical experience. The paper presents that it is difficult for students to appreciate agile software development methods because the students are unable to follow the process due to a lack of technical skills, i.e., a *strong foundation*, which can demand a more focused approach to teaching agile software development methods.

In teaching scrum, an author concluded that coaching along with self-organization is important to beginners. Coaching being the most important factor in delivering a quality product demonstrates that beginner or novice students require structure and guidance to apply the necessary skills when using agile software development [5]. Furthermore, the disciplined process paradigm is the more efficient process to teach inexperienced students in software engineering [6]. In designing the course, the author expects that students have already mastered the traditional methods of software development before taking the capstone course under scrum. Leading to the conclusion that to effectively apply an agile method the student must have mastered the traditional concepts, i.e. developed a *strong foundation*, of software engineering concepts.

To build expertise, i.e. a *strong foundation,* in any engineering discipline a foundation must be built for future expertise. A learning process must be developed to support the acquisition of expertise in the engineering field where the knowledge is structured around key concepts and principles of the discipline to facilitate students' abilities to access and transfer the knowledge when necessary. To develop the expertise of inexperienced engineers, the curriculum must provide multiple opportunities to master newly acquired skills while also integrating their previously learned skills [1]. Instructors must design courses to follow a structured sequence such that the learning experiences guides the students in developing increasing levels of competence in the core concepts such as analysis and design [1]. This structuring of engineering curriculum similar to the waterfall process where a phase must be completed before the next phase can begin and the previous knowledge is applied and integrated. The agile software development method can be difficult to understand by inexperienced software engineering students, therefore guiding the students' education in a systematic, e.g., process-oriented, way will allow students to build an expertise in software engineering that can transfer to any situation.

There has been work aiming to determine the critical success factors for software development projects using traditional or agile methods [7]. The authors note that research is focused on comparing the traditional processes to agile methods in projects instead of providing reasons why following either approach was successful for the group engaged in the project [7]. One of the important items presented on this paper is that the development team's skill, expertise, and experience with software development processes are highly critical indicators of success in using agile methods when compared with traditional, plan-driven methods. An inference can be made that, to use agile software development methods, a student must develop expertise in software engineering concepts and needs to have a development process background. The authors found in contrasting the agile method and traditional software development process that when there is a lack of experience, it is more appropriate to use a software development method with more structure such as a traditional development process. When a team has the expertise and are highly committed, agile software development methods should be used. Therefore, it is evident that novice engineers must first gain expertise through a plan-driven method, e.g., process-oriented, before using the agile approach. Additionally, the published work states that inexperienced developers with a limited skill set perform best following a traditional, guided lifecycle process approach.

To adequately prepare software engineering students there is a need to develop a curriculum that better prepares software engineering students to use agile methods by instructing them using a plan-driven approach. Such curriculum can provide an opportunity to build a *strong foundation* in software engineering students, which then can be applied in excelling under an agile setting.

III. BACKGROUND

A. Traditional Software Development

The traditional approach to software development is a heavyweight methodology. Such approach involves the use of process-oriented methods that are document-driven and require following a disciplined plan [5]. These disciplined strategies have well-defined life cycle phases and templates for how to complete the required artifacts [8]. These defined phases and templates guide software engineers to successfully completing

the tasks demanded in each phase of the life cycle. Commonly used traditional methods are listed below.

Waterfall

The Waterfall model of software development is a linear sequential approach that is simple to understand and easy to implement [9]. It consists of five stages: requirements specification, design, implementation, verification, and maintenance. Each stage results in a set of artifacts that are utilized at the next stage of development. As the life cycle progresses, each stage cascades into the next until the completion of the project.

Rapid Application Development (RAD)

This method of development can be considered as a trial and error process, where attempts at providing the solution is exchanged between a development team and a client. Such approach allows feedback to be provided early in the project [9]. The RAD model uses prototyping as a mechanism for iterative development [10]. In this model, there is an initial investigation of the problem, a short development life cycle for prototype creation, implementation and lastly maintenance of the product.

Incremental Model

The incremental model of software development is similar to the waterfall model but is iterative and each sequence produces deliverable increments to the product [11]. The process does not allow for iterations within a single increment. The model is made up of a series of waterfall activities where the number of increments after the first is determined by how much the product needs to be improved based on feedback [10].

V-Model

The V-Model of software development contains the same stages of development as the waterfall model. In addition, each phase in the life cycle is associated with a testing counterpart, which results in a V shape [12]. Fig. 1 shows the layout of the different phases in the V-Model process with the corresponding verification artifact for that phase. The process starts with the analysis and design phase, followed by system requirements, software requirements, preliminary requirements, detailed design, and coding. The testing and integration phase involves unit testing, component testing, software integration, and acceptance test and system integration [12].

Spiral Model

Spiral Model of software development focuses on risk assessment and managing risk [11]. In this method, there are four phases: objective setting, risk analysis, development and planning [12]. As the project is being completed, the spiral progresses until a prototype is built, verified against its requirements and validated for testing [10]. Risk is considered in all cycles of the spiral model to estimate the time and effort that will be dedicated to development activities, it is a control mechanism to ensure delivery of the product.

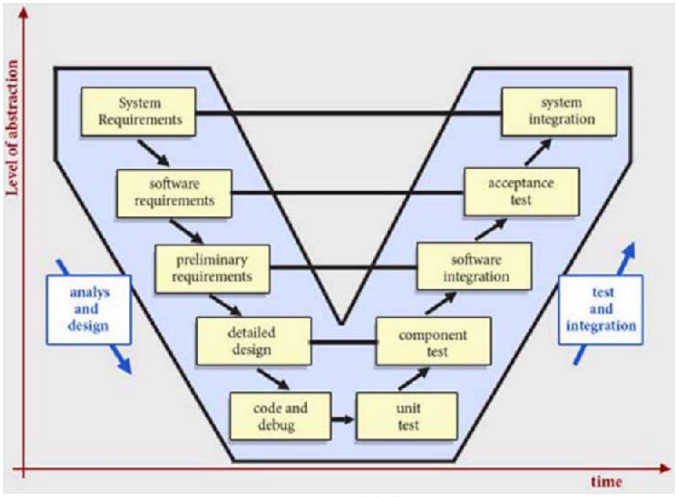

Figure 1: V-Model diagram.

B. Agile Software Development

The agile approaches to software development are lightweight methods that are people-oriented, adaptable to change, and characterized by short incremental iterations. To better develop software, the agile manifesto was created by a group of motivated and experienced individuals that value the customer interactions over following a plan [13]. All agile methods follow the twelve principles backing the agile manifesto, which involves continuous delivery of working software, a high level of customer involvement, flexibility to change, face-to-face communication, and improvements to the process [12]. The benefits of these methods are better software quality, improved productivity, frequent delivery and customer satisfaction [14]. Some agile software development methods are listed below.

Extreme Programming (XP)

XP is an agile software development method where each iteration consists of a life cycle where story cards are used to determine requirements and create tasks [14]. XP includes a planning game to determine how long the tasks will take to be accomplished. Pair programming is used to develop the software while unit testing and acceptance testing is used for verification and validation, which lead to small releases to the customer.

Scrum

An agile software development method that manages software development in iterations called sprints [14]. Each sprint involves a product backlog, sprint backlog, daily scrum meetings. Less emphasis is placed on a process driven framework [10] but instead, the focus is on daily progress and process improvement through retrospective meetings after each sprint and planning before the next sprint.

Kanban

Kanban method emphasizes scheduling work to deliver the product just in time for implementation [14]. A Kanban board is used to visualize the work to be done by categorizing the work completed, to be done and in-progress. The Kanban board limits works in progress because tasks are started when they are needed. This method allows for continuous delivery of

increments, waste minimization and maximizes the productivity of developers.

Test Driven Development (TDD)

TDD is an approach where tests are written first at the start of the development life cycle and then the system is developed based on the specific test case [12]. This method produces code at each iteration that passes a test case, which is then refactored to improve the design and added to the previous iteration code. The steps are the following: add a new automated test, run all the test cases to ensure it fails, write the code to pass the test case, run the tests over again and refactor the code [15].

Feature Driven Development (FDD)

FDD is an agile software development process that consists of five main activities that are performed iteratively [12]. The activities are as follows: develop an overall model of the system, build a features list, develop a plan based on which features to implement, design the specific feature chosen for the iteration and build it. This model focuses on the design and building phases to develop tangible results and provide progress information about the project to the client [15].

Crystal

Crystal agile software development methods can be used for various project depending on size criticality, complexity and number of people involved [15]. Crystal clear methods focus on people and are applied to small teams where projects are small and not safety-critical [12]. Crystal yellow is for medium size projects, crystal orange for large project and crystal red for very large projects. Each opacity of the crystal method can be tailored according to the needs of a specific project [15].

Dynamic Systems Development Method (DSDM)

DSDM is an iterative and incremental method that focuses on building important functionality first [15]. It consists of six phases: pre-project phase feasibility study, business study, functional model iteration, design and build iteration, and implementation and post project phase. The principles of DSDM are focusing on business needs and providing quality, delivering the product on time for continuous feedback and building incrementally.

Rational Unified Process (RUP)

An iterative approach for object-oriented systems based on phases and workflows [12]. The phases are inception, elaboration, construction and transition. At each phase, the effort dedicated to a workflow along the life cycle of the project is considered. The workflows are business modeling, requirements, analysis and design, implementation, test, deployment, configuration and change management, project management and environment. RUP involves use cases and models as the foundation for development [10].

IV. Related Work

Research has been conducted on the different software development processes to determine their effectiveness in academic settings to teach the technical competencies of the software engineering discipline to students in universities across the globe. The research is driven by the need to understand and consequently develop curricula to bridge the gap between industry expectations of software engineering graduates and their current skill set.

The paper written by authors from Universidad de Chile noted that disciplined processes are useful for inexperienced developers. The authors state that little research has been done to support the learning process when using disciplined development strategies [8]. Furthermore, in a systematic mapping study [16], only eight studies out of 104 that report the use of traditional processes to support the practical experiences in software engineering courses. Similarly, the work presented in this paper examines the need for introducing traditional process-oriented life cycles first to improve student's use of agile methods.

Another paper discusses agile development in practice and provides input for software engineering education [17]. The study focused on addressing the challenges associated with adopting agile software development practices through agile coaching and active collaboration. Software engineering practices are a foundation that assures agile processes can be used successfully. A software engineering module was developed to reach the objective of teaching agile software development by first building a foundation and then focusing on agile values through collaboration practices. The first and second semester courses in software engineering follow a plan-driven process. The third semester follows an iterative development process using the rational unified process. The last semester focuses on agile development methods after mastering the technical and collaboration practices necessary for success. This paper reported, from conducting interviews, that a more structured curriculum is necessary to learn software engineering concepts. Likewise, the outcome from the interviews presented in this paper show that the optimal sequence should be a more process-oriented lifecycle and then an agile method.

Finally, some of the challenges associated with the adaptation of agile process in a multi-disciplinary capstone project have been previously published [18]. The authors clearly see the difference between the preparations of students who have previously been introduced to the formal development process versus the ones who have their first process exposure to agile process.

V. Approach

To assess how software engineering students and professionals perceive the traditional software development processes and agile methods, we developed a survey to gauge the exposure of software engineers to agile and/or traditional software development processes as well as to determine their perceptions of the processes based on their experiences. The goal of the survey is to determine the effects of the order of the exposure to the development processes and their competencies in these processes. The survey consists of multiple-choice questions and an open-ended question. The open-ended question serves to provide insights to determine the true experience of the individual by allowing them to explain their

logic in answering the previous questions. The survey is concise such that more people are inclined to respond and provide actual feedback.

The survey was given to software engineering university students as well as novice and experienced software engineering professionals. The goal of the survey is to determine whether the results will support the hypothesis of the research by highlighting the difficulties that inexperienced software engineers have using agile software development processes. More specifically, the results of the survey are used to determine if there is any correlation between having been exposed to a traditional, structured software development process and the perceived difficulties in using agile methods afterwards. Additionally, the survey also inquires on what the optimal learning sequence should be.

The survey consisted of the following nine questions with the corresponding multiple-choice answers.

Q1. Which of these best describes your experience with waterfall process models:

1) *No experience/knowledge*
2) *Only in Academia*
3) *Primarily in Academia, with some in Industry*
4) *Primarily in Industry, with some Academia*
5) *Only in Industry*

Q2. Which of these best describes your experience with agile process model:

1) *No experience/knowledge*
2) *Only in Academia*
3) *Primarily in Academia, with some in Industry*
4) *Primarily in Industry, with some Academia*
5) *Only in Industry*

Q3. Please select the option that best reflects your level of experience in waterfall:

1) *<1 Year – You have a general idea of how the process works.*
2) *1-2 Years – You have used the process in some capacity.*
3) *3-5 Years – You feel comfortable using the process in its entirety on a project.*
4) *>5 Years – You have leadership experience in the use of the process on a project.*

Q4. Please select the option that best reflects your level of proficiency in agile:

1) *<1 Year – You have a general idea of how the process works.*
2) *1-2 Years – You have used the process in some capacity.*
3) *3-5 Years – You feel comfortable using the process in its entirety on a project.*
4) *>5 Years – You have leadership experience in the use of the process on a project.*

Q5. My initial experience in using an agile approach was:

1) *Mostly negative*
2) *Somewhat negative*
3) *Somewhat positive*
4) *Positive*

Q6. In which order were you exposed to the waterfall and agile processes:

1) *Waterfall then Agile*
2) *Agile then Waterfall*
3) *Agile and Waterfall concurrently*

Q7. Select the most appropriate statement:

1) *I found the transition from waterfall process to agile to be seamless.*
2) *I found the transition from waterfall process to agile to be extremely difficult*
3) *Learning the waterfall process did not have any effect in my transition to agile process.*

Q8. In your experience, which of the following would be the most optimal learning sequence:

1) *Waterfall then Agile.*
2) *Agile then Waterfall.*
3) *Agile and Waterfall concurrently.*

Q9. (Optional) Please comment on your answers to questions 7 and 8. The answer provided below will be used to understand your thought process.

The first and second questions are used to determine the perceived experience a respondent had with a traditional process (i.e., waterfall) and an agile process in academia or in industry, respectively. The third and fourth questions gauge the level of proficiency a respondent has with a traditional and an agile process. The fifth question focuses on the impression a respondent had of the agile process when they were first introduced to the process. The sixth question asked the order in which the respondent learned the process, whether the respondent was introduced to both process models at the same time or if a process was learned before the other. The seventh question is used to determine the transitioning experience from a traditional process to an agile method. The eighth question asks which learning sequence the respondent perceives is the most optimal for learning the software development skills associated with the processes and building a strong foundation of software engineering concepts.

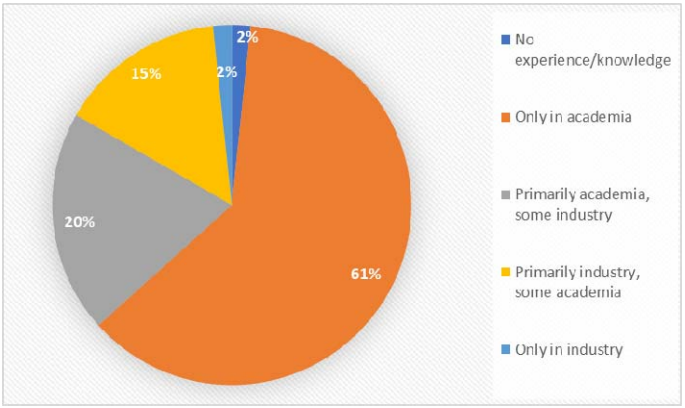

Figure 2: Results of Q1.

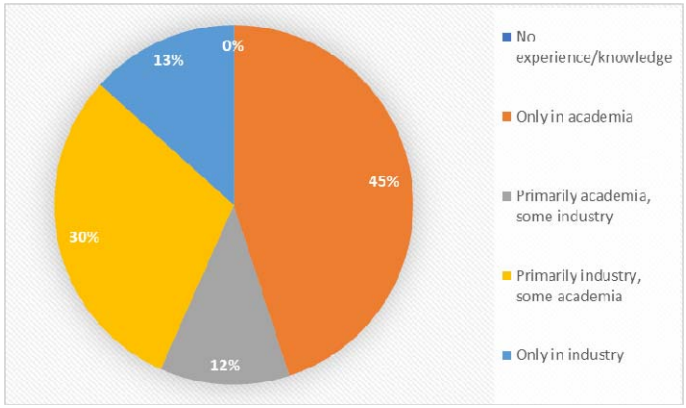

Figure 3: Results of Q2.

VI. Survey Results

The survey was disseminated via social media and in classrooms at both universities and answered by 60 respondents, including 18 professional software engineers. The remaining student respondents included 17 graduate students and 25 undergraduate students. The following survey results are broken down by each question: Fig. 2 displays the results of question one. From the figure, it shows that many of the respondents of the survey, 81%, have been exposed to waterfall process models primarily in academia with some industry. This meets the expectation as most software engineers are introduced to traditional processes primarily in academia.

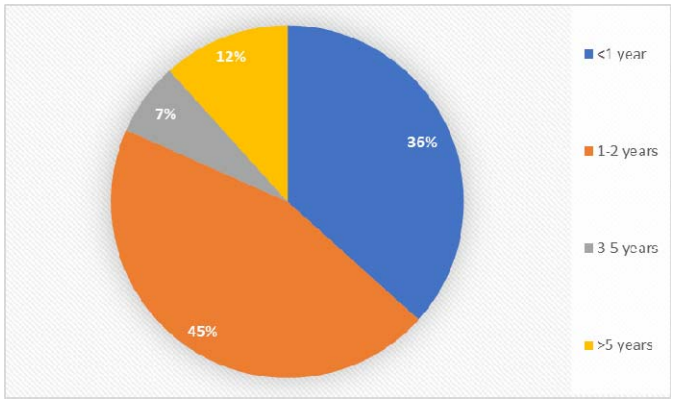

Figure 4: Results of Q3.

Fig. 3 shows that based on the answers from question two, 45% of the respondents have academic experience with the agile methods followed by the 30%, which have primarily industry experience with some academic experience. A simple observation can be made from the results of these two questions, there is a bigger exposure to agile methods within industry compared to traditional process-oriented life cycle approaches.

Fig. 4 shows the results of the third question, which illustrates that 81% of the respondents have less than two years of experience using a waterfall process model, which could denote a lack of experience in using process-oriented models.

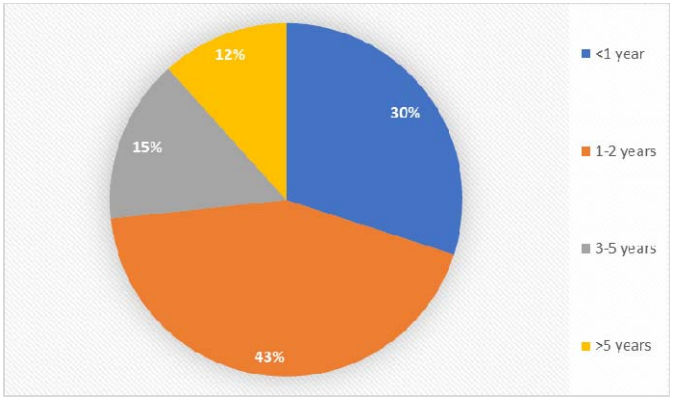

Figure 5: Results of Q4.

Fig. 5 shows the results of the fourth question, which reveals that 73% of respondents have less than two years of experience using an agile process model. The lack of proficiency is consistent with the answers to the previous question. The results of question three and four reflect that most of the respondents were young software engineers. In addition, the results demonstrate that respondents have more working experience with agile methods instead of traditional process-oriented models.

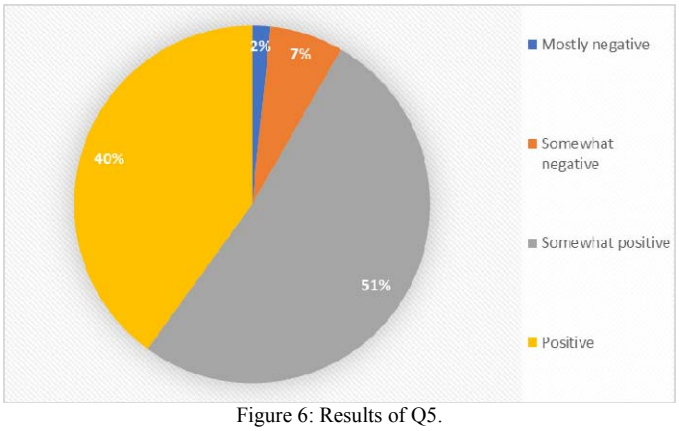

Figure 6: Results of Q5.

Fig. 6 shows the fifth question results, which reveals that 91% of the respondents of the survey had a positive introductory experience to the agile software development process.

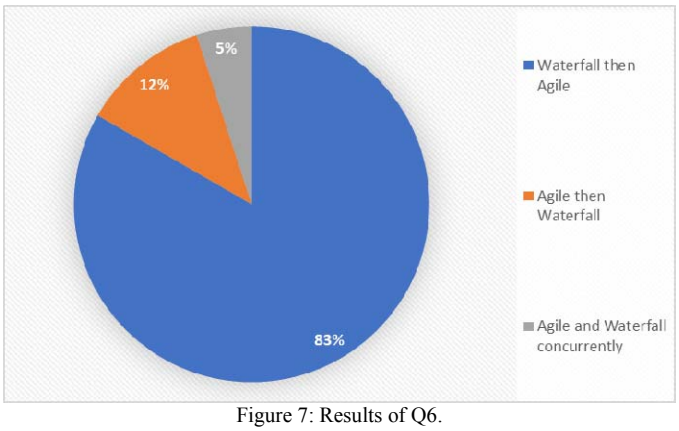

Figure 7: Results of Q6.

The results of the sixth question are shown by Fig. 7, which reveal that most respondents were introduced to a waterfall process model followed by an agile process model.

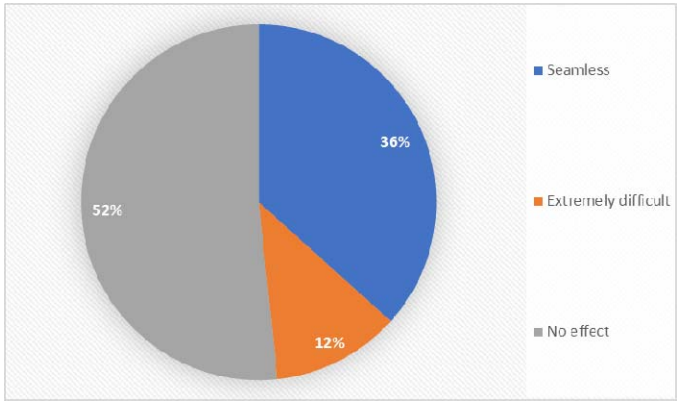

Figure 8: Results of Q7.

Fig. 8 shows that many of the respondents perceived no effect in their transition from a waterfall process model to an agile process model. Considering questions six and seven, 83% of the respondents were exposed to waterfall before being exposed to agile and 88% did not have any difficulty in adopting the agile development process. Of the 83% of respondents that were exposed to waterfall before being exposed to agile, a significant number, i.e. 90%, of them were introduced to a process-oriented life cycle first were better able to transition to an agile method.

The results of question eight, as shown by Fig. 9, reveals that 67% of respondents perceive that the optimal learning sequence of software development processes is learning the traditional software process model first followed by the agile methods.

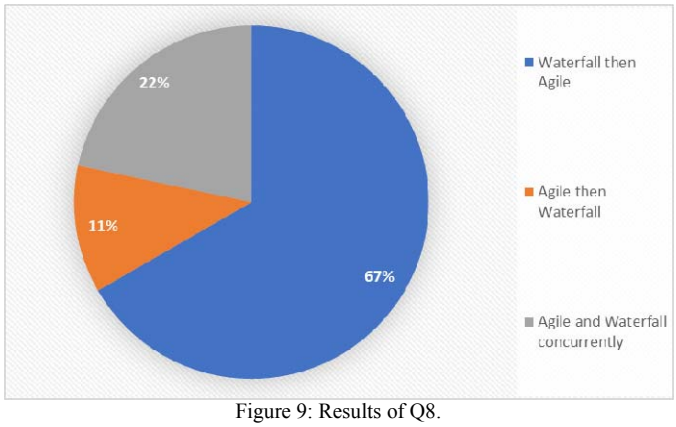

Figure 9: Results of Q8.

Some conclusions can be drawn from the question 8: of the 67% that perceive the optimal learning sequence for software development processes, including 12.5% that have more than 3 years of experience using both waterfall and agile process models. The remaining were respondents who had less experience using both development process models. The result shows that inexperienced engineers perceive that learning the more structured approach as more beneficial.

The open-ended question of the survey provided important insight into the perceptions of the respondents. Some comments from respondents that perceived that the optimal learning sequence should be waterfall then agile were:

- "Waterfall is a good foundation process to learn."
- "It is always best to learn the basics and fundamentals, it helps understand the real software development process."
- "Waterfall allows the developer to take its time and fully understand every stage in development."
- "The waterfall process is a good starter for new software engineers as it takes you into detail of all the necessary steps in the software development process."
- "I strongly believe you should master a disciplined approach that will teach you what the software development cycle is all about before you can move on to agile frameworks."

VII. Conclusion

This paper argues that teaching traditional life cycle models to students before introducing them to agile methods is a more efficient approach to improve student appreciation for both methods. Once students go through a traditional waterfall approach, through which they gain a mastery over the software engineering concepts, they are better prepared to compare traditional life cycles with agile methods. This in turn, allows students to gain a better understanding of the strengths and weaknesses of both approaches. The literature demonstrates that students fail in using agile process models due to their lack of a *strong foundation* in software engineering concepts. Fundamental software engineering concepts should be taught to students in a process-oriented model where each concept can be taught within the strong boundaries of the process. This approach can provide a strong setting to allow students to master the necessary software engineering concepts and develop a *strong foundation* before being introduced to the agile methods. These conclusions are supported by the results of the survey, as most respondents introduced to a waterfall process model before agile, perceived the transition to an agile process seamless or without any effect.

VIII. Lessons Learned and Future Work

While the study conducted as part of this work provided a sample of opinions and feelings from a small number of current software engineering students and professionals, it is necessary to increase the sample size of individuals surveyed to give results that are more accurate. In addition, it will be worthwhile to segregate the survey questions and results based on the level of experience of those surveyed (e.g. undergraduate vs. graduate students vs. early career software engineers). Individual groups can provide different insights that further signifies the results of the study. Furthermore, to provide more qualitative insights, a subset of these groups will be interviewed to capture and document their specific experiences and further reinforce the hypothesis of this work.

Finally, because agile methods are quite dependent on individual competencies, a future survey and analysis will take into account the different competencies (i.e., years of programming experience, experience with continuous integrations, etc.) of those surveyed and to examine if such competencies have an effect on the feelings toward agile and traditional development models. Future work will also include a curriculum of specific courses that can assist students in developing the *strong foundations* under a traditional life cycle that will lead to a better adoption of agile methods.

REFERENCES

[1] T. Litzinger, L. Lattuca, R. Hadgraft and W. Newstetter, "Engineering Education and the Development of Expertise", Journal of Engineering Education, vol. 100, no. 1, pp. 123-150, 2011.

[2] M. Kropp and A. Meier, "Teaching agile software development at university level: Values, management, and craftsmanship", 2013 26th International Conference on Software Engineering Education and Training (CSEE&T), pp. 179-188, 2013.

[3] G. Hislop, M. Lutz, J. Naveda, W. McCracken, N. Mead and L. Williams, "Integrating Agile Practices into Software Engineering Courses", Computer Science Education, vol. 12, no. 3, pp. 169-185, 2002.

[4] J. Steghöfer, E. Knauss, E. Alégroth, I. Hammouda, H. Burden and M. Ericsson, "Teaching Agile - Addressing the Conflict Between Project Delivery and Application of Agile Methods", Proceedings of the 38th International Conference on Software Engineering Companion - ICSE '16, pp. 303-312, 2016.

[5] V. Mahnic, "Teaching Scrum through Team-Project Work: Students' Perceptions and Teacher's Observations", International Journal of Engineering Education, vol. 26, no. 1, pp. 96-110, 2010.

[6] V. Mahnic, "A Capstone Course on Agile Software Development Using Scrum", IEEE Transactions on Education, vol. 55, no. 1, pp. 99-106, 2012.

[7] A. Ahimbisibwe, R. Cavana and U. Daellenbach, "A contingency fit model of critical success factors for software development projects", Journal of Enterprise Information Management, vol. 28, no. 1, pp. 7-33, 2015.

[8] M. Marques, S. Ochoa, M. Bastarrica and F. Gutierrez, "Enhancing the Student Learning Experience in Software Engineering Project Courses", IEEE Transactions on Education, vol. 61, no. 1, pp. 63-73, 2018.

[9] A. Saxena and P. Upadhyay, "Waterfall vs. Prototype: Comparative Study of SDLC", Imperial Journal of Interdisciplinary Research, vol. 2, no. 6, pp. 1012-1015, 2016.

[10] N. Ruparelia, "Software development lifecycle models", ACM SIGSOFT Software Engineering Notes, vol. 35, no. 3, p. 8, 2010.

[11] A. Alshamrani and A. Bahattab, "A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model", International Journal of Computer Science Issues, vol. 12, no. 1, pp. 106-111, 2015.

[12] G. Kumar and P. Kumar Bhatia, "Comparative Analysis of Software Engineering Models from Traditional to Modern Methodologies", 2014 Fourth International Conference on Advanced Computing & Communication Technologies, pp. 189-196, 2014.

[13] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. Martin, S. Mellor, K. Schwaber, J. Sutherland and D. Thomas, "Manifesto for Agile Software Development", Agilemanifesto.org, 2001. [Online]. Available: http://agilemanifesto.org/. [Accessed: 26- Apr- 2018].

[14] G. Matharu, A. Mishra, H. Singh and P. Upadhyay, "Empirical Study of Agile Software Development Methodologies", *ACM SIGSOFT Software Engineering Notes*, vol. 40, no. 1, pp. 1-6, 2015.

[15] F. Anwer, S. Aftab, U. Waheed and S. Muhammad, "Agile Software Development Models TDD, FDD, DSDM, and Crystal Methods: A Survey", International Journal of Multidisciplinary Sciences and Engineering, vol. 8, no. 2, 2017.

[16] M. Marques, A. Quispe and S. Ochoa, "A systematic mapping study on practical approaches to teaching software engineering", 2014 IEEE Frontiers in Education Conference (FIE) Proceedings, 2014.

[17] M. Kropp and A. Meier, "Collaboration and human factors in software development: Teaching agile methodologies based on industrial insight", 2016 IEEE Global Engineering Education Conference (EDUCON), 2016.

[18] R. Stansbury, M. Towhidnejad, J. Clifford, M. Dop, Agile Methodologies for Hardware/Software Teams for a Capstone Design Course: Lessons Learned, Proceedings of the 118th Annual Conference for the American Society for Engineering Education (ASEE 2011), June 2011.