Leveraging Agile Software Methodologies Within Software Development To Introduce A Novel Educational Software Methodology

Montserrat Guadalupe Molina
University of Texas at El Paso

Follow this and additional works at: https://scholarworks.utep.edu/open_etd

Part of the Computer Sciences Commons, and the Engineering Commons

Recommended Citation
https://scholarworks.utep.edu/open_etd/3998

This is brought to you for free and open access by ScholarWorks@UTEP. It has been accepted for inclusion in Open Access Theses & Dissertations by an authorized administrator of ScholarWorks@UTEP. For more information, please contact lweber@utep.edu.
LEVERAGING AGILE SOFTWARE METHODOLOGIES WITHIN SOFTWARE DEVELOPMENT TO INTRODUCE A NOVEL EDUCATIONAL SOFTWARE METHODOLOGY

MONTSERRAT GUADALUPE MOLINA

Master’s Program in Computer Science

APPROVED:

Daniel Mejia, Ph.D., Chair

Oscar Mondragon, Ph.D.

Meagan Vaughan Kendall, Ph.D.

Stephen L. Crites, Jr., Ph.D.
Dean of the Graduate School
Dedication

I dedicate this thesis to my parents, my siblings, and my boyfriend, whose support and love has guided me throughout this entire process. Additionally, this work is dedicated to my professor and committee chair, Dr. Daniel Mejia, who has provided me with invaluable guidance and support throughout my entire journey in graduate school.
LEVERAGING AGILE SOFTWARE METHODOLOGIES WITHIN SOFTWARE DEVELOPMENT TO INTRODUCE A NOVEL EDUCATIONAL SOFTWARE METHODOLOGY

by

MONTSERRAT GUADALUPE MOLINA, BSCS

THESIS

Presented to the Faculty of the Graduate School of
The University of Texas at El Paso
in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE

Department of Computer Science
THE UNIVERSITY OF TEXAS AT EL PASO
December 2023
Acknowledgements

Words cannot express my gratitude to my professor and chair of my committee, Dr. Daniel Mejia. This endeavor would not have been possible without his dedication, patience, and guidance. I am deeply grateful for his continued support throughout my research and journey in graduate school.

I would also like to extend my sincere gratitude to Dr. Meagan Vaughan Kendall and Dr. Oscar Mondragon, my defense committee, who generously provided their knowledge and expertise. None of this would have been possible without their time, dedication, and continued assistance throughout the entire process, which provided invaluable knowledge, help, and support.

I am also very grateful to my family, especially my parents, siblings, and boyfriend. Their patience and emotional support kept my spirits and motivation high during every challenge I encountered throughout this entire process. I would not be here if it was not for them.

Last, I would like to thank everyone who participated in my research study. I am grateful for the time, effort, and willingness of the participants for they provided valuable insights towards my research. Their contributions were instrumental in the process and findings of this research.
Abstract

Agile Software Development has been growing increasingly popular in the software engineering industry as a way to produce working software in a quick and people-centered manner. Agile methodologies require practitioners to have strong technical and non-technical skills, such as teamwork, project management, and communication skills. Students graduating from the software engineering discipline have been found to be lacking in these areas, leading to many difficulties faced by recent graduates as they begin their professional careers. Given that Agile Software Development is the most popular software development lifecycle currently used by practitioners in industry, it is important to expose students to Agile practices prior to graduating from university.

In order to understand students’ perspectives pertaining to software engineering education, surveys were presented to computer science students aimed at assessing and understanding their experiences, familiarity, and motivation within their computer science education, particularly as it related to their knowledge of software development lifecycles. The data was used in the creation of a novel educational software methodology. This methodology leverages the Agile principles, the project-based learning principles, and engineering educational principles in order to create a synergistic learning environment with a goal of enhancing student learning.

The aim of this new methodology is to simulate a real-world problem utilizing an Agile-Waterfall hybrid approach in order to guide the students in their transition towards an Agile mindset. Unlike other educational methodologies and models that have been implemented, this new model combines the strengths of Waterfall and Agile to ensure a structured approach.
towards learning Agile methodologies, establishing mutual support between the students and the professor.
# Table of Contents

Dedication ... iii

Acknowledgements ................................................................. v

Abstract ........................................................................................ vi

Table of Contents ........................................................................... viii

List of Tables .................................................................................. xii

List of Figures .................................................................................. xiii

Chapter 1: Introduction .................................................................. 1
  1. Motivation ................................................................................. 3
     i. Goals and Research Questions ................................................. 3

Chapter 2: Learning Processes ...................................................... 5
  1. Introduction ................................................................................. 5
  2. Active Learning and Passive Learning ........................................ 6
  3. The DOLA Framework .............................................................. 7
     i. Active Mode ........................................................................... 8
     ii. Constructive Mode ................................................................. 8
     iii. Interactive Mode ................................................................. 10
  4. Project-Based Learning ............................................................. 11

Chapter 3: Software Development Lifecycles ............................... 13
  1. Introduction ................................................................................. 13
  2. Waterfall .................................................................................... 13
     i. Step 1: Program Design ......................................................... 16
     ii. Step 2: Document the Design ............................................... 17
     iii. Step 3: Do it Twice ............................................................... 18
     iv. Step 4: Plan, Control, and Monitor Testing ........................... 18
     vi: Evolution of the Waterfall Model ........................................... 19
        a. The “Sashimi Model” ......................................................... 20
        b. Waterfall with Subprojects ................................................. 20
        c. Waterfall with Risk Reduction ........................................... 21
vii: Strengths of the Waterfall Model ................................................................. 22
viii: Weaknesses of the Waterfall Model............................................................. 22
3. Agile Software Development .............................................................................. 23
   i. Agile Methodologies....................................................................................... 25
      a. Scrum ......................................................................................................... 26
      b. Extreme Programming .............................................................................. 29
      c. Joint Application Development............................................................... 32
      d. Lean Development .................................................................................. 34
   ii. Self-Organizing Teams in Agile ................................................................. 35
   iii: Strengths of Agile Software Development .............................................. 36
   iv. Weaknesses of Agile Software Development ........................................... 37
4. Agile and Waterfall in Industry ....................................................................... 37

Chapter 4: Challenges in Software Engineering Education .................................. 40
1. Introduction ...................................................................................................... 40
2. Areas of Identified Knowledge Deficiencies ................................................ 41
3. Agile in Software Engineering Education ...................................................... 42
   i. The Pyramid of Agile Competences ......................................................... 43
   ii. Teaching Scrum in Software Engineering Courses ............................... 47
      a. A Scrum Software Engineering Course ............................................... 48
      b. An Agile Software Development Course with Scrum ....................... 50
      c. A Hybrid Model with Problem-Based Learning and Scrum in a Software
         Engineering Course .............................................................................. 53
   iii. eduScrum ................................................................................................. 55
      a. Transparency ............................................................................................ 56
      b. Review ....................................................................................................... 56
      c. Adaptation ................................................................................................. 57
      d. The eduScrum Team ............................................................................... 57
      e. The Professor ........................................................................................... 58
      f. Ceremonies ............................................................................................... 59
      g. An Example of eduScrum in the Classroom .......................................... 60

Chapter 5: Methodology ....................................................................................... 66
1. Survey Creation ............................................................................................... 66
3. Discussion.........................................................................................................................137
4. Limitations ......................................................................................................................138
Future Work .......................................................................................................................140
References ..........................................................................................................................141
Vita 146
List of Tables

Table 1: Overview of Semester Plan [16] ........................................................................................................ 45
Table 2: Breakdown of the Course [12] ........................................................................................................ 51
Table 3: Certain Survey Items with their Average Scores [70] ................................................................... 65
Table 4: Group Statistics for each Classification Group and Their Familiarity with Agile and Waterfall ......................................................................................................................... 81
Table 5: Independent Samples t-Tests for Student Familiarity with Agile and Waterfall Based on Classification ........................................................................................................................................ 81
Table 6: Comparison of Means Between the two Classification Groups of Students for Which Class Assigns More Group Work .......................................................................................................................... 86
Table 7: Group Statistics for the two Classification Groups of Students for Which Class Assigns More Group Work .......................................................................................................................................... 86
Table 8: Independent Samples t-Tests for the two Classification Groups of Students for Which Class Assigns More Group Work .......................................................................................................................... 86
Table 9: Group Statistics for Student Familiarity with Self-Organized Teams Based on Current Enrollment in Agile Class ........................................................................................................................................ 88
Table 10: Independent Samples t-Tests for Student Familiarity with Self-Organized Teams Based on Current Enrollment in Agile Class ........................................................................................................................................ 88
Table 11: Descriptive Statistics for Student Experience with Self-Organized Teams .................................. 90
Table 12: Labels Assigned to each Statement of the Team Effectiveness Questionnaire .............................. 94
Table 13: Groups Statistics for Each Student Group for the Team Effectiveness Questionnaires .................. 95
Table 14: Independent Samples Welch’s t-Tests for the Team Effectiveness Questionnaires .... 96
Table 15: Descriptive Statistics for the Knowledge Areas ........................................................................... 98
Table 16: Examples of Each Learning Model ............................................................................................... 100
Table 17: Descriptive Statistics for Learning Activities that Help Students Build their Knowledge .............. 101
Table 18: Standardized Regression Weights (Factor Loadings) of the Confirmatory Factor Analysis for the MUSIC Model .................................................................................................................. 103
Table 19: Descriptive Statistics for the Number of Classes that have Used Agile and Waterfall Courses .................................................................................................................................................. 105
Table 20: Descriptive Statistics for Student Experiences in Agile Courses and Waterfall Courses .......................................................................................................................................................... 105
Table 21: Paired Samples Statistics for Students Who Have Experienced Agile and Waterfall Courses .................................................................................................................................................. 105
Table 22: Paired Samples t-Test for Students Who Have Experienced Agile and Waterfall Courses .................................................................................................................................................. 106
Table 23: Mappings of Agile Principles to Educational Objectives ............................................................. 109
Table 24: Mappings of Agile Core Values to PBL Principles ........................................................................ 113
List of Figures

Fig. 1: Relationship Between the Different Learning Modes from DOLA [26] .............................. 7
Fig. 2: Examples of Active Learning & Constructive Learning ...................................................... 9
Fig. 3: Examples of Constructive Learning & Interactive Learning .............................................. 10
Fig. 4: Implementation Steps Needed for a Small Program [35] .................................................. 14
Fig. 5: The Waterfall Model [35] ................................................................................................. 15
Fig. 6: Iterative Approach to Waterfall [35] ............................................................................... 15
Fig. 7: Example of Issues Arising with Iterative Waterfall [35] .................................................. 16
Fig. 8: Waterfall with Documentation [35] ................................................................................... 17
Fig. 9: The Sashimi Model [37] .................................................................................................... 20
Fig. 10: Waterfall with Subprojects [37] ..................................................................................... 21
Fig. 11: Overall Agile Methodology [34] ...................................................................................... 25
Fig. 12: Scrum Model [46] ........................................................................................................... 26
Fig. 13: Pull Model for XP Development ...................................................................................... 30
Fig. 14: JAD Process [37] ............................................................................................................ 34
Fig. 15: The 6 Roles for Facilitating Self-Organized Agile Teams [53] ........................................ 36
Fig. 16: Top 6 Most Identified Knowledge Deficiencies .............................................................. 41
Fig. 17: Pyramid of Agile Competences [16], [19] ..................................................................... 43
Fig. 18: Course Evaluation Results from [16] ............................................................................. 46
Fig. 19: Staff Roles of SE1 Course [66] ....................................................................................... 49
Fig. 20: Proposed Hybrid Problem Based Learning/Scrum Model [67] ..................................... 54
Fig. 21: eduScrum Cycle as Adapted From [68] ......................................................................... 56
Fig. 22: Roles Adapted from eduScrum by [70] .......................................................................... 61
Fig. 23: Class Setup Utilizing eduScrum for [70] ...................................................................... 63
Fig. 24: Course Structure for Summer Term 2019 ..................................................................... 64
Fig. 25: Student Classifications from the First Survey ................................................................. 76
Fig. 26: Student Familiarity with SDLcs ..................................................................................... 77
Fig. 27: Percentages of SDLcs Used Inside of School and Outside of School ......................... 78
Fig. 28: Student Familiarity with Waterfall and Agile ................................................................. 79
Fig. 29: Box Plots Denoting the Relationship Between a Student’s Classification and Their Familiarity with Agile ............................................................... 80
Fig. 30: Box Plots Denoting the Relationship Between a Student’s Classification and Their Familiarity with Waterfall .............................................................. 80
Fig. 31: Areas/Skills Students Believe Themselves to Struggle the Most In ................................. 82
Fig. 32: Student Classifications from Discovery Survey 2 .......................................................... 83
Fig. 33: Relationship Map Between a Student’s Classification and their Experience Working in Teams or Independently on Projects .................................................. 84
Fig. 34: Student Responses of CS Classes that have Assigned More Group Work/Projects ... 85
Fig. 35: Student Familiarity with Self-Organized Teams ............................................................ 87
Fig. 36: Relationship Map for the Relationship Between Student Current Enrollment in Agile Class and Familiarity with Self-Organized Teams ........................................ 89
Fig. 37: Student Experiences Working on Self-Organized Teams ............................................. 90
Fig. 38: Student Classifications from Teamwork, Learning Processes, and Lifecycles Survey .. 91
Fig. 39: Means of How Well Students Believe the Agile Course Helped Them Build the Knowledge Areas ................................................................. 98
Fig. 42: Means of How Helpful Student Find Certain Activities Towards their Learning ....... 100
Fig. 43: IBM AMOS Model of MUSIC Inventory .................................................................. 102
Fig. 44: Results from the MUSIC Inventory ..................................................................... 104
Fig. 45: PBL Principles [31] .......................................................................................... 112
Fig. 46: Preliminary Steps ............................................................................................... 121
Fig. 47: Structured Sprint ................................................................................................. 125
Fig. 48: Sprint Cycles ...................................................................................................... 129
Fig. 49: Final Sprint ......................................................................................................... 132
Fig. 50: AgileFlow Model ............................................................................................... 133
Chapter 1: Introduction

Software engineering is a field met with constant changes and challenges, requiring practitioners to have strong technical and non-technical skills. With a growing interest and use of Agile software methodologies in industry [1], [2], graduates should also have the skill set required to leverage these methodologies. Unfortunately, according to multiple studies, there is a gap between the graduates that academia is producing and the expectations that industry has for these graduates pertaining to their technical and non-technical skills [3], [4], [5], [6]. As such, many graduates face difficulties when they begin their professional careers given that what is learned in academia may be misaligned with the skills expected of them in industry [4].

It is important to understand the core skills of the software engineering field as compared to those from the graduates that academia is producing. For example, [7] conducted a study in which they identified several areas of knowledge deficiencies in computer science graduating students, which included communication, project management, and teamwork, among others. For industry, it is crucial to hire software engineering graduates that have been properly prepared academically in the discipline. This way, companies spend less time incorporating these recent graduates into the workplace [4].

One such problem that has been studied is that institutions of higher education have historically used lecture-based teaching approaches in their classrooms [6], [8], [9], [10], [11]. Consequently, this has led to many issues regarding student success, such as: students struggling to fully comprehend all of the material [10], low student engagement with the course content [11], a lack of non-technical development [11], and an increased likelihood of course failure [8]. In addition, Agile Software Development – which requires active learning practices [12] – is beginning to gain traction in academia [13], [14], [15], [16]. Agile Software Development not only
requires strong communication, teamwork, and project management skills, but it also requires a change in mindset of the practitioners [17]. Unfortunately, these were amongst the top areas of identified knowledge deficiencies in computer science students [7].

Communication is an important skill for any engineering discipline. Effective software engineers possess strong communication and listening skills with customers and stakeholders. Employers have also expressed dissatisfaction with the project management skills of recent graduates, citing that there was a large gap between what recent graduates knew for their current job versus what they needed to know [7]. Teamwork, another essential skill that software engineers should possess, is a common practice in industry, where individuals should be skilled in working within cross-disciplinary teams [7]. Unfortunately, computational thinking is usually considered an individual skill, and it is taught and implemented as such in academia [13]. Most of the current academic curricula focuses on individual work, and when group work is involved, teams are generally small, with 2 or 3 individuals [7].

As industry continues to adopt Agile Software Development methodologies and practices [1], [14], [15] it is important for students to acquire the skills of communication, project management, and teamwork. Agile methodologies are centered on teams [18] and require high levels of collaborative, communicative, and project management skills. Depending on the Agile methodology, some of these practices can include daily stand-ups, retrospectives, and pair programming, among others [19]. As such, several studies have suggested integrating Agile Software Development practices into the software engineering academic curricula to expose students to teamworking skills [1], [13], [14], [20], [21]. In addition, it is imperative that students learn Agile methodologies within the classroom prior to being exposed to them in industry. Since
Agile requires active learning methods over passive learning, one such approach to introducing Agile into the classroom would be by using project-based learning (PBL) as the form of instruction.

1. Motivation

In parallel with the increasing demand and use of Agile software methodologies in industry, many universities have started to explore ways of introducing Agile Software Development into their classrooms [16], [12]. Unfortunately, there are still many gaps and unknowns that need to be studied in order to understand how to best teach Agile methodologies to software engineering students. It is well understood that each student learns differently, and there are many pedagogical approaches that can be leveraged to expose students to Agile.

Different studies have suggested introducing Agile software methodologies into the classroom by adopting active learning techniques such as constructive learning [21], [22] and project-based learning [13], [12], [23] which derives from the constructive learning pedagogy [24]. This is because “as the name calls for, Agile should be taught in an agile way” [12]. For example, teaching characteristics such as self-directed learning, project-based learning, problem solving, teamwork, and leadership are similar to the Agile principles [12]. Constructive learning emphasizes the construction of knowledge via observation and experience, as opposed to the direct transmission of knowledge from teacher to student [21]. Therefore, project-based learning – which offers students the experience of working collaboratively on a project – is an agile way to teach Agile Software Development.

i. Goals and Research Questions

The overarching goal of this work is to propose a novel educational software methodology that incorporates the principles of Agile Software Development. This goal is decomposed into the following objectives: (a) understand the gap within the software engineering discipline between
academia and industry; (b) understand how leveraging Agile Software Development methodologies can improve the different areas of identified knowledge deficiencies of students within the discipline, and (c) analyze the experiences of computer science students within software engineering courses at an institution of higher education. Based on this goal and the objectives, the following research questions are raised:

- **RQ1**: What are the gaps between industry and academia in the software engineering field with respect to software development practices?
- **RQ2**: How can Agile Software Development practices be leveraged to create a synergistic learning environment that enhances student learning within software engineering courses that use project-based learning?
- **RQ3**: How can utilizing the strengths of existing software development lifecycles aid in the creation of a novel educational software methodology that incorporates the principles of Agile Software Development?
  
  i. How are different Agile methodologies currently leveraged in software engineering education?
Chapter 2: Learning Processes

This chapter will focus on defining the different learning processes that exist and are recognized in academia, as well as providing examples of each. In addition, this chapter will also give a brief history of how learning processes have traditionally been used in the classroom, explaining the shift from passive learning to active learning. This chapter will also introduce the Differentiated Overt Learning Activities framework, which breaks down the definition of active learning into 3 different modes.

1. Introduction

The learning process is a unique task for each student, given that every student has a different way of constructing their own knowledge [12], [22]. In engineering disciplines, it is important to adequately prepare students by stimulating critical thinking within the learning process [25]. It is suggested by [25] to adopt educational practices that will encourage the mobilization of knowledge, new skills, attitudes, and new values to meet the ever-evolving demands of industry. This is especially true for the practice of developing software projects, where students need to have a solid foundation of technical and non-technical skills.

The instructor’s teaching process should aim to establish a synergistic relationship between the instructor and the student to offer support and feedback [12]. Traditionally, students have played the role of the “absorbers of knowledge” and the instructors that of the “holders of knowledge”, whereby the teacher directly transmits their knowledge onto the student, who absorbs that knowledge [12]. Modern understanding of this learning/teaching process suggests that instructors should play the role of “facilitator” in accompanying students to construct their own knowledge [12]. Some examples of these learning processes include aspects of self-directed
learning, project-based learning, problem solving, teamwork, communication skills, and leaderships skills that instructors can incorporate into their teaching processes [12].

Educators within the engineering discipline have advocated for adopting the use of active learning over passive learning techniques [9]. As educators move towards adopting active learning techniques over passive learning ones in the classroom, it is important to understand the different learning methods that fall under the active learning process and the potential benefits and challenges that come with each.

2. ACTIVE LEARNING AND PASSIVE LEARNING

Ever since the establishment of universities in Western Europe over 900 years ago, lecturing has remained the predominant model for instructors to teach their students [8], [10]. Recent learning theories have emphasized the need for students to construct their own knowledge, challenging the traditional lecture-based approach [8]. Additionally, modern learning theory suggests that this “teaching by telling” method is not the most effective way of getting students to thoroughly comprehend all of the material [10]. This type of learning is known as passive learning, formally defined by [9] as: “Passive learning activities are those methods in which learners are not visibly doing anything, other than watching, listening, or reading. In these methods, it is not possible to know whether the learner is engaged with the activity or not.” Furthermore, passive learning encompasses the traditional classroom model of lecturing via direct instruction by the professor [10].

Active learning encourages students to take an active role in their learning process by engaging with instructors and other students, posing questions, and by motivating students to develop their own comprehension of the material [10]. Active learning is formally defined by [9] as: “Active learning activities involve some form of physical action. The students do something
while learning, although they do not need to create anything additional to the learning material.” The active learning approach is student-oriented, and it emphasizes the concept of knowledge retention by an active construction of meaning through educational experiences [10]. It is highly collaborative and can be very effective for improving the knowledge retention, communication skills, and self-directed learning skills of a student [10].

Although several studies have shown that active learning methods are often superior to passive learning ones [8], [26], [27], there is a discrepancy on the full meaning of what active learning encompasses. From taking notes to working collaboratively on a team, active learning can take many forms. In addition, the definitions and terminologies for active learning methods can vary across various disciplines [26].

3. THE DOLA FRAMEWORK

In order to address the issue of a lack of a consolidated definition for active learning, the Differentiated Overt Learning Activities (DOLA) framework was proposed by [26]. This framework divides active learning into three modes: active, constructive, or interactive. Figure 1 provides a visual representation of the relationship between the different learning methods.

![Fig. 1: Relationship Between the Different Learning Modes from DOLA [26]](image)

These are individually defined by what activities students overtly display during the learning process. A hypothesis postulated by [26] and [27] is known as the ICAP hypothesis, where each letter stands for, in order: Interactive, Constructive, Active, and Passive, in terms of the
different modes of learning. The ICAP hypothesis makes the claim that “student learning is more effective in interactive activities than constructive activities, which are more effective than active activities, which are more effective than passive activities”, summarized by (I>C>A>P) [26]. This hypothesis was studied and found to be supported by a research experiment conducted by [26]. The following subsections define the 3 different modes of the DOLA Framework.

i. Active Mode

In the active mode, students activate their own knowledge by engaging in overt activities [26], [27]. Some examples of these active mode activities can include underlining/highlighting text, following along with the instructor as they write down the solution to a problem on the board by copying it on a notebook, or searching for specific information within a text [26]. All active activities involve the student doing something with the material being taught, rather than passively receiving the information [26]. Furthermore, several studies have found or suggested that students who engage in active activities learn better than students engaged in passive activities [8], [26], [27], [28].

ii. Constructive Mode

Educators and psychologists have suggested that learning is a constructive process [27]. As such, constructive learning has become a very influential educational philosophy in the modern century [24]. Formally defined, constructive learning is “a learning process that requires students to construct their own understanding through a series of activities” [24]. These activities can include creation of artifacts [24], drawing a concept map, comparing and contrasting cases, and making plans [27], among others.

Constructive learning leads to what is known as discovery learning, whereby a student learns not from being told the material (direct instruction), but by discovering the material on their
own (construction of knowledge via discovery) [27]. Furthermore, given that constructive learning requires students to be active in the process of constructing their knowledge, constructive learning subsumes active learning [26], [27]. The key distinction between the two types of learning is that while both require activities, constructive learning has students produce some additional output which may contain new ideas (related to the topic being taught) that help the students construct their knowledge [27].

Figure 2 illustrates an example differentiating between active and constructive learning. Underlining a text is classified as active learning. The activity of self-explanation (i.e., a student articulating on the presented material) can be constructive only if the self-explanation is meaningful and goes beyond what was presented (such as making a deeper connection or understanding of the material), otherwise it is simply active learning [26], [27]. As such, constructive learning activities rely on two characteristics: (1) students/learners produce some sort of output, and (2) the output produced is different but related to what was presented in the material [27].

![Diagram of Active Learning vs. Constructive Learning]

**Fig. 2: Examples of Active Learning & Constructive Learning**
iii. Interactive Mode

In the interactive learning mode, two or more students undertake “activities that develop knowledge and understanding extending beyond the materials being studied (similar to the constructive mode), but the interaction of the learners further enables them to build upon one another’s understanding” [26]. The key difference between constructive learning and interactive learning is that the latter consists of at least two students engaging in an activity together [26] (thus, constructing their knowledge together). Figure 3 summarizes the relationship between constructive learning and interactive learning. Some examples of interactive learning include study groups, discussions/debates, and interacting with the professor, an expert, or a computer agent to receive feedback [26].

Fig. 3: Examples of Constructive Learning & Interactive Learning

Not every group activity, however, can or should be automatically classified as interactive learning based solely on the premise that it involves more than one student. For example, a pair activity in which one student does not contribute means that there is a lack of interaction between the two, and it is not interactive learning [26]. Interactive learning comes with the challenges of teamwork, such as student participation, collaboration, and effort towards the activity.
4. **Project-Based Learning**

This section will define Project-Based Learning, a learning process that falls under active learning. In addition, an explanation of where Project-Based Learning fits into the DOLA framework is provided. Last, its use and efficiency within STEM education is also discussed.

Project-based learning (PBL) is a form of instruction which focuses on the student as the center of learning [24], [29]. In a project-based learning environment, students engage directly with the material by taking an active role in the learning process through prioritizing project work, which in turn allows students the freedom to collaborate with one another, learn from each other, and lead their own project work [29]. Project work, one of the central aspects of PBL, is a collaborative form of learning that requires the active participation and engagement of each group member to achieve a shared goal [29]. It emphasizes active learning over passive learning experiences [29].

Formally defined, PBL is “a learning model that requires students to solve a problem together in a particular group” [24]. This learning model is derived from the constructive learning model, and it is based on the presumption that in order to construct their knowledge, students need opportunities to solve real-world problems, ask and refine questions, design and conduct experiments, gather data to analyze, and draw their own conclusions [29]. Furthermore, given the collaborative nature of most project works (specifically when a project requires two or more members working on a team), PBL can also exercise interactive learning.

PBL has been found to be effective specifically within STEM education, helping students to develop their critical thinking skills, scientific literacy skills, increase their involvement and engagement in the learning process, and has a significant effect on the creativity skills of students [24]. In software engineering education, PBL is one of the most successful student-centered
teaching methods used [30], [31], given that it allows students to be involved in the investigation of realistic problems, learn by working on projects, and discover solutions throughout their project progress [30].
Chapter 3: Software Development Lifecycles

This chapter focuses on defining two different software development lifecycles: Waterfall and Agile Software Development. A brief history is provided for each model along with different examples and uses, followed by a subsection describing the strengths and weaknesses of each model. Last, there is a section that compares the use of each model in industry. This research focuses on these two lifecycles is because Waterfall was the very first software development lifecycles to emerge; as such, it paved the way for the numerous models that are used today for software development. Waterfall, however, had many pitfalls, such that many different lifecycles were introduced into the world of software development in order to attempt to solve the issues that came with Waterfall. Agile Software Development, one of the most recent software development lifecycles (and most widely used [2]), is a direct response to the ever-evolving needs of industry.

1. INTRODUCTION

Software development is an ever-growing field with rapidly changing requirements and technologies. In order to meet the increasing demands of productivity of software, many different software development lifecycles were developed [32]. A software development lifecycle (SDLC) is defined by [33] as “a conceptual framework or process that considers the structure of the stages involved in the development of an application from its initial feasibility study through to its deployment in the field and maintenance.” There are many different SDLCs in existence, generally falling under three categories: linear, iterative, and a combination of linear and iterative [33]. This work focuses on one linear SDLC (Waterfall) and one iterative SDLC (Agile).

2. WATERFALL

The Waterfall Model was introduced in 1970 by Winston Royce [34]. Formally defined, the “Waterfall model is a sequential project management methodology where one phase finishes
before the next phase” [34]. When first introducing the Waterfall model, Royce argued that there are essentially 2 common steps involved in software development, regardless of the size or complexity: an analysis step followed by a coding step [35]. This very simple concept, he argued, (see Figure 4) is all that is required if the work needed is sufficiently small and if the final product is intended to be operated by those who built it [35].

![Diagram of Analysis and Coding Steps](image)

**Fig. 4: Implementation Steps Needed for a Small Program [35]**

Royce further stated that for larger software projects, this very simple design was doomed to failure, as many additional development steps are required [35]. He then came up with a more detailed, “grandiose approach to software development” [35] (illustrated in Figure 5). This model – which still incorporates the analysis and coding steps – consists of additional steps that must each be planned and staffed differently for optimal use of program resources [35]. Although Royce himself did not name this model “Waterfall”, Figure 5 illustrates what is known today as the Waterfall Model.
Royce further proposed an iterative relationship between each successive step of the model, such that “as each step progresses and the design is further detailed, there is an iteration with the preceding and succeeding steps, but rarely with the more remote steps in the sequence” [35]. This model is illustrated in Figure 6. The idea behind this model, as stated by Royce, is that as the design of the product proceeds, the change process is scoped down to manageable limits [35].
This model, however, “invites failure” and “is risky”, according to Royce [35]. For example, the testing stage of the model is the point of the development cycle in which timing, storage, input/output transfers, etc., are actually experienced [35]. The issue that arises from this is the following: if these fail to satisfy the constraints and requirements of the product, then a major redesign is required (see Figure 7) [35]. This is because a simple “redo of some isolated code” will not fix these kinds of problems; rather, the required design changes can affect even the software requirements upon which the design was based on [35]. As such, “the development process has returned to the origin and one can expect up to a 100-percent overrun in schedule and/or costs” [35].

**Fig. 7: Example of Issues Arising with Iterative Waterfall [35]**

In order to eliminate some of these developmental risks, Royce proposed 5 additional features that accompany his model [35] which are described in the following sections.

**i. Step 1: Program Design**

Royce introduced a preliminary program design phase between the software requirements and analysis phases [35]. To implement this phase, Royce proposed that the design process must
begin with program designers, not analysts nor programmers, and the sole focus should be on designing, defining, and allocating the data processing modes, even at the risk of being incorrect [35]. This can include designing/defining the database, allocating the functions, defining interfaces, and describing input and output processes, among other procedures [35]. Designers should then consolidate all of this information into an overview document that is easy to understand, descriptive, and up to date [35].

ii. Step 2: Document the Design

Royce’s direct answer to the question of how much documentation should be written is “quite a lot” [35]. He strictly enforced the need for extensive documentation, claiming that the “management of software is simply impossible without a very high degree of documentation” [35]. Figure 8 illustrates Royce’s proposed documentation plan that should accompany his development model, with a total of 6 documents produced by the time the final product is produced [35].

![Waterfall with Documentation](image)
iii. Step 3: Do it Twice

The third step is to ensure that the developed program is checked twice, such that the final version that is delivered to the customer is not the first version, but the second version [35]. Royce proposed to redo the entire process in “miniature”, meaning keeping it within a small time scale with respect to the overall effort [35]. Royce referred to this as running a simulation of the program by retracing (from beginning to end) all of the steps [35]. This would allow questions regarding timing, storage, or any other factors relevant to the program to be studied with more precision, as it allows for more experimental tests to be carried out [35].

iv. Step 4: Plan, Control, and Monitor Testing

The testing phase in Royce’s proposed model occurs in the latest stage of development, at which point designing alternatives is less feasible [35]. This is why Steps 1 – 3 described above for designing the program are aimed at uncovering and solving problems before the test phase takes place [35]. Royce provides some recommendations for testing, including [35]:

- Having test specialists who did not contribute to the original design of the product test the product. The documentation written for testing should provide good context for the specialist to test the software product. Royce argued that the specialist would do “a better job of testing than the designer” [35].

- A second party – who also did not contribute to the original analysis or code – should perform a visual scan of the code.

- Every logic path of the program should be tested at least once. This includes running different test with controlled values of input. While it might seem impossible to achieve full testing for very large systems, Royce still recommends that “every logic path be subjected to at least one authentic check” [35].
• Once the smaller errors are uncovered and removed, a final “checkout” should be performed. This means that the software should be tested in completion prior to being delivered to the customer.

v. Step 5: Involve the Customer

The customer should be formally involved early in the development cycle to provide their insight and judgement along the way [35]. Royce proposes 3 reviews in which the customer should be involved along the process [35]. After the Preliminary Program Design phase and before the Analysis, a Preliminary Software Review (PSR) should take place [35]. Similarly, after the Program Design step but before the Coding stage, a Critical Software Review (CSR) should take place [35]. Finally, after Testing and before Operations, a Final Software Acceptance Review (FSAR) should be held [35]. Royce fails to provide any detailed information relevant to what should happen in these review sessions.

The following subsections will describe the evolution of the Waterfall model by providing different examples of variations of the Waterfall model that attempted to address the issues with Royce’s original model.

vi: Evolution of the Waterfall Model

Royce’s original model has since evolved from when he first proposed it in 1970 [36]. In fact, when Royce first crafted his model, he did so in the context of constraints of government contracting [36]. Steve McConnell, author of *Rapid Development: Taming Wild Software Schedules*, claimed that the weaknesses of the pure Waterfall model arise from the treatment of each stage in the process as “disjoint, sequential phases” [37]. He describes numerous attempts to modify the original Waterfall model, starting with Peter DeGrace’s “Sashimi Model” [37].
a. The “Sashimi Model”

Named after the Japanese practice of presenting raw fish with each slice overlapping each other, the Sashimi model adapts the Waterfall model to allow a stronger degree of overlap between each phase [37]. For example, a developer can be in the Preliminary Program Design phase and partway into the Analysis phase, while also still considering the Software Requirements phase incomplete [37]. An example of the Sashimi model is shown in Figure 9 [37].

Fig. 9: The Sashimi Model [37]

Another idea behind the Sashimi model was to allow for continuity with regards to documentation between phases, rather than keeping documentation separate from each other such as in traditional Waterfall [37]. This would in turn allow for less documentation [37]. However, by allowing overlap between phases, milestones in each step become more ambiguous, and tracking progress is also more challenging [37]. For small projects, pure Waterfall would be more efficient than Sashimi [37].

b. Waterfall with Subprojects

When considering rapid development, another challenge that arises from traditional Waterfall is the need for each phase to be fully complete before moving on to the successive step
McConnell pointed out the situation in which a system contains certain areas that have been implemented before, asking the question: “Why delay the implementation of the areas that are easy to design just because we’re waiting for the design of a difficult area?” [37]. One such modification to Waterfall includes adding logically independent subsystems to allow for separate “subprojects” to proceed at their own pace [37]. Figure 10 shows this modification. The main issue associated with this model is unforeseen interdependencies, which can hinder the development of the subprojects [37].

Fig. 10: Waterfall with Subprojects [37]

c. *Waterfall with Risk Reduction*

Another issue with the traditional Waterfall model is that in order to begin the design phases, you must fully understand and have completed the requirements phases [37]. Modifying the model to add a risk-reduction spiral (see Figure 11) at the very top would allow the developers to address this requirements risk [37]. This is because more emphasis is given to the requirements phase by doing practices such as: developing a user-interface prototype, conducting interviews with users, and other similar practices [37]. As such, the risk reduction phase would
allow developers to address the requirements and analysis phases during this risk reduction phase instead [37]. This idea of a risk reduction phase isn’t limited to just the requirements – it can be applied to any phase of risk of the project [37].

vii: Strengths of the Waterfall Model

The Waterfall model is a particularly strong model for projects in which the product definitions are stable [37]. This model is also known for providing the requirements stability – given the extensive documentation required – that many developers (and customers) crave [37]. In addition, Waterfall helps minimize planning overhead given that all the planning is done up front [37]. For complex projects with clear definitions, Waterfall is beneficial since it provides an orderly way to tackle complexity issues [37].

McConnel also claims that Waterfall “works especially well if you have a technically weak staff or an inexperienced staff because it provides the project with a structure that helps to minimize wasted effort” [37]. This is yet another advantage of Waterfall – it is an easy-to-understand development cycle [38]. It also helps reinforce good design habits by forcing developers to define requirements and product specifications before designing, and, similarly, to design the product before coding [38].

viii: Weaknesses of the Waterfall Model

Unfortunately, Waterfall is not flexible in the face of changing requirements [37], [39], [40]. If the requirements of a product are not well understood, defined, or are very likely to change, Waterfall becomes unsuitable for the project at hand [41]. Although Iterative Waterfall (see Figure 6) does allow the developer to go back and forth between successive phases, doing so is extremely difficult [37]. McConnell uses the example of a salmon attempting to swim upstream, as he claims: “you’re allowed to swim upstream, but the effort might kill you!” [37]. At the end of each phase,
multiple major events have already taken place to successfully mark that phase as completed, such as: reviews, documentation signoffs, etc., which means that if a flaw is later discovered in a downstream phase, it is quite challenging to go back “upstream” to fix any changes made “downstream” [37]. This is an extremely important pitfall of Waterfall given that many projects rarely follow the sequential approach [38] and in a lot of instances, users/customers “don’t know what they want” [37].

Another disadvantage of Waterfall is that working versions of the software product are not produced until the end of the development cycle, thus users/customers must be patient in receiving a working product and can only rely on written specifications of the software’s intended functionalities [38], [40]. Additionally, Waterfall’s strict adherence to numerous documentations can also hinder the progress of a rapid-development project, since attempting to retain flexibility in a project that contains certain unknowns requires constantly updating the documentation, creating “perceptions of slow development” [37]. Last, Waterfall is also a very costly development process [40], [41], [42].

As customer needs and demands increased, Waterfall was quickly outpaced by new SDLCs which attempted to modify the Waterfall model to make it iterative or incremental [32].

3. AGILE SOFTWARE DEVELOPMENT

With a focus on responding to changing requirements, prioritizing customer needs, and delivering working products to the customer in short intervals, eventually a new SDLC was created: Agile Software Development [32]. The Agile Software Development (ASD) model was created in 2001 with the publication of the Agile Manifesto [32]. ASD balances prioritizing customer needs and changing requirements while delivering incremental parts of the product to the customer in short intervals of time [32]. The Agile Manifesto consists of four core values [43]:
1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

From these four core values, the Agile Manifesto derives twelve guiding principles for the methodologies that fall under it [43]:

1. Collaboration between the business stakeholders and developers throughout the project.
2. Agile processes to support a consistent development pace.
3. Enable face-to-face interactions.
4. Regular reflections on how to become more effective.
5. Customer satisfaction through early continuous software delivery.
6. Accommodate changing requirements throughout the development process.
7. Frequent delivery of working software.
8. Support, trust and motivate the people involved
9. Working software is the primary measure of progress.
10. Attention to technical detail and design enhances agility.
11. Simplicity.
12. Self-organizing teams encourage great architectures, requirements, designs.

Beyond being an SDLC, effective Agile practice requires a particular attitude and way of thinking, commonly referred to as an agile mindset [17]. It is insufficient to simply apply Agile practices and methodologies [17], there is also a need for the team members to adopt changes in mindset [44]. As stated by [17], “the right way to agility should start with a proper agile mindset
instead of applying Agile methods directly”. As such, it is important that practitioners fully understand the principles and values of Agile before attempting to practice the methodologies [17].

The following subsection provides a few examples of some methodologies that fall under Agile, and the different practices that of each methodology.

i. Agile Methodologies

The ASD consists of multiple different models that fall under the term Agile [32]. Figure 11 provides a vast generalization of an agile methodology, with each methodology differing in their own ways. Figure 11 shows how each iteration consists of a requirements, implementation, and verification phase, followed by presenting parts of the product to the customer for review [34]. This way, any feedback provided by the customer can be addressed in the next iteration of these phases [34]. All methodologies that fall under Agile are consistent with the values and principles of the Agile Manifesto for software development [32], [34]. Some of these agile methodologies are Scrum, Extreme Programming, Joint Application Development, and Lean Development [33], which are described in the following subsections.

Fig. 11: Overall Agile Methodology [34]
a. Scrum

Scrum is a framework which uses small, pre-defined time frames - known as sprints - where progress is measured daily [22], [33]. Scrum is ideal for smaller projects since it places less emphasis on a process driven framework [33]. It utilizes an iterative, incremental process to develop the project (see Figure 13) [45]. The main goal of Scrum is to utilize sprint cycles (time-boxed for 2-4 weeks in length) to implement a small number of customer requirements.

Fig. 12: Scrum Model [46]

Figure 12 illustrates the Scrum Model, which includes the following different events/artifacts:

- **Sprint Planning**: this event initiates the sprint [47]. The team plans out the work to be performed during the sprint, resulting in a plan agreed upon by the team [47]. The Scrum Guides pose the following three questions that need to be addressed during a Sprint Planning: (1) “Why is this sprint valuable?” (2) “What can be done this sprint?” (3) “How will the chosen work get done?” [47]

- **Daily Scrum**: also commonly referred to as a daily stand-up meeting, which is a daily meeting requiring the presence of all team members to contextualize and report on their work status [22]. In this meeting (time-boxed to 15 minutes, but the duration can vary by team), progress towards the sprint goal is inspected, and the Sprint Backlog is
modified as necessary [47]. The Daily Scrum is commonly held at the same time and place every workday [47].

- **Sprint Review**: a meeting in which team members inspect the outcome of the current sprint that has just been finalized [22], [47]. In this meeting, the team also presents their results and progress to key stakeholders for review and discussion on what to do next [47].

- **Sprint Retrospective**: a meeting in which the team inspects how the sprint went in order to identify the most helpful changes to improve the effectiveness of future sprints [47]. Teams discuss the last sprint’s effectiveness in terms of each individual team member, interactions, processes, tools, and their Definition of Done [47]. In addition, the team discusses what worked, what did not work, and how problems that were encountered were (or were not) solved [47].

- **Product Backlog**: an ordered list of everything that is needed to compose the project which is never complete and is always evolving [22]. This list includes items of what is needed to improve the product [47]. A Product Backlog refinement can take place to break down items from the backlog into smaller, more precise items [47]. This is an ongoing activity in which the team works to refine details, order, and size [47].

- **Sprint Backlog**: a list of everything the team will be working on during the current sprint [22]. This list is made up of the set of Product Backlog items selected for the current sprint, for which the team creates a plan for delivering the Increment [47]. This backlog is a real-time picture of the work that developers are doing during the sprint, for which the Sprint Backlog is updated to reflect the team’s progress [47].

27
• Increment: a steppingstone towards the goal for the product [47]. A sprint is made up of multiple Increments, where the sum of Increments is presented at the Sprint Review [47]. In order for any work done to be considered part of an Increment, it must meet the Definition of Done [47]. All Increments must be thoroughly verified in order to ensure that all Increments work together [47]. A single Sprint may consist of multiple Increments being created, all of which are presented at the Sprint Review if they each meet the Definition of Done [47]. In some circumstances, an Increment may be delivered to stakeholders throughout the Sprint (before it is finished) [47].

• Definition of Done: “a formal description of the state of the Increment when it meets the quality measures required for the product” which creates transparency for everyone by providing a shared understanding of what work was done [47]. For example, if an item from the Product Backlog does not meet the Definition of Done, it can neither be presented at the Sprint Review, nor be released, thus it must be returned to the Product Backlog [47]. On the other hand, once an item from the Product Backlog meets the Definition of Done, an Increment is born [47]. All of the developers that are part of a Scrum Team are required to conform to the Definition of Done [47]. There are two ways in which a Definition of Done can be defined. The first way is through defining it as part of the standards of the organization, meaning that all Scrum Teams that are part of the organization must follow it as a minimum [47]. The second way is if it is not defined as an organizational standard, then each Scrum Team is responsible for creating their own Definition of Done that is appropriate for the product [47].

In addition to these events and artifacts commonly practiced, Scrum also has specific roles for the team members. Some of these are [22]:

28
- **Product Owner**: this team member is responsible for listing the most important requirements for the project backlog.

- **Scrum Master**: makes sure that the team is working within the Scrum values and is using the framework correctly.

- **Developers**: members of the team who primarily perform the task of developing the project through self-organization and management while working collaboratively. A development team must consist of 3 to 9 members.

**b. Extreme Programming**

Extreme Programming (XP) prioritizes the development of the project via increments without having an explicit initial design stage [33]. As stated by Kent Beck, the creator of XP, this methodology is “a philosophy of software development based on the values of communication, feedback, simplicity, courage, and respect” [48]. The main aim in XP is to enable successful software development considering software requirements that are unclear or constantly changing [45]. It utilizes short development cycles to gather informal requirements from customers, organize teams of pair programmers, develop simple designs, refactor, and perform continuous testing, therefore yielding frequent releases [45].

XP is designed for managing projects where the requirements are unstable and unpredictable [45]. It improves quality and productivity by introducing checkpoints where new customer requirements can be easily adopted [39]. This methodology is highly collaborative, encouraging managers, developers, testers, stakeholders, and others to interact regularly [39]. Thus, it is good for providing faster development and frequent releases of the product [39].

Further, in XP, Beck emphasized the need for “stories” instead of “requirements” [48]. These stories are meant to be units of customer-visible functionality, for example, “Provide a two-
click way for users to dial frequently used numbers” [48]. Once a story is written, developers should estimate the development efforts necessary to implement it [48]. Estimation, Beck argues, “is a key difference between stories and other requirements practices” [48]. This is because clear estimations can give the business and technical perspectives a chance to interact since the team knows the cost of the features [48]. Stories in XP have short names and short descriptions, and it is a common practice to write these on index cards which are put on a frequently passed wall [48].

XP is based on what Bent called a “pull model of development” (see Figure 13) [48]. This works as follows: first, stories are specified in detail; then, tests are pulled from these specifications, and the programming interface design is then pulled from the tests to match their needs, such that the code is written to match the tests and interface design [48]. Thus, each phase pulls from each other, allowing for the design to be refined to match the needs of the code as it is written [48].

![Fig. 13: Pull Model for XP Development](image)

Some of the most common and important practices in XP include [22]:

- The planning game, in which the customer decides which story has the highest priority for the next iteration.
- Small releases since the software is built incrementally through small deliveries of functionalities.
• Testing constantly at every iteration.

• Pair programming – all code must be written in pairs where two people share a single computer.

• On-site customer: the customer must always be present.

Some of the roles included in XP are the following:

• Programmer: at the heart of XP, programmers spend their time working with the code [48]. In addition to this, the first value of an XP programmer is communicating with the other team members and the customer [48]. Programmers must have good pair programming skills, but they must also possess strong communication skills [48]. As such, XP programmers should prioritize shared ownership of the system, rather than seeking individual ownership of their portion of the program [48].

• Customer: responsible for knowing what the programmers should program. Customers need to learn the skills of writing good stories in order to communicate these to the programmers [48]. Customers, however, will be aided by the XP team to learn how to write these stories [48]. In addition, the hardest responsibility that befalls the customer is that of making decisions [48]. In XP, the customer must be confident in decision-making, such as prioritizing some stories over others [48]. The customer must also write functional tests for the system, with the XP team working closely with the customer to create these tests [48].

• Tester: this role is focused more on the customer than it is on the programmers [48]. The tester is responsible for helping the customer write functional tests [48]. This individual might not be separate from any other role – rather, they are the team member designated to run tests regularly [48].
• Tracker: “the conscience of the team” [48]. This team member is responsible for keeping an eye on the big picture and notice how reality conforms to the estimates of the stories [48]. In addition, they are responsible for monitoring the progress of other team members in order to tell them if they should follow through with their current course or if they need to change something [48]. Last, this member is also the team historian, meaning that they keep a log of all functional test results, defects reported, and all other appropriate logs [48].

• Coach: this team member is responsible for the entire process [48]. They should have a deeper understanding of XP than the other team members in order to avoid anyone from deviating from the team’s process [48]. They serve as a guide for the team, directing the team in design, refactoring, and testing [48]. The responsibilities of the XP Coach diminish as the team matures, since everyone should be responsible for the process [48].

c. Joint Application Development

Joint Application Development (JAD) was developed due to a realization that more customer involvement would lead to better systems [49]. Accordingly, JAD focuses on the collaborative development of the project with the customer or end user by involving them in the process as much as possible through workshops during the design and development phases [33]. These workshops are where a designated experienced session leader interviews managers and end users to define the scope and objectives of the project [50]. These workshops usually last between 3-5 days. Thus, it emphasizes strong end user participation in the eliciting of requirements and design documentation, enabling for strong communication to be established [50].

JAD is made up of two main phases: a Planning phase and a Design phase [37]. These phases deal with the system’s requirements at different levels, both focusing on the business-design problem rather than on the technical details [37]. During the Planning phase, the team works
on mapping out the broad capabilities of the product, with the main outcomes being the goals for
the product and preliminary schedule estimates [37]. In the Design phase the team elicits more
detailed requirements in order to create the user-level design of the product [37]. This phase uses
prototyping rather than focusing on the functional design of the product, with the main outcomes
of this phase including: a user-interface design, a database schema, and refined schedule/cost
estimates [37]. In each of these two phases, JAD iterates through three different subphases (see
Figure 14) [37], [49]:

1. Customization: a preparatory phase leading up to the workshop session. The session leader
and 1-2 other team members gather documents and conduct preliminary interviews.

2. Session/Workshop: the customers, team members, and facilitators come together for
extended meetings. There is a designated session leader to lead the elicitation of
requirements from the customers. This session could also make use of modeling, defining
objectives and scope, and clarifying what does and does not need to be included.
Participants also come up with timelines, assumptions, constraints, etc., and as the
development reaches later stages of the project, customers are asked to provide more
details.

3. Wrap-up: in this phase, team members consolidate all documentation provided during the
workshops into a single, coherent rigorous document. In some phases, a prototype might
also be constructed for use in the next cycle of workshops.

In order to transition between the Planning phase to the Design phase (see Figure 14), the
project must be approved. Therefore, after the Planning phase, the materials are reviewed both by
the team and the Executive Sponsor (the customer) [37]. At this point, a decision can be made as
to whether or not they will continue with further product development [37]. If it is approved, the
Design phase comes next [37]. Similarly, at the end of the Design phase, a decision must be made in order to approve continuation of the product [37]. Once again, the materials are reviewed by the customer and the team [37]. If approved, the implementation step is next, ending the JAD-design phase [37].

![JAD Process Diagram](image)

**Fig. 14: JAD Process [37]**

**d. Lean Development**

The pioneer of Lean Development (LD) is Toyota, who came up with what was first known as the Toyota Production System, nowadays known all over the world as Lean Development [51]. The mastermind behind LD, Taiichi Ohno, developed the Toyota Production System with the fundamental lean principle: eliminate waste [52]. Ohno defined waste as anything that does not create value for the customer [52]. Thus, in Lean, when a development project is started, “the goal is to complete it as rapidly as possible” [52].

LD adopts an ‘80% today is better than 100% tomorrow’ paradigm by attempting to deliver a project earlier at the cost of minimal functionality [33]. The main idea in Lean is to give the customer what they want, when they want it, and at the desired amount requested [51]. It is an iterative methodology with a focus on reducing waste and optimizing the entire process to achieve
the maximum gain possible [45], trading designs, prototypes, and intermediate steps in the
development process for a working product delivered to the customers [52].

ii. Self-Organizing Teams in Agile

Self-organizing teams are at the heart of the ASD and are one of the 12 principles in the
Agile Manifesto [53]. In such a team, each member contributes with their knowledge, abilities,
technical, and non-technical skills [13]. According to [54], self-organizing teams are teams that:
(1) typically perform highly related or interdependent jobs, (2) are identified as a social unit in an
organization, and (3) are given significant authority and responsibility for their work. Other
important aspects of these types of teams are that they manage their own work and organize
themselves regarding details of the tasks at hand, greatly influencing team effectiveness [53].

Instead of having a single team leader, leadership within self-organizing teams in Agile is
distributed amongst team members, allowing for each member to have some control over the
project as opposed to having centralized management [53]. Some examples of this include the
Scrum Master within the Scrum methodology and the XP Coach in Extreme Programming, both
of which are facilitators rather than leaders within their teams [53]. Given that each team member
manages their own work, self-organized teams must have common focus, mutual trust, and respect
between team members [53].

A key question that comes up this structure of teams is: “How do self-organizing Agile
teams organize themselves?” [53]. This is a crucial question given that ASD fails to provide
information on how self-organized teams should, in fact, organize themselves. ASD does not
explicitly define roles for team members in self-organized teams. As such, a study conducted by
[53] found that typically, self-organized teams adopt 6 roles, described in Figure 15. These roles
help facilitate the self-organization of the team, and a single team member could play one or more
of these roles [53]. In addition, the study stated that not all of these roles are always found within self-organized teams, and these roles were sometimes played by developers and/or business analysts [53].

![Fig. 15: The 6 Roles for Facilitating Self-Organized Agile Teams [53]](image)

### iii: Strengths of Agile Software Development

In the first 10 years of its creation, Agile faced a lot of criticism, cited as “undisciplined” by critics [55]. However, ASD eventually grew in popularity due to its lightweight nature of prioritizing working software over process controls and documentation, which can lead to a lot of overhead and delay [55]. The ability to make progress visible to stakeholders while also allowing them to make changes throughout the process is a key benefit of Agile [55]. Additionally, Agile is not limited to one particular method, given the range of different methodologies that fall under Agile [55].

ASD is a flexible lifecycle for projects with rapidly changing requirements or for projects in which there is a lot of competition in the product field, thus requiring rapid development [56]. A key benefit of ASD is the emphasis on constant communication and active involvement with
Given its incremental and iterative nature, Agile is better suited for projects in which customers require early or partial solutions [38].

Agile has also been helping companies – such as Salesforce – increase productivity while decreasing costs [57]. In 2006, Salesforce transitioned to Scrum, releasing 94% more features, delivering 38% more features per developer, and delivering over 500% more value to its customers during the first year of making the switch [57]. In the next two years, Salesforce more than doubled their revenue [57].

**iv. Weaknesses of Agile Software Development**

Transitioning to an Agile methodology is often very hard, given that Agile requires a change in mindset as well [57]. Additionally, Agile is highly collaborative, and most of the time, collaborative practices in developers grow gradually over time [19]. As such, only more experienced teams will lean towards more collaborative practices, emphasizing the need for experienced team members [19]. While emphasizing collaboration between team members and customers/users is not in itself a negative, it can be a limitation since project success is heavily dependent on the collaboration and cooperation of the user/customer [58].

Another pitfall of ASD is maintenance of the product, given the lack of documentation [58]. Agile is also best practiced under the assumption that developers have strong individual technical skills [58]. Last, the self-organized teams in Agile come with their own challenges, many requiring support from senior management and benefiting most when there is a mentor role to provide guidance and coaching for Agile methods [53].

**4. AGILE AND WATERFALL IN INDUSTRY**

Given the distinctive approaches to both ASD and Waterfall, [56] devised a study in order to understand the uses in industry for each SDLC. They interviewed experts from 19 international
IT companies spanning across Romania, America, Switzerland, Austria, and England [56]. Some of the experts that participated in their study included respondents from Google, Oracle, Gameloft, Ubisoft, and Luxoft, among others [56]. This study was carried out between November 2017 and February 2018 [56].

A survey was conducted in which each of the 19 respondents answered questions relating to the SDLCs in their companies [56]. One of the findings of the study was that the most implemented SDLC across these companies was the ASD, with 68% of respondents stating that Agile was the lifecycle used within their department [56]. The rest (32%) mentioned that Waterfall was the SDLC used in their department, with only 1 respondent stating that both Waterfall and Agile were used [56].

An interesting observation from the study was that out of those who responded that Waterfall was the SDLC used in their department, 50% were working between the years 2006-2012 at most, with half of these mentioning that ASD was implemented after 2011 in their department [56]. In addition, 85% of respondents who stated they were using Agile mentioned that amongst the advantages of using the ASD was the “ability to adapt to meeting end user requirements through a dynamic feedback” [56].

More recently, the latest State of Agile Report, which took place in 2022, reported that 80% of respondents are using the ASD and 24% are using Waterfall, with a few using a combination of the two [2]. This report consists of respondents from over 3,000 businesses and IT professionals worldwide and was conducted between July 2022 through August 2022 [2]. The report surveyed companies of all sizes, with 29% of companies having 1,000 or fewer employees and 34% of them with over 20,000 employees [2]. The study also included respondents from all
over the world, with the majority (55%) from North American and the next majority (25%) from Europe [2].
Chapter 4: Challenges in Software Engineering Education

This chapter dives into the different challenges that currently exist in software engineering education in institutions of higher learning. Different areas of identified knowledge deficiencies in computer science students are defined, followed by a subsection detailing the current state of Agile Software Development in software engineering education. This subsection includes various examples of how Agile has currently been applied in software engineering education, and the different models and strategies that have been proposed in the literature. Last, eduScrum is introduced, a framework that combines Scrum and education.

1. INTRODUCTION

Developing and designing the academic curricula for computer science and software engineering education is a challenging process [21]. The technical skills that students learn in university can quickly become obsolete due to these ever-evolving technologies [21]. The employability of recent graduates is of the utmost importance, as they are the bridge between academia (having recently graduated) and industry. Both industry and academia must address the gap between the expectations from companies and the graduates that academia is producing [5].

Universities have usually delivered technical skills well, while the non-technical skills – such as leadership, teamwork, communication, etc. – have usually been less supported in computer science programs [23], [31], [59]. These skills – commonly referred to as “soft skills” – heavily impact a team’s performance and work climate, given that the software engineering discipline relies heavily on collaboration and social interaction [23]. While many universities are aware of these issues, the challenge lies in how to develop these soft skills in computer science students.
2. AREAS OF IDENTIFIED KNOWLEDGE DEFICIENCIES

A literature review conducted by [7] revealed the most occurring knowledge deficiencies among graduating students in computer science. A knowledge deficiency was defined as “any skill, ability, or knowledge of concept which a recently graduated student lacks based on the expectations of industry or academia” [7]. The top six of these deficiencies are shown in Figure 16.

![Figure 16: Top 6 Most Identified Knowledge Deficiencies](image)

Communication was the first area that recent graduates were seen to struggle in. In this study, communication was broken down into oral communication and written communication. Within Oral Communication, graduates had difficulty in communicating with and listening to customers [7]. Within Written Communication, technical writing was found to be the ability with which most students struggled. In addition, other examples revealed that newly hired graduates lacked the ability to write documentation and provide an appropriate amount of detail in other forms of written communication, such as an email [7].

Project Management is a very important skill within the computer science field. Unfortunately, this study revealed that employers were unsatisfied with the project management
skills of graduates [7]. Another paper indicated that there was a large gap between what recent graduates knew and what they needed to know for their current job [60]. The last three skills – *Software Tools, Testing*, and *Teamwork* – had similar results, with the study suggesting that there was a gap between what graduates were taught regarding each of these skills and what industry expected of these graduates [7]. Software tools in the context of this study referred to any project management tools – including database tools, testing, and debugging tools – in which students lacked proficiency [7].

Given that many of the knowledge deficiencies identified are soft skills, it has been suggested that soft skills should be more included in academic curricula [13]. In fact, a survey conducted by [61] found that software engineering practitioners believe that academia should redesign their curriculum to include more coverage of soft skills.

The next section will describe the use of Agile Software Development in software engineering education, with a focus on different ways that Agile methodologies have been leveraged in academia.

**3. AGILE IN SOFTWARE ENGINEERING EDUCATION**

Agile has grown increasingly popular as a way to produce software in a quick and people-centered manner [1]. As such, common collaborative practices in Agile – such as pair programming, daily stand-up meetings, and retrospectives – have also become mainstream [19]. Unfortunately, studies have shown that these practices are rarely applied by inexperienced teams [19], [62]. This in turn leads to dysfunctional teams and disastrous projects within inexperienced Agile teams [19], [62]. In addition, considering that self-organized teams are at the heart of Agile, it is imperative that graduates in the computing disciplines are properly prepared with the skills of
teamwork, communication, and project management, as well as in knowing and learning Agile methodologies.

It is proposed in [1], [13], [19], [20], and [21] to craft a classroom experience with the ASD principles to expose students to industry-wide practices of Agile. Studies have suggested different frameworks and tools for teaching agile in the classroom. These are based upon models and principles used to guide the Agile experience in academia, producing different examples of how Agile has been taught in the classroom before. The following subsections describe different models and examples of how Agile has been leveraged in education.

i. The Pyramid of Agile Competences

Articles [16] and [19] claim that before developing an Agile software engineering course, it is important to understand the skills and competences needed for ASD. They propose the Pyramid of Agile Competences (see Figure 17), which breaks down the required competences into three major categories: Technical Practices, Collaboration Practices, and Agile Values [16], [19]. The pyramid in Figure 17 is read from bottom-to-top, with each category building on top of the other.

![Fig. 17: Pyramid of Agile Competences](image-url)
The first level of the pyramid is technical practices, that is, students must have the technical foundations for being able to develop high quality software [16]. These engineering practices can include unit testing, clean coding, test-driven development, refactoring, and continuous integration, among others [16], [19]. Generally, these practices are competences that refer to the single individual [16], and within the course project, the focus is on applying and refining these skills through practice [19].

On the second level of the pyramid lie the collaboration (or management) practices. These are specific to agile management practices, and they define how agile projects are organized and run [16]. These include collaboration practices within agile such as short release cycles, retrospectives, daily stand ups, iterative planning, pair programming, code reviews, and strong customer involvement [16], [19]. Given the emphasis on collaboration, communication, and feedback [19], these practices require social competences [16].

On the top of the pyramid lie the Agile values articulated in the Agile Manifesto. These are at the very top because they are of the utmost importance in ASD [19]. An ideal student project should create a realistic environment where students can experience and practice the Agile values [19]. The authors of [19] further claim that teaching the values is insufficient; students must experience these in practice.

The pyramid is organized in such a way that from the bottom to the top, it visualizes the decreasing number of required skills, consequently reflecting the increasing difficulty to teach these skills [16]. For example, technical practices can be taught in classes through lectures and can be learned by students at their own pace, while management practices are best taught through experiential learning [16]. That is, allowing students to practice these skills through projects in
teams [16]. Furthermore, the Agile values are the most difficult competences to teach, given that they require a change in the attitude of the student [16].

Utilizing the Pyramid of Agile Competences, [16] developed a 16-week Agile software engineering course in Zurich University of Applied Sciences for final year undergraduate students. Students were divided into teams of 6 to 8 members and were required to complete a project in Java [16]. The authors decided on using a Scrum-XP-hybrid claiming that they work well together because Scrum focuses on management practices while XP focuses on engineering practices, therefore complementing each other well [16].

The course was divided into two equal parts, each reflecting the competence pyramid in Figure 8. Every week there was a 2-hour lecture and a 2-hour programming workshop [16]. Weeks 1 to 7 (part one) focused on building the engineering practices, while weeks 8 to 14 (part two) focused on the second layer of the pyramid, i.e., the management practices [16]. The Agile values were introduced in the first part of the course as theory and applied/practiced in the second part of the course [16]. Table 1 shows the overview of the semester plan for the course.

<table>
<thead>
<tr>
<th>Week</th>
<th>Lecture</th>
<th>Workshop</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Agile Manifesto &amp; Introduction to XP</td>
<td>Install IDE &amp; Plug-Ins, Coding Assessment 1</td>
</tr>
<tr>
<td>2</td>
<td>XP continued &amp; Version Control</td>
<td>Version Control System, Coding Assessment 2</td>
</tr>
<tr>
<td>3</td>
<td>XP continued &amp; Project Automation</td>
<td>Build Scripts</td>
</tr>
<tr>
<td>4</td>
<td>Continuous Integration</td>
<td>CI (Jenkins Build Server)</td>
</tr>
<tr>
<td>5</td>
<td>Unit Testing</td>
<td>JUnit</td>
</tr>
<tr>
<td>6</td>
<td>Unit Testing continued, Mock Objects Clean Code, Code Smells</td>
<td>JUnit EasyMock</td>
</tr>
<tr>
<td>7</td>
<td>Refactoring</td>
<td>Refactoring</td>
</tr>
<tr>
<td>8</td>
<td>Introduction to Test-Driven Design Introduction to Scrum</td>
<td>TDD, The Craftsman articles</td>
</tr>
<tr>
<td>9</td>
<td>Scrum continued</td>
<td>Agile Game Development (Sprint 1)</td>
</tr>
</tbody>
</table>
In the last week of the semester, students were asked to fill in an evaluation form. A total of 24 students filled out the evaluation, and the results from the form were mostly positive [16]. As adapted from [16], Figure 18 illustrates some of the evaluations that students provided on the course. Using a scale of “Very bad”, “Bad”, “Good”, and “Excellent”, all students rated the content of the course between “Good” and “Excellent”, with no “Bad” nor “Very bad” scores [16]. In addition, most students rated their experience as “Good” for how the agile values came across in the lectures and the workshops [16]. The area with the most negative results was related to their experience working on a Scrum team, as can be seen in Figure 18, 4 students judged this concept as “Bad” [16].

![Fig. 18: Course Evaluation Results from [16]](image-url)
These findings – in particular the one regarding the Scrum team – lead to the following suggestion: special attention needs to be paid to the way that Scrum teams are put together [16]. From this study, the authors of [16] recommend that students should have access to a room where they can meet for stand ups and have a wall for the task board. A second finding suggested that working only a couple of hours a week on the project was not ideal – students themselves suggested a more intensive work week in which they would only work on the project within their Scrum teams [16]. The authors concluded by stating that ASD cannot be taught in an isolated software engineering course, and further work is necessary to learn how ASD can be successfully integrated into the computer science curriculum [16].

ii. Teaching Scrum in Software Engineering Courses

The latest State of Agile Report (2022), with respondents from over 3,000 businesses and IT professionals worldwide, revealed that Scrum is used by 87% of respondents, and 9 out of 10 of them are leveraging Scrum [2]. Given its popularity, [63], [64] have suggested that Scrum should be taught to students in university. However, understanding how Scrum should be incorporated into software engineering courses is a challenge [64]. As stated by the creators of Scrum, “Scrum is a lightweight process framework which is simple to understand but difficult to master” [65].

A review protocol conducted by [64] in which different studies were selected and classified by their topics allowed for the analysis of different approaches of teaching Scrum. One popular method from these studies was the design of a capstone software engineering course in which students were required to work in teams to develop a software project [64]. Two studies – [14] (discussed earlier) and [66] – used similar designs of dividing the semester into Sprints, with Sprint 0 (the first sprint) consisting of an introductory lecture into Scrum and the project they will
develop. A regular sprint would then consist of a planning meeting, daily stand ups, a sprint review, and a retrospective [64]. Each design differed in sprint length, the assignment of sprint roles, and the number of releases [64].

a. A Scrum Software Engineering Course

The software engineering course from [66] was designed by two researchers from the University of Kassel in Kassel, Germany. The authors’ main intent in their design of the course was to “teach students how to deliver a high-quality piece of software within a limited amount of time working in a team” [66]. The course – Software Engineering 1 (SE1) – is a mandatory course for all 4th-year students, as such, students are expected to have background knowledge from previous classes in programming, algorithms, data structures, and software methodologies [66]. The SE1 course is a 16-week long class in which students are divided into teams of 7-8 members and given a large project [66]. Scrum was selected as the framework to be used for project management within the teams for several reasons [66]. First, Scrum is the most widely used ASD methodology in industry [2]. Second, the authors determined that Scrum was well-suited for a 16-week course [66].

There were a total of 11 people as part of the course staff: the professor, 2 PhD research assistants supervising the course, 2 PhD research assistants who played the role of customers, and 6 undergraduate student research assistants to supervise each team as their tutor [66]. A descriptive breakdown of the role of each member of the staff team can be seen in Figure 19.
Fig. 19: Staff Roles of SE1 Course [66]

The semester was broken down into 4 releases with 2 sprints each, each release consisting of: release meetings where the Product Owner and the customer go through the requirements for the release, a sprint planning meeting with the whole team to go through requirements and user stories, developers working on their tasks, and at the end of the release cycle, a presentation by the Product Owner and Scrum Master in which a short summary of the work done is given to the course staff [66]. One important rule that was strictly observed was that students had to stay within their role. For example, the Scrum Master and Product Owner were both not allowed to engage in programming [66]. However, after each release, the role of the Product Owner and Scrum Master changed, allowing every student in the team to undertake at least one of these roles during the entire course duration [66].

The authors made attempts to balance the workload by allowing only 10 hours of work each week for a student [66]. Of these 10 hours, it is required for the student teams to work/meet for at least 2 hours in a designated room with the tutor present in order to enhance communication
and effectiveness [66]. Aside from the project work, students are required to meet every week with their tutors to discuss any problems beyond the scope of the Scrum Master [66].

The authors evaluated the time students spent per week on the course after each release, concluding that the workload of the course was very high, but acceptable [66]. Many students spent more than the recommended 10 hours per week on the course, however this time was spent voluntarily by the students to improve on the software project by adding additional features not listed in the requirements [66]. About two-thirds of the students worked at least 5 hours per week, with only 20% working more than 10 hours [66].

Once the course was finished, the authors issued a comprehensive survey to students with a rating skill from 1 (not sufficient) to 6 (very good). From the survey, results showed that approximately 80% of the students were satisfied with the content and the structure of the SE1 course [66]. The technical and social skills of students improved, with 70% of students rating these skills from satisfying to very good [66]. However, only 24% of students rated their teamwork as good or very good, failing to meet the primary objective concerning teamwork for the course [66]. Final conclusions remarked on how this course was very time consuming and required a large staff, leading to challenges concerning communication between staff [66]. The authors suggested that more support and guidance is needed for the teams to understand the required workflow, especially as it pertains to adapting the Scrum process [66].

b. An Agile Software Development Course with Scrum

Similar results were found by [12] in their Agile Software Development (ISE425) course taken by final-year Information Systems Engineering undergraduate students at Sakarya University in Turkey. Their study presented an Agile teaching experience using Scrum and Lean practices with a goal of providing theoretical Agile foundations to students complemented with
hands-on experiences through the development of a project within a self-organized team of 6 to 9 members [12]. Scrum was selected as the Agile methodology to teach and practice due to it being the most widely used and practiced Agile methodology in industry [2]. Kanban was also introduced to the students to manage their project development for the course [12].

The professor divided the course content as shown in Table 2, with the entire duration of the class being 15 weeks [12]. The structure of the semester was to introduce students to the basics and fundamentals of Agile, agility, and Scrum, offering an “enlighten and caution” after each lecture in which the professor included discussions about evangelizing Agile as to caution the students about misleading concepts about agility [12]. The professor also provided input throughout the lectures about their own experience being an Agile practitioner for many years in industry, in addition to inviting 2 guest speakers at the end of the semester from the IT industry to give talks about their own experiences [12].

Table 2: Breakdown of the Course [12]

<table>
<thead>
<tr>
<th>Week</th>
<th>Topic(s)</th>
</tr>
</thead>
</table>
| 1-4  | • Definition of agile; Introduction to Agile.  
      | • Agile in software development  
      | • Different software methodologies, problems with traditional methodologies, different approaches  
      | • Agile values, principles, and practices  
      | • Agile manifesto  
      | • Agile in industry |
| 5-7  | • Scrum introduced.  
      | • Textbook provided: Essential Scrum: “A practical guide to the most popular Agile process” (K. S. Rubin) |
| 8    | • Midterm exam |
| 9    | • List of Agile methods  
      | • How to scale Agile  
      | • Case study of Agile transformation in a bank |
| 10   | • Principles of Lean  
      | • Focus on pull system, continuous flow, and management |
At the end of the course, the professor conducted interviews and surveys with the students to collect data on the course’s effectiveness [12]. Through these interviews and surveys, it was reported that students found the task of assigning Scrum Master, Product Owner, and other team member roles was critical to the project success, and tasks/responsibilities should be distributed by considering every team member’s ability [12]. One area that faced many challenges was communication with the customer [12]. Since each team was required to find real customers for their projects, many different challenges presented themselves, such as: customers being overly demanding, and delayed project progress due to being unable to get quick feedback from the customers [12].

Students also reported challenges faced within teamwork, such as certain members not understanding/realizing their responsibilities, forcing other team members to take on extra workload [12]. Additionally, the teams had issues with complying with the daily standup meetings due to challenges in finding suitable time slots for all team members to get together [12]. As such, online meetings were found to be more productive and conducted more often, with the help of online flow management tools – such as Trello – to help teams collaborate [12].

Through the interviews and surveys, students reported that the most valuable parts of the lesson were teamwork, a close connection between the class and industry, learning and practicing Agile, and the experience provided by the professor as an Agile practitioner themselves [12]. For improvements on the course, students suggested providing more concrete examples from industry explaining ASD and the Scrum methodology, as well as providing more time to practice these

| 11-13 | • Project presentations |
| 14    | • Guest speakers        |
| 15    | • Final exam            |
methodologies rather than allotting so much time to the delivery of the theoretical information [12].

The last remarks made by the author state that while Agile should be taught in university, it needs to be tailored to suit the university context [12]. This can include variations made in sprint lengths, the frequencies of standup meetings, team roles, and the use of online tools [12]. Last, the author found it a challenging process to have a combination of teaching the theory of Agile, practicing Agile multiple times, and revisiting the theory based on these experiences all in the duration of one semester [12].

c. A Hybrid Model with Problem-Based Learning and Scrum in a Software Engineering Course

In two Federal Universities in Brazil, the authors of [67] carried out two separate experiments in order to test out a proposed model with undergraduate students in software engineering and computer engineering. Their model leveraged problem based learning and active learning in order to use real-world problems to motivate student learning [67]. The problem is the starting point by which the student is motivated, starting the learning process [67]. Problem based learning is combined with Scrum in the proposed hybrid model to give the students a project management framework by which to develop their project [67].

The typical practices found in Scrum – daily stand ups, sprints, backlogs – were all integrated as part of this model [67]. The first activity that takes place under this hybrid model (see Fig. 20) is the presentation of the problem followed by compiling a list of items that will guide the solution of the problem [67]. In this way, both problem based learning (presenting the problem) and Scrum (making a product backlog) are combined to facilitate problem solving [67]. After this step, the professor and the students break down the product backlog into a set of solutions, coming
up with the sprint backlog that will be developed in 30-day sprints [67]. Once this phase is finished, students execute the sprint, where they practice daily stand ups with the team and the professor (who plays the role of project manager) to review the work done so far [67].

Fig. 20: Proposed Hybrid Problem Based Learning/Scrum Model [67]

In addition, Weekly Meetings – a monitoring and control tool of the model – took place every week to allow the professor a more in-depth review of the work accomplished by each student, the struggles/challenges faced, and the planned activities for the coming week [67]. In addition, the professor provided guidance and support to each student [67]. At the end of each sprint, the team delivered/presented the product to the professor for review and feedback, after which a new sprint backlog is generated, and the next sprint is started [67]. This process continues until the final sprint is completed and the full product is delivered at the end of the course [67].

The case studies that were carried out in both universities involved a total of 64 students who were tasked with solving problems for game and mobile application development projects
utilizing the hybrid problem-based learning/Scrum model [67]. At the end of each course, interviews were conducted with the students in order to gather data about their opinions regarding the experiment [67]. After the data collection and analysis was performed, it was shown that students had positive experiences with the Weekly Meetings, as it allowed communication between a student and the professor to be facilitated [67]. Project management also had a satisfactory result based on the analysis of the interviews with students [67]. Overall, the hybrid model had positive results, with students citing the Weekly Meetings as very helpful to their learning, motivation, communication, and project success [67].

iii. eduScrum

This subsection introduces eduScrum, a framework that has its origins in both “education and Scrum” [68]. Derived from Scrum, eduScrum is a lightweight, easy to understand but difficult to master framework that enables professors and students to tackle complex, challenging problems while pursuing learning objectives in a productive and creative manner [68]. eduScrum was created by Willy Wijnands, a professor of chemistry and physics in Ashram College in Alphen aan den Rijn, Netherlands, in 2011 [69]. Wijnands is also the co-founder of the worldwide initiative “Agile in Education” [69].

One of the main challenges of eduScrum is that it only prescribes the “Why” and the “What”, but not the “How” [68]. Since it is a framework, it can use various processes and techniques, and students have the freedom to decide, requiring professors to have an agile mindset [68]. Similar to Scrum, eduScrum consists of self-organized teams with their associated roles and practices, however, the specific implementation strategies used may differ, and are therefore not explicitly defined within the framework [68]. eduScrum is based on experiential learning theory,
with three foundational pillars: transparency, review, and adaptation [68]. A detailed figure of the eduScrum cycle can be seen in Figure 21 [68].

![Fig. 21: eduScrum Cycle as Adapted From [68]](image)

**a. Transparency**

This pillar stipulates that certain aspects of the process must be defined according to a common standard such that all involved have a common understanding of what is being seen [68]. One such example of this is having a common language related to the process that is shared across all participants, such as understanding what the word “Ready” means in the context of the work to be performed and what the working product should look like [68]. In this way, eduScrum is intended to make information transparent to help students make the right decisions in their work, supporting their learning process and maximizing value [68].

**b. Review**

This pillar is also referred to as “Inspection” [68]. It states that users of the eduScrum framework must frequently review the eduScrum elements and progress towards the learning goals [68]. The intention of this is to detect undesirable deviations early and make sure that everyone is on the correct path for the learning objectives [68]. Reviews are most useful when they are carried
out by both the students and the professor(s) in the setting where the work is done (the classroom) [68].

c. Adaptation

Students and professors should be ready to adjust the process if one or more aspects are not working [68]. Further, the adjustment must take place as quickly as possible in order to limit further deviations from the expected outcome [68]. These means that students and the professor need to be flexible and ready to adapt to new processes.

d. The eduScrum Team

Teams within the eduScrum framework are self-organized and consist of a Product Owner (played by the professor), an eduScrum Master (shared role between the professor and a student), and the rest of the team members [68]. The eduScrum Master role played by the student is called team captain [68]. Team sizes vary between 4 to 6 students, and the students are in charge of putting themselves together in their teams [68]. eduScrum teams are encouraged to collaborate beyond their own teams in order to seek insight from other teams, however each team is responsible for their own work and results [68].

The team captain (or student eduScrum Master) ensures that the team is performing optimally, however, this student is not above the rest of the team – they are an equal team member [68]. This student can be designated by the professor or chosen by the team [68]. eduScrum limits the role of the eduScrum Master as compared to the traditional role of the Scrum Master by dividing the typical tasks/responsibilities that would befall the Scrum Master between the professor and the student [68]. Therefore, eduScrum ensures that as the student eduScrum Master gains more experience, they are given more responsibilities from the professor, meaning that the professor’s total responsibilities gradually decrease [68].
One important responsibility that befalls the team captain is taking care of the “Flap” [68]. The “Flap” is the “visual board that makes the work and agreements of the team visible” [68]. It is also known as the eduScrum Board [68]. Tasks within the “Flap” move based on their status, which can either be “To Do”, “Busy”, or “Done” [68]. It is the team captain’s responsibility to make sure that the “Flap” is available when needed and that it is continuously updated to reflect the current work [68]. The team captain is also responsible for initiating cross-team collaboration, ensuring that eduScrum is being performed correctly, and leading the removal of impediments in the team’s progress [68].

An important factor of eduScrum teams that must be observed is that nobody – not even the professor – can tell the team how to achieve the learning objectives [68]. In this way, student teams are given the power to self-organize and self-manage their own work [68]. These teams must also be multidisciplinary in the sense that collectively they have all the necessary skills and personal development areas to achieve the learning goals [68]. While certain students might have specific skills or strengths, it is the responsibility of the team to decide how to best utilize their skills and develop new areas [68]. Last, each team is responsible for monitoring their own progress and quality based on the Celebration Criteria (explained in a later section) [68].

**e. The Professor**

The professor takes on a hybrid role between Product Owner and eduScrum Master, sharing the latter with a student on each team [68]. The professor’s responsibility is to determine the learning objectives, monitor the eduScrum process, and facilitate the learning process of the students [68]. Additionally, the professor should also encourage cross-team collaboration and determine how this will be done [68]. It is also up to the professor to come up with what and why to learn [68].
eduScrum makes use of the Celebration Criteria (determined by the professor) to set the requirements which the assignments/projects must meet [68]. These criteria are established in advance and shared with the students [68]. The criteria can include requirements for exams, presentations, rubrics, deadlines, and other constraints on the work to be delivered [68]. Aside from the professor-developed Celebration Criteria, each student team must also come up with their Celebration Criteria for their own projects/products [68].

f. Ceremonies

There are multiple different events/practices – referred to as “ceremonies” – in eduScrum that take place [68]. At the heart of eduScrum is the sprint, which can consist of lessons, a project, a reading from a chapter, and so on [68]. Within the eduScrum framework, a sprint has a duration of approximately 7 weeks, depending on the course schedule and length [68]. Sprints consist of the planning meeting, stand ups, task execution, reviews, and retrospectives [68].

Another ceremony practiced within eduScrum is team formation, which focuses on the careful formation of a team based on qualities and skills of students, as well as the work that needs to be completed [68]. There are certain criteria that guide the creation of a team, including [68]:

- the qualities of team members are complementary
- there is a balanced gender distribution
- knowing that teams can be changed for each assignment
- team formation based on existing friendships between students is undesirable

A team formation typically takes place by first selecting a team captain [68]. The team captain is selected by the professor or by the rest of the class [68]. Afterwards, the team captain chooses the additional members by assessing the different qualities they bring to the team [68]. This ceremony is part of the sprint planning [68].
g. An Example of eduScrum in the Classroom

At the University of Applied Sciences and Arts – Hochschule Hannover in Germany, the authors of [70] adapted eduScrum within their Innovative Methods of Project Management course for students enrolled in the Master of Science program in Digital Transformation. They performed the course with the eduScrum framework twice: once in the summer of 2019 in person, and in the summer of 2020 online due to the COVID-19 pandemic [70]. Among their learning objectives was for students to understand the differences between agile, linear, and hybrid lifecycles, as well as learning about leadership and teamwork via project work [70]. Each course was divided into the following different phases [70]:

- Lecture phase: 16 weeks
- Exam phase: 3 weeks
- Lecture-free phase: 7 weeks

Utilizing the eduScrum Guide, the authors adapted the roles described in eduScrum – Product Owner, eduScrum Master/Team Captain – while also adding the role of an Agile Coach and changing the responsibilities of the Product Owner [70]. They ended up with 4 roles, described in Figure 22 [70]. For the role of the eduScrum Master, rather than having one student play this role for the entire course, the authors decided to make this a rotating role, such that in each sprint, it would fall to another student, ensuring that every student had the opportunity to play this role during the term [70]. Student teams were made up of 3 to 7 members, and students were assigned to their teams [70].
Similarly, the authors adapted the following ceremonies from eduScrum in the course [70]:

- **Sprint planning**: plan and organize the work for the current sprint. Students work with the Product Owner to determine what requirements will be worked on for the given sprint. The selected product backlog items compose the sprint goal and are used to determine what tasks will need to be completed to achieve the goal. Additionally, students use this time to select their new eduScrum Master. If, however, the entire team agrees, this role can be omitted for the given sprint. The team chooses whether to estimate tasks in terms of complexity or effort, and the Agile Coach should help the team perform an estimation method. This ceremony is estimated to last about 45 minutes.

- **Sprint review**: practiced at the end of every sprint. The team presents their results for the past sprint to the Product Owner and Agile Coach. The Product Owner accepts or rejects the product backlog items reflected in the results. The rejected items are transferred back to the Product Backlog for the next sprint. The Agile Coach, focusing on the learning objectives of the course, provides feedback to the team. This ceremony is estimated to last about 30 minutes.
• Sprint retrospective: for this course, the retrospective was optional, and was only conducted if the team agreed to it. If so, the Agile Coach provides support and guidance to conduct the retrospective. This ceremony is estimated to last about 45 minutes.

• Stand-up meeting: the recommendation provided by the eduScrum Guide was followed for stand-up meetings. Therefore, a 5-minute stand-up meeting was performed at the beginning of each lesson by each team.

• Refinement: this event takes place in the week when no sprint planning occurs. The purpose of this event is to allow the Product Owner to present or explain any new or changed requirements. This is a time for students to ask questions directly to the Product Owner and delegate tasks. This ceremony is estimated to last about 30 minutes.

The authors also adapted certain artifacts from eduScrum, such as the “Flap”, as follows [70]:

• Product backlog: given that the eduScrum Guide does not provide any explicit artifact for a backlog, the authors added a product backlog to contain the items representing the product requirements. These are documented using user stories. The Product Owner is responsible for the product backlog, and for prioritizing tasks in the backlog. The team and the Agile Coach can also add their own requirements to the product backlog.

• Sprint backlog: this backlog contains the selected tasks to be worked on for the given sprint. It is up to the team to decide how the sprint backlog will be visualized. The backlog should always be updated to reflect the team’s current progress. It also serves as communication with the Product Owner and the Agile Coach during the sprint.

• Definition of Done: a well-known artifact commonly used in Scrum, the teams use this to describe the acceptance criteria for requirements. In addition, it is used to define work agreements for the team.
While eduScrum sets the sprint length to approximately 7 weeks, the authors of [70], who found this to be too long, adjusted the length to 2 weeks. Therefore, each ceremony (described above) takes place in 2-week sprints [70]. The Agile Coach and Product Owner reserve the right to cancel any given sprint [70].

During the summer 2019 course, the authors asked 7 companies to provide a Product Owner and a project, narrowing down the selection to 4 companies [70]. The selected projects were all based on real world problems [70]. The Product Owner from each company prepared a product backlog and presented it to the student teams [70]. The class consisted of 4 student teams, each with one of the company projects, and 2 professors, each playing the role of Agile Coach [70]. Each professor oversaw 2 teams each, with the class setup as described on Figure 23 [70]:

![Fig. 23: Class Setup Utilizing eduScrum for [70]](image)

Once groups and projects were established, the professors held an introductory class in which they introduced themselves, the course objectives, rules, and asked the students about their experience with project management [70]. During this first course, the professors also introduced the students to agile, the Agile Manifesto, Scrum, and Waterfall [70]. The second class was considered the “kick-off” in which each student team met with their Product Owner, who presented
the project and goals [70]. The professors also took part in these events, helping the student teams organize and plan their sprint [70]. After the kick-off, the first iteration of eduScrum started with the sprint planning ceremony, followed by refinement in the next week, review, and retrospective [70]. An overview of this cycle can be seen in Figure 24 below [70]:

![Course Structure for Summer Term 2019](image)

**Fig. 24: Course Structure for Summer Term 2019**

The Agile Coach and Product Owners partook in the weekly events and provided feedback on each team’s progress, in addition to the Product Owner always explaining the current product backlog and any new tasks [70]. At the end of the course, the professors used a survey to evaluate the course [70]. The survey, along with their own observations of eduScrum, allowed them to collect data from students on the course [70]. Informal discussions were held regularly with both student teams and Product Owners, each of which was documented for data collection [70]. Last, review discussions were held at the end of the summer term with the Product Owners [70]. The last piece of data considered came from the paper exams taken by students and the student team presentations [70].

A total of 21 students answered the survey [70]. The questions were based on a Likert scale between 1 through 5, where 1 was “completely applicable” and 5 was “completely inapplicable” [70]. An adaptation of some of the survey items and their average values are summarized on Table 3 below [70]:
Table 3: Certain Survey Items with their Average Scores [70]

<table>
<thead>
<tr>
<th>Survey Item</th>
<th>Average Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>“I rate the overall content of the course positive.”</td>
<td>1.7</td>
</tr>
<tr>
<td>“My understanding of project management in real world problems increased.”</td>
<td>1.4</td>
</tr>
<tr>
<td>“I was interested by the course objectives and content.”</td>
<td>1.9</td>
</tr>
<tr>
<td>“I rate the overall eduScrum framework positively.”</td>
<td>1.6</td>
</tr>
<tr>
<td>“The connection between the course material importance to my professional activity was clear to me.”</td>
<td>1.5</td>
</tr>
</tbody>
</table>

From these results, the authors concluded that one of their main goals – integrating real world problems into the course and students’ learning – was achieved for the 2019 summer course [70]. For the summer 2020 virtual adaptation of the course, the authors decided not to take collected data into account, given that they did not see the two courses as comparable in terms of implementation [70]. For example, the first two weeks of the course were in person, but as the pandemic reached Germany, the course had to me moved online, therefore new issues – such as collaboration, coordination, communication, coaching and mentoring remotely – arised for the professors to tackle [70]. eduScrum was also not fully adopted in the virtual class [70].

Concluding remarks from the authors state that the study is limited by the small number of students who took the course and responded to the survey in the summer 2019 term (21 students) [70]. In addition, only a single survey was carried out towards the end of the semester, whereas an additional survey conducted in the middle of the semester would have provided more context and feedback [70].
Chapter 5: Methodology

This chapter will describe the methodology used for this research. Using the background research, 3 different surveys were created and presented to students. The data from each survey was analyzed and used to aid in the creation of a novel educational software methodology, which is also presented in this chapter. The first section describes the creation of each survey and how they were each presented to students, followed by the data analysis and results. The next section introduces a mapping between the Agile principles and different educational principles for engineering students. Last, the proposed educational software methodology is introduced and described in detail.

1. Survey Creation

To understand students’ perspectives and experiences in computer science courses related to software development lifecycles and learning models, 3 different surveys were created using QuestionPro. Surveys were chosen as the approach to understand students’ perspectives and experiences because surveys allow for the collection of more data in a shorter amount of time, given that these can be distributed to large groups of students at once, as opposed to other approaches (such as interviews). All surveys were presented in multiple computer science courses at the University of Texas at El Paso, primarily to Junior, Senior, and graduate students. The first 2 surveys – named Discovery Survey 1 and Discovery Survey 2 – were presented in the Spring 2023 semester. Discovery Survey 1 was presented approximately the 10th week of the semester and Discovery Survey 2 was presented during the 14th week of the semester. The third survey – entitled Teamwork, Learning Processes, and Lifecycles Survey – was presented approximately the 6th and 7th weeks of the Fall 2023 semester. Below is a list of all the classes in which the surveys were presented to students:
Spring 2023:

- CS 2302: Data Structures
- CS 3331: Advanced Object-Oriented Programming
- CS 4310: Software Engineering: Requirements Engineering
- CS 4311: Software Engineering: Design & Implementation

Fall 2023:

- CS 3331: Advanced Object-Oriented Programming
- CS 4310: Software Engineering: Requirements Engineering
- CS 4311: Software Engineering: Design & Implementation
- CS 4374/5374: Software Construction

These classes were selected because they target primarily Junior and Senior students. Based on the literature, project-based software development classes typically occur at the Senior level courses. As such, the primary target for data collection through these surveys were Junior and Senior level students.

All three surveys contained the same 2 demographic questions, which asked students to select which computer science classes they were currently taking and what their current classification was. Aside from these demographic questions, all other questions in each survey were different. The Teamwork, Learning Processes, and Lifecycles Survey had additional demographic questions, explained in a later section. All three surveys consisted of a mix of multiple choice and multiple response questions, with the discovery surveys also including free response questions.
After IRB approval, (IRB consent in the appendix) each survey was introduced to the students with a brief explanation of the purposes of the study. It was also emphasized that participation in the survey was voluntary and, if they chose to participate, required an informed consent. After introducing the survey, students were given approximately 20 minutes of class time to participate in the survey.

The responses that were considered in the data for each survey had to meet the following requirements:

1. The response was fully complete, as in, the entire survey was responded by the student
2. The response was a unique response by the student – duplicate responses by a single student were invalidated

i. Discovery Survey 1

The purpose of this survey was to gauge a general understanding of students’ perspectives in different areas pertaining to this research. Since one of the focuses of this study was to understand the different ways in which students learn and their familiarity with different software development lifecycles, the questions for this survey were more general in order to get an initial idea of students’ experiences in these areas. This survey was divided into the following blocks:

1. Project Based Learning: this block contained different questions asking students about their familiarity and experiences with project-based learning. Students were asked to describe their experiences in project-based learning courses if they had ever experienced one. In addition, students were asked about their experiences in lecture-based and non-lecture-based courses.
2. Software Development Lifecycles: this block focused on asking students about their familiarity with Agile and Waterfall. First, students were asked if they had ever heard of SDLCs, followed by a question asking them to select all of the SDLCs that they had heard of, then to select all those that they had ever used to implement a project inside a course. They were also asked to select all the SDLCs they had every used to implement a project outside of school. Each question included an option for students to select if they had never used an SDLC for a project, whether inside or outside of school.

3. Group Work: this block asked students a variety of questions pertaining to their preferences for group work. Some questions gave an example scenario of a project and asked students if they preferred working in groups or individually on that project. Other questions asked students for their opinions regarding what they believed to be an optimal group size based on an example project. Students were also asked about their experiences in group work in general.

Before checking the data against the requirements for consideration (described in the previous section), this survey had a total of 290 responses from students.

ii. Discovery Survey 2

The objective of this survey was to gather more data on students’ perceptions on group work, project work, collaborative practices, software tools, and learning processes. The questions about team projects were narrowed even further than those in the first survey, asking about collaborative practices that students have partaken in both inside and outside of school. This survey contained a total of 30 questions (excluding the informed consent and demographic questions). The survey was divided into blocks as follows:
1. **Group Work & Project Work**: this block consisted of questions asking students more about group work. One question asked if they had, in their college experience so far within CS courses, worked more independently on projects or in groups. Questions were also asked about which classes have assigned the most group work/projects, what their experiences have been in group work, and which collaborative practices have they applied in any of their group projects. All these questions were concerning computer science classes. For comparison purposes, students were also asked some similar questions pertaining to group project experiences outside of school. Students were also asked some questions related to self-organized teams.

2. **Software Tools**: a few questions about students’ familiarities with software tools were asked. In addition, students were asked to select all of the software tools (from a given list) that they had ever used on a group project. This question was asked twice, once for a group project in their CS classes and another for a group project outside of school. Students were also asked to list any software tools that they used outside of school that they did not learn to use in any of their CS classes.

3. **Learning Processes**: students were asked about their attention during classes, specifically if they believed their attention declined at any point during lectures in their CS courses. If they responded yes to this question, they were then asked to select at which point(s) during the lecture they believed their attention declined. A few questions were asked regarding the types of activities during class that helped maintain their attention.
Questions that asked about “outside of school” always referred to internships, co-ops, and/or jobs related to software engineering. The intention of the questions within the Learning Processes block was to see if the experiences of students matched the literature pertaining to the learning processes of students. This survey had a total of 158 student responses before checking the data for duplicate and incomplete responses.

iii. Teamwork, Learning Processes, and Lifecycles Survey

This survey was created after a thorough analysis of the data collected from the first 2 surveys. This allowed for questions to be more structured in order to answer questions that the first surveys had not yet answered. In addition, this survey extensively used from literature existing questionnaires and models. This survey had a total of 265 responses. Questions were divided into different blocks as follows:

1. Demographics: this first block consisted of more specific demographic questions. While the first surveys did include 2 demographic questions – one about their classification and another about which classes they were currently taking – this survey included a total of 6 demographic questions. More specific questions were asked, such as (for graduate students only) what their graduate program was, (for undergraduate students only) what their concentration was, and, aside from which classes they were currently taking, which classes they had taken before. The reason for asking more specific demographic questions was to allow for a different data aggregate.

2. Agile Teamwork Experience: this block consisted of a single question focused on the student’s experiences working within an Agile team within an ASD course. The question utilizes parts of an existing questionnaire developed by [71] for assessing team effectiveness. It uses a 5-point Likert scale ranging from “Strongly Disagree” to
“Strongly Agree” [71]. This questionnaire was used to measure the student’s ability to effectively work in teams within the ASD course. Although the complete questionnaire contains over 40 questions, only 23 were selected, based on their relevance to this research. This questionnaire was only shown to students who had, in the demographics block, selected the ASD course as one of the courses that they had previously taken.

3. Agile Knowledge-Building Experience: This block consisted of a question based off the identified areas of knowledge deficiencies by [7]. This question also uses a Likert scale ranging from “Not at all” to “Very Well” in order to assess how the ASD course helped the student build the knowledge area. Similar to the first question, this question was only shown to students who had selected the ASD course as one of the courses that they had previously taken.

4. Other Teamwork Experience: this block focused on questions related to any other teamwork experience within a software development course that did not utilize Agile. Thus, students were first asked if they had ever worked on a team of at least 3 members in a software engineering course on a software development project that lasted longer than one week. If the students responded “Yes” to this question, they would be shown the same team effectiveness questionnaire from the second block in order to measure their ability to effectively work in teams outside of ASD. In order to avoid duplicate responses from students who answered this questionnaire pertaining to the Agile course, QuestionPro’s logic feature was used to do the following:

   a. If a student had taken the ASD course, they took the team effectiveness questionnaire for the Agile course. (This was part of the second block described above)
b. If a student had taken the ASD course, they were asked if they had ever worked on a group project of at least 3 members in a software engineering course that lasted longer than a week other than the ASD course. If so, they took the questionnaire again for that other course.

c. If a student had not taken the ASD course, they were also asked if they had ever worked on a group project of at least 3 members in a software engineering course that lasted longer than a week, and if they had, they took the questionnaire again for that course.

Aside from these questions, students were also asked more general questions about the types of group work they had participated in through their CS courses.

4. Learning Processes: this block consists of questions about students’ learning processes. Some descriptions of the questions are as follows:

a. One of the questions uses the examples of class activities for each learning mode described earlier provided by the DOLA Framework in [26] along with a Likert scale ranging from “Not at all helpful” to “Extremely helpful” to ask students how helpful they find different activities towards their learning during a class. Each activity can be mapped to one of the learning modes presented under the DOLA Framework. The purpose of this question is to assess students’ perceptions of different learning modes based on how helpful they find them towards their learning in the classroom.

b. Another question from this section uses the MUSIC Inventory Survey, 19-questions version, as specified in [72], which is based on the MUSIC Model of Motivation. This model was developed by Dr. Brett Jones to help instructors
“(a) design instruction that motivates students, (b) to diagnose motivational strengths and weaknesses of instruction, and (c) to research relationships among factors critical to student motivation” [72]. The model has 5 principles [73]:

i. **Empowerment**: students should feel empowered by having the freedom to make decisions about certain aspects of their learning,

ii. **Useful**: students should understand why what they are learning is useful for their goals,

iii. **Succeed**: students should believe that they are capable of succeeding if they put the effort required to do so,

iv. **Interested**: students should be interested in the content being taught and in the instructional activities, and

v. **Care**: instructors should ensure that student believe that others in the learning environment, including the instructor and other students, care about their learning.

From the MUSIC Model, Dr. Jones developed the MUSIC Inventory Survey to ensure that all the items under each principle of the model followed a similar format and would provide valid scores [72]. Dr. Jones created multiple versions of the MUSIC Inventory Survey [72]. For the purposes of this survey and per recommendation from Dr. Jones, the 19-item College Student version was used. According to Dr. Jones, this version is better suited for college students because it takes less time to complete, and it still provides equal validity of scores as the 26-item version (the longest version) [72]. All the detailed instructions provided
in the guide for the presentation of this survey were followed. The questions each use a scale from 1 to 6, with 1 being “Strongly Disagree” and 6 being “Strongly Agree”. Per Dr. Jones instructions on the guide, the survey was adopted for students to respond to the questions pertaining to the overall program rather than a single course.

5. Agile & Waterfall: this final block of questions pertains to Agile and Waterfall. Based on observations from the first 2 surveys which indicated that students might have interpreted different definitions for Agile and Waterfall, a definition for each cycle was provided in the questions, as well as figures illustrating each lifecycle. These questions focused on asking student about their experiences working with each lifecycle.

2. RESULTS & DATA ANALYSIS

The data analysis process was different for each survey. The third survey did not contain any free-response questions, but it did make use of different questionnaires and models as detailed above. For the question pertaining specifically to the MUSIC Model, a formula was provided in the user guide to score the values. IBM SPSS and IBM AMOS were used in the statistical analysis of the data. Given the length of all surveys, only a subset of questions most relevant to this research will be discussed in the following sections.

i. Discovery Survey 1

This survey had a total of 290 responses. As mentioned in an earlier section, for the purposes of this study, only responses that were fully completed – as in, the entire survey was completely answered by the student – were considered in the data. This reduced the number to 225 completed responses. From these 225 responses, 3 students responded twice, therefore disqualifying a total of 6 responses, such that only 219 responses were considered in the data.
These duplicate responses were not considered in the data due to them differing in responses in certain questions for the same student who took the survey twice. Figure 25 shows the classification of all students when they partook in the survey (Spring 2023). The majority of respondents (63%) were Seniors. There were no Freshmen students. The rest of this section will breakdown a subset of questions followed by a discussion and analysis on the responses.

![Studnet Classification Pie Chart]

Fig. 25: Student Classifications from the First Survey

**Question 1:**

("Have you ever heard of Software Development Lifecycles?"")

70% of respondents stated that they had heard of software development lifecycles. The overwhelming majority of these – about 87% – were Senior or graduate students. Figure 26 shows a comparison of 2 groups of students: Sophomore and Junior students in one group, and Senior and graduate students in the other group. Figure 26 shows the answers for all respondents split up into these 2 groups. Based on this figure, a correlation between the classification of the student and their familiarity with SDLCs can be observed, with students in higher classifications – Seniors & graduates – having more familiarity with SDLCs than those in lower classifications.
Fig. 26: Student Familiarity with SDLCs

**Question 2 & Question 3:**

“Please select all of the Software Development Lifecycles that you have ever used yourself to implement a project inside of the classroom, whether this was on a group or individually:”, and “Please select all of the Software Development Lifecycles that you have ever used yourself to implement a project outside of the classroom (whether at an internship, co-op, or job):”

These questions aimed to compare students’ use of SDLCs inside and outside of the classroom. As such, students were asked this same question twice, one relating to inside a class and the other to any outside experience (such as an internship, job, and/or co-op) related to computer science. In addition to a list of different SDLCs, student were also given 2 exclusive options, one if they were unsure, and another one if they had never used an SDLC for a project. As such, the data was qualified by considering the responses for each software development lifecycle.

Question 2 had 219 responses, of which 136 were qualified. Question 3 was only responded by students who met the following 2 conditions: (1) they had partaken in an internship, job, and/or
co-op related to computer science, and (2) they had completed a project during that experience. As such, Question 3 received a total of 73 responses, of which 50 were qualified.

Figure 27 shows a comparison between these 2 questions, using the mean for each SDLC. The mean for each SDLC ranges from 0 to 1, where 0 represents that the SDLC was not selected, and 1 represents that it was.

![Percentages of SDLCs Used Inside of School and Outside of School](image)

Fig. 27: Percentages of SDLCs Used Inside of School and Outside of School

From Figure 27, the use of Agile both inside and outside of school is largest, with 37.8% of students having used it in school and 57.7% for outside of school. Waterfall was the second largest, with a 24.1% in school and 14.1% outside of school.

**Question 4:**

“How familiar are you with the Waterfall and Agile Software Development lifecycles?”

This question used 2 Likert-scales, one for each lifecycle. Each scale ranged from “Very Unfamiliar” (1) to “Very Familiar” (5). Both lifecycles showed similar patterns when comparing the raw data from all students (see Figure 28).
Descriptive statistics for this data showed that there was a correlation between the student’s classification and their familiarity with each lifecycle. Students were grouped by classification into 2 groups: Sophomores and Juniors in the first group, and Seniors and graduates in the second group. Figures 29 and 30 show a boxplot for each lifecycle describing the relationship between the student’s classification and their familiarity with each lifecycle. Not only is this correlation evident from the figures, but it can also be observed that there are outliers in the data. Figures 29 and 30 show the familiarity levels (1 – 5) on the y-axis and the classification on the x-axis. The outliers in both figures represent how, for Sophomore or Juniors, higher familiarity with either lifecycle was uncommon, such that these data points were outliers. For Seniors and graduates, the outliers represented lower familiarity with either lifecycle, meaning that most Seniors and graduates had higher levels of familiarity with either lifecycle.

Fig. 28: Student Familiarity with Waterfall and Agile
Fig. 29: Box Plots Denoting the Relationship Between a Student’s Classification and Their Familiarity with Agile

Fig. 30: Box Plots Denoting the Relationship Between a Student’s Classification and Their Familiarity with Waterfall
Given the observations from the box plots, two independent t-tests were performed on the data by comparing the familiarity with each lifecycle for the 2 groups of student classifications: Sophomores and Juniors in one group, and Senior and graduates in the second group. An inspection of the data revealed that the data was normally distributed and that, per Levene’s Test for Equality of Variances, there was homogeneity of variances for both groups. As such, the independent t-tests were run, each with a 95% confidence interval for the mean difference. Table 4 shows the group statistics, while Table 5 shows the results of both t-tests:

Table 4: Group Statistics for each Classification Group and Their Familiarity with Agile and Waterfall

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>Mean</th>
<th>Std. Deviation</th>
<th>Std. Error mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agile</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sophomore or Junior</td>
<td>69</td>
<td>1.72</td>
<td>1.042</td>
<td>.125</td>
</tr>
<tr>
<td>Senior or Graduate</td>
<td>150</td>
<td>3.45</td>
<td>1.133</td>
<td>.092</td>
</tr>
<tr>
<td>Waterfall</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sophomore or Junior</td>
<td>69</td>
<td>1.58</td>
<td>.847</td>
<td>.102</td>
</tr>
<tr>
<td>Senior or Graduate</td>
<td>150</td>
<td>3.17</td>
<td>1.104</td>
<td>.090</td>
</tr>
</tbody>
</table>

Table 5: Independent Samples t-Tests for Student Familiarity with Agile and Waterfall Based on Classification

<table>
<thead>
<tr>
<th>Lifecycle</th>
<th>t</th>
<th>df</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agile</td>
<td>-10.755</td>
<td>217</td>
<td>&lt;.001</td>
</tr>
<tr>
<td>Waterfall</td>
<td>-10.634</td>
<td>217</td>
<td>&lt;.001</td>
</tr>
</tbody>
</table>

The independent t-test for Agile found that the 69 Sophomores and Juniors (M = 1.72, SD = 1.042) compared to the 150 Seniors and Graduates (M = 3.45, SD = 0.092) demonstrated significantly lower familiarity scores with Agile, t(217) = -10.755, p = < 0.001. Similarly, the independent t-test for Waterfall found that the Sophomores and Juniors (M = 1.58, SD = 0.847) compared to the Seniors and Graduates (M = 3.17, SD = 1.133) had significantly lower familiarity scores with Waterfall, t(217) = -10.634, p < 0.001. Thus, it can be interpreted that the higher the classification, the more familiarity with Agile and Waterfall the student will most likely have.
**Question 5:**

"Which of the following areas/skills do you believe you struggle the most in when working in computer science projects?"

This question allowed students to select all the areas/skills they believed they struggled the most in. A list of 6 different areas/skills – all corresponding to the areas/skills of identified knowledge deficiencies from [7] – were provided, along with an “Other” option and an “I don’t struggle in any of these areas/skills” (shortened to “None” in Figure 31) exclusive option (exclusive refers to an option that once selected, is the only option that can be selected). The responses for all students are shown in Figure 31.

![Bar Chart](chart.png)

**Fig. 31: Areas/Skills Students Believe Themselves to Struggle the Most In**

Based on this data, the 3 topmost skills (in order from most to least) with which students believed themselves to struggle the most in were testing, written communication, and software tools.
ii. Discovery Survey 2

This survey had a total of 158 responses. However, since only responses that were fully completed were considered in the data, this reduced the number to 133 completed responses. From these 133 responses, the data was checked to ensure no duplicate responses. Figure 32 shows the classification of all students when they partook in the survey (Spring 2023). The majority of respondents (63%) were Seniors. There were no Freshmen students. The rest of this section will breakdown a subset of questions followed by a discussion and analysis on the responses.

![Student Classification Pie Chart]

**Fig. 32: Student Classifications from Discovery Survey 2**

*Question 1:*

“In your college experience so far within CS courses, have you worked more independently on projects, or in teams on projects?”

Students were given 4 options to choose from for this question: mostly independently, equally for both, mostly in teams, and unsure. As such, responses that selected one of the first 3 options were qualified for this data, yielding a total of 132 responses. Descriptive statistics of the data showed a relationship between the student’s classification and their experiences working on
teams or independently on projects. As such, a relationship map was created to visualize this relationship (see Figure 33).

From Figure 33, a thicker line indicates a higher relationship count between the variables, with a thinner line representing lower relationship counts between the variables. As such, of the 3 lines originating from “Senior or Graduate”, the thickest (the strongest relationship) goes to “independent and in teams”. On the other hand, of the two lines originating from “Sophomore or Junior”, there is a very thin line (indicating a weak relationship) that goes to “independent and in teams”, and the strongest line goes to “mostly independent”. There is no line between “Sophomore or Junior” and “mostly in teams”, meaning that no relationship between the 2 groups exists.

Fig. 33: Relationship Map Between a Student’s Classification and their Experience Working in Teams or Independently on Projects

---

84
There is seemingly an equal number of Seniors and graduates as there are Sophomores and Juniors who have worked equally independently and in teams on projects. For Seniors and graduates, there is a relationship with having worked mostly in teams, as opposed to Sophomore and Juniors. As such, from Figure 33, it can be observed that there is a connection between a student’s classification and their experience working on teams or independently on projects.

**Question 2:**

“Generally, which CS classes have assigned more group work/projects over individual work/projects?”

For this question, students were given the options between Freshmen, Sophomore, Junior, Senior, and graduate level computer science courses, in addition to 3 more options, one for “all classes have assigned an equal amount of group work/projects”, another for “my classes have assigned more individual work than group work”, and an “unsure” option. To qualify the data, responses for each classification were considered, yielding 97 responses, for which the raw data is shown in Figure 34.

![Student Responses of CS Classes that have Assigned More Group Work/Projects](image)

Fig. 34: Student Responses of CS Classes that have Assigned More Group Work/Projects
Given that this question was answered by students from Sophomore to graduate levels, the data was filtered to only consider Senior and graduate responses, who have experienced all class levels from Freshmen to Senior. A comparison between the means of the two groups was carried out. For this, numerical values were assigned to each of the listed options from a range of 1 – 5, with Freshmen Classes equaling 1, going up to Graduate Classes equaling 5. Table 6 shows the results of comparing the means between the 2 groups:

Table 6: Comparison of Means Between the two Classification Groups of Students for Which Class Assigns More Group Work

<table>
<thead>
<tr>
<th>Groups</th>
<th>Mean</th>
<th>N</th>
<th>Std. Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sophomores and Juniors</td>
<td>2.49</td>
<td>41</td>
<td>0.925</td>
</tr>
<tr>
<td>Seniors and graduates</td>
<td>3.86</td>
<td>56</td>
<td>0.645</td>
</tr>
</tbody>
</table>

An independent t-test was performed to obtain statistical significance in dividing the data this way. Levene’s Test for Equality of Variances suggested that there was strong evidence to reject the null hypothesis of equal variances within the data. As such, Welch’s t-test was performed on the data, which does not assume equal variances. Table 7 shows the group statistics for the t-test, while Table 8 shows the results of the t-test:

Table 7: Group Statistics for the two Classification Groups of Students for Which Class Assigns More Group Work

<table>
<thead>
<tr>
<th>Type of Work</th>
<th>N</th>
<th>Mean</th>
<th>Std. Deviation</th>
<th>Std. Error mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sophomore or Junior</td>
<td>41</td>
<td>2.49</td>
<td>.925</td>
<td>.145</td>
</tr>
<tr>
<td>Senior or Graduate</td>
<td>56</td>
<td>3.86</td>
<td>.645</td>
<td>.086</td>
</tr>
</tbody>
</table>

Table 8: Independent Samples t-Tests for the two Classification Groups of Students for Which Class Assigns More Group Work

<table>
<thead>
<tr>
<th>Independent Samples Test</th>
<th>t</th>
<th>df</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of work</td>
<td>-8.140</td>
<td>67.303</td>
<td>&lt;.001</td>
</tr>
</tbody>
</table>
The independent t-test found that the 41 Sophomores and Juniors (M = 2.49, SD = 0.145) compared to the 56 Seniors and Graduates (M = 3.86, SD = 0.086) demonstrated significantly lower classification scores (as in, they selected classes of lower classification), \( t(67.303) = -8.140, p < 0.001 \). Given these results, there is statistical significance in the correlation between the student’s classification and their selection of the classification of the class with most teamwork. Therefore, given that Seniors and graduates have experienced all undergraduate courses, it makes sense to filter the data to only consider their responses. As such, the classification with the classes that assign the most group work are Senior level courses.

**Question 3:**

“How familiar are you with self-organized teams?”

This question utilized a 5-point Likert scale for students to rate their familiarity with self-organized teams. Figure 37 shows the raw data of the responses to this question.

![Student Familiarity with Self-Organized Teams](image)

Fig. 37: Student Familiarity with Self-Organized Teams
Given that almost 33% of respondents were taking CS4381/5381 (an Agile Software Development course) at the time, an independent samples t-test was performed on the data in order to see if there was any statistical significance on the student’s enrollment in the Agile course and their familiarity with self-organized teams. The data was normally distributed and passed Levene’s Test for Equality of Variances, meaning that there was homogeneity of variances. The t-test had a 95% confidence interval for the mean difference. Table 9 shows the group statistics, while Table 10 shows the results of both t-tests:

Table 9: Group Statistics for Student Familiarity with Self-Organized Teams Based on Current Enrollment in Agile Class

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>Mean</th>
<th>Std. Deviation</th>
<th>Std. Error mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Organized Teams Familiarity</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not in Agile Class</td>
<td>90</td>
<td>2.42</td>
<td>1.298</td>
<td>.137</td>
</tr>
<tr>
<td>In Agile Class</td>
<td>43</td>
<td>3.77</td>
<td>.996</td>
<td>.152</td>
</tr>
</tbody>
</table>

Table 10: Independent Samples t-Tests for Student Familiarity with Self-Organized Teams Based on Current Enrollment in Agile Class

<table>
<thead>
<tr>
<th></th>
<th>t</th>
<th>df</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Organized Teams Familiarity</td>
<td>-6.000</td>
<td>131</td>
<td>&lt;.001</td>
</tr>
</tbody>
</table>

The independent t-test for familiarity with self-organized teams found that the 90 students not in the Agile class (M = 2.42, SD = 1.298) compared to the 43 students in the Agile class (M = 3.77, SD = 0.996) demonstrated significantly lower familiarity scores with self-organized teams, t(131) = -6.000, p = < 0.001. In order to visualize the difference, Figure 38, a relationship map between the groups and the familiarities, was created.
Figure 38 shows that there is a stronger relationship between the students who were in the Agile class and their familiarity with self-organized teams than those who were not in the Agile class. The students who were not in the Agile class had a stronger relationship with lower familiarity with self-organized teams. As such, student enrollment in the Agile class affects their familiarity with self-organized teams.

**Question 4:**

“If you answered ‘Yes’ to the previous question, how was your experience working on that self-organized team?”

This question was a follow-up question to a question that asked if students had ever been on a self-organized team for a team project. A 7-point Likert scale ranging from Very Negative
(1) to Very Positive (7) was used. The data was validated to ensure that only those who responded “Yes” to the previous question were considered in this response, qualifying 66 responses. Figure 39 shows the raw data results of this question. Based on this data, the majority of respondents have had positive experiences on self-organized teams.

![Experience Working on Self-Organized Team](image)

**Fig. 39: Student Experiences Working on Self-Organized Teams**

Descriptive statistics for this data showed a mean of 2.39, which would fall between Mostly Positive (2) and Somewhat Positive (3). Table 11 shows the full descriptive statistics for the data.

<table>
<thead>
<tr>
<th>Experience with Self-Organized Teams</th>
<th>N</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Mean</th>
<th>Std. Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experience with Self-Organized Teams</td>
<td>65</td>
<td>3</td>
<td>7</td>
<td>5.62</td>
<td>.963</td>
</tr>
</tbody>
</table>

Table 11: Descriptive Statistics for Student Experience with Self-Organized Teams
iii. Teamwork, Learning Processes, and Lifecycles Survey

This survey had a total of 265 responses, with 229 completed responses. From these 229 responses, the data was validated to ensure no duplicate responses. Figure 40 shows the classification of all students when they participated in the survey. The majority of respondents (75%) were Seniors. There were no Freshmen students. The rest of this section will breakdown a subset of questions followed by a discussion and analysis on the responses.

![CLASSIFICATION](image)

Fig. 40: Student Classifications from Teamwork, Learning Processes, and Lifecycles Survey

**Question 1 and Question 2:**

“Answer this question for the Agile Software Development (CS4381/5381) course you took.

What was your experience working on an Agile team in that course? Please rate the following based on whether you agree or disagree with the statement.”, and

“What has been your experience working on a team (of at least 3 members) for a software development project that lasted longer than a week within a CS class? Please rate the following based on whether you agree or disagree with the statement.”
In order to understand the experiences of students working in teams within and outside an Agile course, Question 1 and Question 2 were asked using the Team Effectiveness Questionnaire from [71]. Each of these questions had different ways of qualifying the data, described below:

- **Question 1:** This question only appeared to students who had, in the demographics blog, selected that they had taken the Agile Software Development (CS4381/5381) course. There were a total of 31 students that responded this question.

- **Question 2:** The data for this question was qualified by first asking students if they had ever participated in a course in which they completed a software development project that lasted longer than a week within a CS class. Students who had taken the Agile course saw a different version of this question that clarified that this refers to a course other than the Agile course. All students who responded “Yes” to these qualifying questions were shown Question 2. As such, this question was answered by a total of 178 students.

The Team Effectiveness Questionnaire provided a total of 23 statements that students had to rate using a 5-points Likert scale ranging from “Strongly Disagree” (1) to “Strongly Agree” (5). For all but one statement of the questionnaire, the higher rating indicated a positive connotation, while the lower rating indicated a negative connotation. The statement for which this was not the case (and the higher rating indicated a negative connotation with the lower rating indicating a positive connotation) was the sixth statement on the questionnaire: “It was difficult to ask members of the team for help”.

In order to be able to run independent samples t-test on the data, 147 responses to Question 2 were qualified, such that the 31 students who responded Question 1 were compared to 147 different students who responded Question 2. Given the vast difference in sample size from each group (31 in Question 1 and 147 in Question 2), Welch’s t-test was performed on the
data which does not assume equal variances in the data. The t-tests were performed on each statement of the Team Effectiveness Questionnaire, yielding a total of 23 separate comparisons. Given the length of the statements, each statement was labeled with a “Q” and a number. Table 12 below shows the label that corresponds to each statement. The group statistics for each statement are shown in Table 13, and Table 14 shows the results from running the t-tests on the data.
Table 12: Labels Assigned to each Statement of the Team Effectiveness Questionnaire

<table>
<thead>
<tr>
<th>Label</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Team members were committed to a common purpose</td>
</tr>
<tr>
<td>Q2</td>
<td>I understood what my job entailed within the team</td>
</tr>
<tr>
<td>Q3</td>
<td>What I thought I would be doing and what I actually did in the team were the same</td>
</tr>
<tr>
<td>Q4</td>
<td>Team purpose and team goals were related</td>
</tr>
<tr>
<td>Q5</td>
<td>I believe I was a team player</td>
</tr>
<tr>
<td>Q6</td>
<td>It was difficult to ask members of the team for help</td>
</tr>
<tr>
<td>Q7</td>
<td>The team's performance exceeded our expectations</td>
</tr>
<tr>
<td>Q8</td>
<td>Disagreements were accepted and encouraged in the team</td>
</tr>
<tr>
<td>Q9</td>
<td>Accomplishing small goals helped the team stay focused</td>
</tr>
<tr>
<td>Q10</td>
<td>The team goals emerged through team interactions</td>
</tr>
<tr>
<td>Q11</td>
<td>I could effectively communicate my idea in the team</td>
</tr>
<tr>
<td>Q12</td>
<td>Our team meetings were productive</td>
</tr>
<tr>
<td>Q13</td>
<td>We motivated each other when things were difficult</td>
</tr>
<tr>
<td>Q14</td>
<td>Team goals were challenging</td>
</tr>
<tr>
<td>Q15</td>
<td>I clearly understood team goals</td>
</tr>
<tr>
<td>Q16</td>
<td>I clearly understood what was my role in the team</td>
</tr>
<tr>
<td>Q17</td>
<td>I did need help from other people in the team</td>
</tr>
<tr>
<td>Q18</td>
<td>Team members felt ownership of the team purpose</td>
</tr>
<tr>
<td>Q19</td>
<td>My experiences in the team will contribute to my career success</td>
</tr>
<tr>
<td>Q20</td>
<td>I understood clearly what other members' duties were in the team</td>
</tr>
<tr>
<td>Q21</td>
<td>I received valuable feedback from the team</td>
</tr>
<tr>
<td>Q22</td>
<td>I could trust in the other team members to do their part of the job</td>
</tr>
<tr>
<td>Q23</td>
<td>The team was able to resolve differences</td>
</tr>
</tbody>
</table>
Table 13: Groups Statistics for Each Student Group for the Team Effectiveness Questionnaires

<table>
<thead>
<tr>
<th>Statement</th>
<th>N</th>
<th>Mean</th>
<th>Std. Deviation</th>
<th>Std. Error Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>147</td>
<td>3.84</td>
<td>.973</td>
<td>.080</td>
</tr>
<tr>
<td>Q2</td>
<td>147</td>
<td>3.56</td>
<td>.929</td>
<td>.077</td>
</tr>
<tr>
<td>Q3</td>
<td>147</td>
<td>3.97</td>
<td>.802</td>
<td>.066</td>
</tr>
<tr>
<td>Q4</td>
<td>147</td>
<td>4.68</td>
<td>.541</td>
<td>.097</td>
</tr>
<tr>
<td>Q5</td>
<td>147</td>
<td>4.24</td>
<td>.841</td>
<td>.099</td>
</tr>
<tr>
<td>Q6</td>
<td>147</td>
<td>2.64</td>
<td>1.205</td>
<td>.099</td>
</tr>
<tr>
<td>Q7</td>
<td>147</td>
<td>3.39</td>
<td>.969</td>
<td>.080</td>
</tr>
<tr>
<td>Q8</td>
<td>147</td>
<td>4.05</td>
<td>.834</td>
<td>.069</td>
</tr>
<tr>
<td>Q9</td>
<td>147</td>
<td>3.90</td>
<td>.920</td>
<td>.076</td>
</tr>
<tr>
<td>Q10</td>
<td>147</td>
<td>3.90</td>
<td>.797</td>
<td>.066</td>
</tr>
<tr>
<td>Q11</td>
<td>147</td>
<td>4.05</td>
<td>.824</td>
<td>.120</td>
</tr>
<tr>
<td>Q12</td>
<td>147</td>
<td>3.81</td>
<td>.946</td>
<td>.123</td>
</tr>
<tr>
<td>Q13</td>
<td>147</td>
<td>3.66</td>
<td>.910</td>
<td>.163</td>
</tr>
<tr>
<td>Q14</td>
<td>147</td>
<td>3.60</td>
<td>.948</td>
<td>.080</td>
</tr>
<tr>
<td>Q15</td>
<td>147</td>
<td>3.66</td>
<td>.744</td>
<td>.101</td>
</tr>
<tr>
<td>Q16</td>
<td>147</td>
<td>4.10</td>
<td>.783</td>
<td>.065</td>
</tr>
<tr>
<td>Q17</td>
<td>147</td>
<td>3.62</td>
<td>.884</td>
<td>.159</td>
</tr>
<tr>
<td>Q18</td>
<td>147</td>
<td>3.55</td>
<td>.722</td>
<td>.238</td>
</tr>
<tr>
<td>Q19</td>
<td>147</td>
<td>3.95</td>
<td>.927</td>
<td>.101</td>
</tr>
<tr>
<td>Q20</td>
<td>147</td>
<td>3.89</td>
<td>.908</td>
<td>.075</td>
</tr>
<tr>
<td>Q21</td>
<td>147</td>
<td>3.73</td>
<td>.709</td>
<td>.127</td>
</tr>
<tr>
<td>Q22</td>
<td>147</td>
<td>3.73</td>
<td>1.055</td>
<td>.090</td>
</tr>
<tr>
<td>Q23</td>
<td>147</td>
<td>4.05</td>
<td>.844</td>
<td>.152</td>
</tr>
</tbody>
</table>
Based on these results, it can be observed that for some statements, there is a statistically significant difference between the students who answered the Team Effectiveness Questionnaire for the Agile course and those who answered it for a non-Agile course. The statements for which there was a significant difference are highlighted in Table 14 in green, and those in which there was not a significant difference are highlighted in blue. For most of the statements, there was a statistically significant difference in rating scores between the students in Question 1 and those in Question 2. For example, Q1, which is “Team members were committed to a common purpose”,

<table>
<thead>
<tr>
<th>Statement No.</th>
<th>t</th>
<th>df</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>-3.565</td>
<td>48.094</td>
<td>&lt;.001</td>
</tr>
<tr>
<td>Q2</td>
<td>-2.219</td>
<td>50.969</td>
<td>.031</td>
</tr>
<tr>
<td>Q3</td>
<td>-3.516</td>
<td>42.803</td>
<td>.001</td>
</tr>
<tr>
<td>Q4</td>
<td>-5.996</td>
<td>61.552</td>
<td>&lt;.001</td>
</tr>
<tr>
<td>Q5</td>
<td>-3.314</td>
<td>63.582</td>
<td>.002</td>
</tr>
<tr>
<td>Q6</td>
<td>1.225</td>
<td>38.720</td>
<td>.228</td>
</tr>
<tr>
<td>Q7</td>
<td>-2.851</td>
<td>40.493</td>
<td>.007</td>
</tr>
<tr>
<td>Q8</td>
<td>-3.564</td>
<td>53.500</td>
<td>&lt;.001</td>
</tr>
<tr>
<td>Q9</td>
<td>-4.560</td>
<td>60.967</td>
<td>&lt;.001</td>
</tr>
<tr>
<td>Q10</td>
<td>-3.973</td>
<td>46.633</td>
<td>&lt;.001</td>
</tr>
<tr>
<td>Q11</td>
<td>-3.021</td>
<td>55.261</td>
<td>.004</td>
</tr>
<tr>
<td>Q12</td>
<td>-2.121</td>
<td>44.758</td>
<td>.040</td>
</tr>
<tr>
<td>Q13</td>
<td>-3.185</td>
<td>46.522</td>
<td>.003</td>
</tr>
<tr>
<td>Q14</td>
<td>-1.970</td>
<td>43.597</td>
<td>.055</td>
</tr>
<tr>
<td>Q15</td>
<td>-1.954</td>
<td>43.447</td>
<td>.057</td>
</tr>
<tr>
<td>Q16</td>
<td>-1.557</td>
<td>47.143</td>
<td>.126</td>
</tr>
<tr>
<td>Q17</td>
<td>-.230</td>
<td>38.732</td>
<td>.820</td>
</tr>
<tr>
<td>Q18</td>
<td>-3.716</td>
<td>43.210</td>
<td>&lt;.001</td>
</tr>
<tr>
<td>Q19</td>
<td>-4.699</td>
<td>62.096</td>
<td>&lt;.001</td>
</tr>
<tr>
<td>Q20</td>
<td>-3.138</td>
<td>52.988</td>
<td>.003</td>
</tr>
<tr>
<td>Q21</td>
<td>-4.053</td>
<td>62.299</td>
<td>&lt;.001</td>
</tr>
<tr>
<td>Q22</td>
<td>-2.339</td>
<td>44.656</td>
<td>.024</td>
</tr>
<tr>
<td>Q23</td>
<td>-2.013</td>
<td>42.161</td>
<td>.051</td>
</tr>
</tbody>
</table>
had statistically significant higher scores for the Agile students (M = 4.45, SD = 0.850),
compared to the non-Agile students (M = 3.84, SD = 0.973), with t(48.094) = -3.565, p < 0.001.

On the other hand, statements such as Q14, which is “Team goals were challenging”, did not have statistically significant difference in rating scores between the Agile students (M = 3.97, SD = 0.948), as compared to the non-Agile students (M = 3.60, SD = 0.948), with t(43.597) = -1.970, p =0.55.

Based on these results, it can be said that there is a statistical significance in students’ experiences on teams within an Agile course as compared to a non-Agile course related to certain aspects and parts of the team experiences. In certain aspects, there is not a statistically significant difference in comparison.

**Question 3:**

“How well did the agile process from the Agile Software Development (CS4381/5381) course help you build the following skills or knowledge areas?”

Similar to Question 1, this question was only asked to students who had taken the Agile course. This question presented all of the identified areas of knowledge deficiencies from [7] and asked students, using a scale from “Not at all” (1) to “Very well” (4), to rate how well the agile process from the ASD course helped them build these areas. A “N/A” option was also provided. In order to qualify the data, only the responses from the Likert-scale were considered, and the mean results of each are shown in Figures 41 below. The full question provided examples for each knowledge area that was presented. For the purposes of Figures 41, these were condensed to just show the title of the knowledge area. Table 15 below the figure provides descriptive statistics.
Fig. 41: Means of How Well Students Believe the Agile Course Helped Them Build the Knowledge Areas

Table 15: Descriptive Statistics for the Knowledge Areas

<table>
<thead>
<tr>
<th>Knowledge Area</th>
<th>N</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Mean</th>
<th>Std. Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Written Communication</td>
<td>29</td>
<td>1</td>
<td>4</td>
<td>3.10</td>
<td>.817</td>
</tr>
<tr>
<td>Oral Communication</td>
<td>31</td>
<td>2</td>
<td>4</td>
<td>3.55</td>
<td>.624</td>
</tr>
<tr>
<td>Project Management</td>
<td>31</td>
<td>2</td>
<td>4</td>
<td>3.58</td>
<td>.564</td>
</tr>
<tr>
<td>Software Tools</td>
<td>31</td>
<td>1</td>
<td>4</td>
<td>3.45</td>
<td>.723</td>
</tr>
<tr>
<td>Testing</td>
<td>31</td>
<td>1</td>
<td>4</td>
<td>3.00</td>
<td>.931</td>
</tr>
<tr>
<td>Teamwork</td>
<td>30</td>
<td>2</td>
<td>4</td>
<td>3.70</td>
<td>.535</td>
</tr>
<tr>
<td>Problem Solving</td>
<td>30</td>
<td>2</td>
<td>4</td>
<td>3.63</td>
<td>.615</td>
</tr>
<tr>
<td>Programming</td>
<td>29</td>
<td>2</td>
<td>4</td>
<td>3.55</td>
<td>.572</td>
</tr>
<tr>
<td>Leadership</td>
<td>30</td>
<td>2</td>
<td>4</td>
<td>3.57</td>
<td>.626</td>
</tr>
<tr>
<td>Design</td>
<td>31</td>
<td>1</td>
<td>4</td>
<td>3.29</td>
<td>.739</td>
</tr>
<tr>
<td>Databases</td>
<td>31</td>
<td>1</td>
<td>4</td>
<td>3.42</td>
<td>.765</td>
</tr>
</tbody>
</table>
From Figure 41 and Table 15, we can observe that overall, the majority of students responded between “Well” (3) and “Very Well” (4) for the agile process from the ASD course helping them build the presented knowledge areas, with the lowest mean being a 3 belonging to testing. Teamwork (M = 3.70, SD = 0.535), problem solving (M = 3.63, SD = 0.615), and project management (M = 3.58, SD = 0.564) had the highest means. Thus, it can be concluded that the Agile course overall had a very positive effect on students’ perceptions of how well the course helped them build the presented knowledge areas.

**Question 4:**

“Please rate the following class activities based on how helpful you find them towards your learning during a class. If you’ve never experienced one of the activities, select ‘N/A’.”

For this question, students were presented a list of different in-class activities and asked to rate them based on a 5-point Likert scale ranging from “Not at all Helpful” (1) to “Extremely Helpful” (5). An “N/A” option was also available for students to select if they had never experienced one of the activities. All of the activities provided on Question 4 can be mapped directly to examples provided by [26] for the DOLA framework. As such, Table 16 below shows the mappings of each activity that was shown to students in Question 4 to the type of learning that it falls under according to the DOLA framework.
Table 16: Examples of Each Learning Model

<table>
<thead>
<tr>
<th>Learning Model</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passive Learning</td>
<td>Watching the professor’s presentation (such as slides, etc.)</td>
</tr>
<tr>
<td></td>
<td>Listening to the professor</td>
</tr>
<tr>
<td></td>
<td>Reading the textbook</td>
</tr>
<tr>
<td></td>
<td>Watching a video</td>
</tr>
<tr>
<td>Active Learning</td>
<td>Underlining/Highlighting Notes</td>
</tr>
<tr>
<td></td>
<td>Copying the solution of a problem from the board while the professor is solving it</td>
</tr>
<tr>
<td>Constructive Learning</td>
<td>Drawing a concept map</td>
</tr>
<tr>
<td></td>
<td>Writing notes</td>
</tr>
<tr>
<td></td>
<td>Asking comprehensive questions</td>
</tr>
<tr>
<td></td>
<td>Drawing/Interpreting maps</td>
</tr>
<tr>
<td>Interactive Learning</td>
<td>Working in pairs/groups</td>
</tr>
<tr>
<td></td>
<td>Getting feedback from the professor</td>
</tr>
<tr>
<td></td>
<td>Debating a topic by using evidence</td>
</tr>
</tbody>
</table>

Using the mappings from this table, Figure 42 shows the mean results of this question, categorizing the activities by the type of learning that they fall under. Table 17 below the figure shows descriptive statistics for this data.

![Means of How Helpful Students Find Certain Activities Towards their Learning](chart)

**Fig. 42: Means of How Helpful Student Find Certain Activities Towards their Learning**
From Figure 42 and Table 17, it can be observed that in general, activities belonging to active learning, constructivist learning, and interactive learning have higher means, with the lowest mean in all 3 categories being a 3.25, belonging to underlining/highlighting notes. The activity with the lowest mean belongs to passive learning: reading from the textbook (M = 2.87, SD = 1.161). The 3 activities with the highest means were: getting feedback from the professor (M = 4.27, SD = 0.798), asking comprehensive questions (M = 4.04, SD = 0.870), and writing notes (M = 3.77, SD = 1.087), which tied for third place with listening to the professor (M = 3.77, SD = 1.012).

**Question 5:**

"Thinking about the Computer Science courses you have taken and are currently taking, please rate your level of agreement or disagreement with the following statements. Note that the word
“coursework” refers to anything that you did in these courses, including assignments, activities, readings, etc."

This question utilized the MUSIC Inventory survey from [72]. In order to test the MUSIC Inventory model, which hypothesizes casual relationships between the observed variables and the factors, a Confirmatory Factor Analysis (CFA) was performed on the raw data using IBM AMOS. There are 5 factors: “eMpowerment”, “Usefulness”, “Success”, “Interest”, and “Caring”, and 19 observed variables, one for each of the items from the MUSIC Inventory. The data was pre-processed in order to fit the model, and the factors and variables were provided for the model specifications in order to run CFA. Figure 43 shows the model created in IBM AMOS used to run CFA. Table 15 shows the standardized regression weight (or factor loadings) of the model.
Table 18: Standardized Regression Weights (Factor Loadings) of the Confirmatory Factor Analysis for the MUSIC Model

<table>
<thead>
<tr>
<th>Item</th>
<th>Factor Loading Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item 15</td>
<td>Success</td>
</tr>
<tr>
<td>Item 11</td>
<td>Success</td>
</tr>
<tr>
<td>Item 7</td>
<td>Success</td>
</tr>
<tr>
<td>Item 2</td>
<td>Success</td>
</tr>
<tr>
<td>Item 12</td>
<td>Interest</td>
</tr>
<tr>
<td>Item 6</td>
<td>Interest</td>
</tr>
<tr>
<td>Item 3</td>
<td>Interest</td>
</tr>
<tr>
<td>Item 17</td>
<td>Caring</td>
</tr>
<tr>
<td>Item 13</td>
<td>Caring</td>
</tr>
<tr>
<td>Item 8</td>
<td>Caring</td>
</tr>
<tr>
<td>Item 4</td>
<td>Caring</td>
</tr>
<tr>
<td>Item 19</td>
<td>eMpowerment</td>
</tr>
<tr>
<td>Item 14</td>
<td>eMpowerment</td>
</tr>
<tr>
<td>Item 9</td>
<td>eMpowerment</td>
</tr>
<tr>
<td>Item 5</td>
<td>eMpowerment</td>
</tr>
<tr>
<td>Item 18</td>
<td>Usefulness</td>
</tr>
<tr>
<td>Item 16</td>
<td>Usefulness</td>
</tr>
<tr>
<td>Item 10</td>
<td>Usefulness</td>
</tr>
<tr>
<td>Item 1</td>
<td>Usefulness</td>
</tr>
</tbody>
</table>

The CFA revealed that all the loadings range from 0.646 to 0.918, indicating a strong relationship between the latent factor and the observed variable. As such, using the guide provided for the MUSIC model, Figure 44 was created to show the results for each factor of the MUSIC model based on the student responses.
Based on these results, it can be observed that there are somewhat high levels of empowerment, usefulness, success, interest, and caring within student’s motivation in computer science coursework, with the lowest mean score belonging to interest (4.38), and the highest score belonging to caring (4.91).

**Questions 6 & 7:**

“In your CS education so far, how many of your CS classes have used Agile Software Development as the lifecycles for a software development project? Select your best estimate.”, and “In your CS education so far, how many of your CS classes have used Waterfall as the lifecycle for a software development project? Select your best estimate.”

Students were asked these questions in order to understand how many computer science classes use Agile and Waterfall. Students were given options ranging from 0 to 5, more than 5, and a decline to respond option. The responses that included a numerical value for the number of
classes were qualified for this data. Table 19 shows the descriptive statistics for both of these questions.

Table 19: Descriptive Statistics for the Number of Classes that have Used Agile and Waterfall

<table>
<thead>
<tr>
<th>Descriptive Statistics</th>
<th>N</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Mean</th>
<th>Std. Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of classes that have used Agile</td>
<td>211</td>
<td>0</td>
<td>6</td>
<td>1.75</td>
<td>1.386</td>
</tr>
<tr>
<td>Number of classes that have used Waterfall</td>
<td>226</td>
<td>0</td>
<td>6</td>
<td>1.49</td>
<td>1.446</td>
</tr>
</tbody>
</table>

Based on Table 19, most students responses between 1-2 for both Agile (M = 1.75, SD = 1.386) and Waterfall (M = 1.49, SD = 1.446).

Questions 8 & 9:

“How would you rate your experience working on a project within a CS course where you used the Agile Software Development lifecycle?”, and “How would you rate your experience working on a project within a CS course where you used the Waterfall lifecycle?”

In order to assess students’ experiences in courses that have used Agile and Waterfall, Questions 8 and 9 were asked, using a 7-point Likert scale ranging from “Very Negative” (1) to “Very Positive” (7). The data for these questions was qualified by ensuring that respondents answered at least “1” in Questions 6 and 7 respectively, such that students have taken at least one course in Agile and Waterfall. A total of 169 were qualified for Question 8, and a total of 158 were qualified for Question 9. Table 20 shows the descriptive statistics for both questions.

Table 20: Descriptive Statistics for Student Experiences in Agile Courses and Waterfall Courses

<table>
<thead>
<tr>
<th>Descriptive Statistics</th>
<th>N</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Mean</th>
<th>Std. Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agile Experience</td>
<td>169</td>
<td>3</td>
<td>7</td>
<td>5.68</td>
<td>1.037</td>
</tr>
<tr>
<td>Waterfall Experience</td>
<td>158</td>
<td>1</td>
<td>7</td>
<td>5.06</td>
<td>1.166</td>
</tr>
</tbody>
</table>
In order to compare student experiences in who have partaken in both types of courses, a paired sample t-test was carried out on the data. To qualify the data for the paired test, only students who responded having experiences in both Agile and Waterfall courses were considered, yielding a total of 139 responses that were used in this comparison. Table 21 shows the paired samples statistics of this comparison, and Table 22 shows the results of the paired samples t-test:

Table 21: Paired Samples Statistics for Students Who Have Experienced Agile and Waterfall Courses

<table>
<thead>
<tr>
<th>Paired Samples Statistics</th>
<th>Mean</th>
<th>N</th>
<th>Std. Deviation</th>
<th>Std. Error Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agile Experience</td>
<td>5.72</td>
<td>139</td>
<td>1.029</td>
<td>.087</td>
</tr>
<tr>
<td>Waterfall Experience</td>
<td>5.05</td>
<td>139</td>
<td>1.188</td>
<td>.101</td>
</tr>
</tbody>
</table>

Table 22: Paired Samples t-Test for Students Who Have Experienced Agile and Waterfall Courses

<table>
<thead>
<tr>
<th>Paired Samples Test</th>
<th>t</th>
<th>df</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agile Experience - Waterfall Experience</td>
<td>6.074</td>
<td>138</td>
<td>&lt;0.001</td>
</tr>
</tbody>
</table>

Based on the results from the paired samples t-test, it can be observed that there is statistical significance in the Agile experience scores (M = 5.72, SD = 1.029) as compared to the Waterfall experience scores (M = 5.05, SD = 1.188), with t(138) = 6.074, p < 0.001. Thus, the Agile experience scores were significantly higher than those the Waterfall experience scores.
3. **PROPOSED MODEL**

This section will introduce the new educational software development lifecycle. This model is intended for use in Senior-level courses, based on the results which showed that students generally practice more group work in higher level courses (see Figures 33 and 34). The model should be complemented with small lectures that introduce new topics and theory as necessary (such as the Agile Principles and Values, or any necessary software tools). Ideally, lectures should be kept short, and most of the class meeting times should be used to allow students time to work on their project, which is the central aspect of the course. The following subsections describe important components necessary in the creation of this model.

i. **Mapping of Agile Principles to Educational Principles**

To propose a new model meant to be integrated into the classroom, it was important to first understand how Agile principles can be mapped directly to the educational principles of the engineering discipline. As such, a mapping of each Agile principle to a student outcome from Criterion 3 of the Accreditation Board for Engineering and Technology (ABET) from [74] was created. The ABET Student Outcomes Criteria for Baccalaureate Level Engineering Programs are listed below [74]:

1. an ability to identify, formulate, and solve complex engineering problems by applying principles of engineering, science, and mathematics.

2. an ability to apply engineering design to produce solutions that meet specified needs with consideration of public health, safety, and welfare, as well as global, cultural, social, environmental, and economic factors.

3. an ability to communicate effectively with a range of audiences.
4. an ability to recognize ethical and professional responsibilities in engineering situations and make informed judgments, which must consider the impact of engineering solutions in global, economic, environmental, and societal contexts.

5. an ability to function effectively on a team whose members together provide leadership, create a collaborative and inclusive environment, establish goals, plan tasks, and meet objectives.

6. an ability to develop and conduct appropriate experimentation, analyze and interpret data, and use engineering judgment to draw conclusions.

7. an ability to acquire and apply new knowledge as needed, using appropriate learning strategies.

In addition, another mapping was created between each Agile principle to one or more of the Characteristics of Computer Science Graduates according to [75] from The Joint Task Force on Computing Curricula, made up of 3 organizations: the Association for Computing Machinery (ACM), the Institute of Electrical and Electronics Engineers-Computer Society (IEEE-CS), and the Association for Advancement of Artificial Intelligence (AAAI). These characteristics are listed below [75]:

A. Algorithmic problem-solver - Good solutions to common problems at an appropriate level of abstraction

B. Competent programmer

C. In possession of mental model of computation - deep learning

D. Life-long learner

E. Collaborative

F. Socially responsible - ethical behavior
G. Global and cultural competence

H. Cross-disciplinary - understanding of non-computing disciplines

I. Adversarial thinker/Computational thinker

J. Think at multiple levels of abstraction.

K. Adaptable

L. Handle ambiguity and uncertainty

M. Analytical and problem-solving skills

N. Knowledge of algorithms and data structures

O. Familiarity with software engineering principles

P. Strong mathematical and logical skills

Q. Effective communication skills

Table 23 serves the purposes of showing how the Agile principles can be mapped directly to educational outcomes of engineering students. All of the mappings can be seen in Table 23 below:

<table>
<thead>
<tr>
<th>Agile Principle</th>
<th>ABET Student Outcome</th>
<th>Characteristics of Computer Science Graduates According to ACM/IEEE/AAAI</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.</strong></td>
<td>#2 - an ability to apply engineering design to produce solutions that meet specified needs with consideration of public health, safety, and welfare, as well as global, cultural, social, environmental, and economic factors.</td>
<td>B – Competent programmer O – Familiarity with software engineering principles.</td>
</tr>
<tr>
<td><strong>Welcome changing requirements, even late in development. Agile processes harness change</strong></td>
<td>#7 - an ability to acquire and apply new knowledge as needed, using appropriate learning strategies.</td>
<td>A – Algorithmic problem-solver - Good solutions to common problems at an appropriate level of abstraction</td>
</tr>
</tbody>
</table>
| for the customer's competitive advantage. | K – Adaptable  
L – Handle ambiguity and uncertainty. |
| Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale. | #1 - an ability to identify, formulate, and solve complex engineering problems by applying principles of engineering, science, and mathematics.  
A – Algorithmic problem-solver  
- Good solutions to common problems at an appropriate level of abstraction  
B – Competent programmer  
K – Adaptable  
M – Analytical and problem-solving skills |
| Business people and developers must work together daily throughout the project. | #3 - an ability to communicate effectively with a range of audiences.  
#4 - an ability to recognize ethical and professional responsibilities in engineering situations and make informed judgments, which must consider the impact of engineering solutions in global, economic, environmental, and societal contexts.  
E – Collaborative  
H – Cross-disciplinary - understanding of non-computing disciplines  
Q – Effective communication skills |
| Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done. | #5 - an ability to function effectively on a team whose members together provide leadership, create a collaborative and inclusive environment, establish goals, plan tasks, and meet objectives.  
E – Collaborative  
F – Socially responsible - ethical behavior  
G – Global and cultural competence  
Q – Effective communication skills |
| The most efficient and effective method of conveying information to and within a development team is face-to-face conversation. | #3 - an ability to communicate effectively with a range of audiences.  
E – Collaborative  
Q – Effective communication skills |
| Working software is the primary measure of progress. | #1 - an ability to identify, formulate, and solve complex engineering problems by applying principles of engineering, science, and mathematics.  
A – Algorithmic problem-solver  
- Good solutions to common problems at an appropriate level of abstraction  
B – Competent programmer  
O – Familiarity with software engineering principles. |
<table>
<thead>
<tr>
<th>Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.</th>
<th>#3 - an ability to communicate effectively with a range of audiences.</th>
</tr>
</thead>
</table>
| **Continuous attention to technical excellence and good design enhances agility.** | **B** – Competent programmer  
**C** – In possession of mental model of computation – deep learning  
**L** – Handle ambiguity and uncertainty |
| #6 - an ability to develop and conduct appropriate experimentation, analyze and interpret data, and use engineering judgment to draw conclusions. | **B** – Competent programmer  
**J** – Think at multiple levels of abstraction  
**O** – Familiarity with software engineering principles. |
| Simplicity--the art of maximizing the amount of work not done--is essential. | **#1** - an ability to identify, formulate, and solve complex engineering problems by applying principles of engineering, science, and mathematics. |
| **A** – Algorithmic problem-solver  
- Good solutions to common problems at an appropriate level of abstraction  
**J** – Think at multiple levels of abstraction  
**M** – Analytical and problem-solving skills |
| The best architectures, requirements, and designs emerge from self-organizing teams. | **#5** - an ability to function effectively on a team whose members together provide leadership, create a collaborative and inclusive environment, establish goals, plan tasks, and meet objectives. |
| **E** – Collaborative  
**M** – Analytical and problem-solving skills  
**O** – Familiarity with software engineering principles.  
**Q** – Effective communication skills |
| At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly. | **#7** - an ability to acquire and apply new knowledge as needed, using appropriate learning strategies. |
| **D** – Life-long learner  
**E** – Collaborative  
**F** – Socially responsible - ethical behavior  
**G** – Global and cultural competence |

The ABET student outcomes criteria focus on ensuring that engineering students have both technical and non-technical skills. Some of the outcomes emphasize strong communication and teamwork skills (#3, #5) and strong problem-solving skills (#1, #2), in addition to being able to apply new knowledge as needed (#7), all of which are essential skills in Agile. The
Characteristics of Computer Science Graduate have similar outcomes, stressing problem-solving skills (A, I, M, P), teamwork and communication skills (E, F, Q), as well as an adaptability mindset (D, K, L), all of which are integral components of the Agile principles.

**ii. Project-Based Learning Principles**

Given that PBL is the most common instructional approach used to imitate industrial processes in the classroom to teach students soft skills [23], the proposed novel educational model is intended for project-based learning software engineering courses. As such, it leverages the principles of project-based learning as presented by [31], shown in Figure 45 below:

![Fig. 45: PBL Principles [31]](image_url)

The PBL principles emphasize the need for a software development project that is central to the course [31]. Everything else within the course – activities, lectures, assignments, etc. – should be driven by the progression of the project [31]. The project within a PBL course should be based on real world problems where the instructor takes on the role of the customer and students are put into teams (of 3 – 6 students) to work on the project [31]. The problems presented as part of the project should be driven by questions (from the students) that will guide them towards investigating and applying software engineering practices [31]. Through these practices, students must provide evidence to show task progress towards the project [31]. Last, there should be a
balance between the involvement of the professor and the freedom that students are given in
developing their project [31].

From the PBL principles Table 24 was created to provide a mapping between the PBL
principles and the Agile core values, shown below:

Table 24: Mappings of Agile Core Values to PBL Principles

<table>
<thead>
<tr>
<th>Agile Core Value</th>
<th>PBL Principle</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Individuals and interactions over processes and tools</strong></td>
<td><strong>Teamwork:</strong> Students should work in teams</td>
</tr>
<tr>
<td><strong>Working software over comprehensive documentation</strong></td>
<td><strong>Project-Based:</strong> There is a software development project that is central to the course. Most classroom activities and lectures are driven by the progression of the project. <strong>Driving Questions:</strong> All project activities are driven by meaningful questions <strong>Evidence-based:</strong> Students must provide evidence of performing tasks (related to the software development lifecycle)</td>
</tr>
<tr>
<td><strong>Customer collaboration over contract negotiation</strong></td>
<td><strong>Realistic:</strong> The project is based on real world problems. The professor plays the role of the customer. <strong>Balance:</strong> There should be a balance between the guidance the professor provides and the freedom that students have <strong>Driving Questions:</strong> All project activities are driven by meaningful questions</td>
</tr>
<tr>
<td><strong>Responding to change over following a plan</strong></td>
<td><strong>Realistic:</strong> The project is based on real world problems.</td>
</tr>
</tbody>
</table>

Table 24 shows the correlations between the PBL principles and the core values of Agile. Agile values teamwork, which is an important aspect of PBL. In addition, the PBL principles
state that the project is the central component of the course; in Agile, prioritizing working software is central. Students can be guided towards progression of the project through driving questions which they can ask each other and the customer, emphasizing collaboration with the customer. Last, one of the principles of PBL is that the project should be realistic, and in many cases, real world problems require constant changes, for which Agile prioritizes being able to respond to these changes.

iii. Proposed Model

This subsection introduces the novel educational software methodology. The new methodology, AgileFlow, combines the strengths of Agile and Waterfall in an attempt to provide a structured approach to lead the students towards adopting an Agile mindset. This new model leverages the Agile principles and values, the PBL principles, the ABET Student Outcomes, and the Characteristics of Computer Science Graduates. In addition, it was created with the help of the background literature and the data collected and analyzed from the surveys. Specifically from Agile, Scrum and JAD were combined to create the sprints for this new model.

This new model consists of the following roles for the students and the professor:

- **Product Owner**: This role is played by the professor, who is responsible for creating, updating, and maintaining the product backlog, and presenting the product goals to the students. As the Product Owner, the professor also has the right to introduce changing or new requirements to the product, updating the product backlog as necessary.

- **Mentor**: This role is played by the professor, who is responsible for mentoring and guiding the student team towards their knowledge and practice of Scrum and JAD. As such, the professor should mentor the students and guide them to adopt an Agile mindset, all while overseeing that the team is understanding and practicing Agile principles and
values. The professor, however, should be cautious in this role not to hinder the freedom that students should have to self-organize their Scrum/JAD practices and events. The professor is also a mentor when it comes to the technical practices required for the project and should provide students with the necessary tools and resources needed for project success.

- **Scrum Master**: This role is initially played by the professor for the first sprint, but for all subsequent sprints it is played by a student. It is up to the professor to decide whether or not this should be a rotating role during each sprint. The Scrum Master is in charge of overseeing the team’s progress and guiding the team in their Scrum practices, such as the daily stand-ups, the review, and the retrospective. In addition, the Scrum Master is also responsible for maintaining the “Flap” (taken from eduScrum), and ensuring that team members are updating their tasks on the “Flap”. The Scrum Master also coaches and helps their team members in their distribution and completion of tasks that lead to Increments. Last, the Scrum Master should make sure that the team is being productive and working in a positive, inclusive environment to establish mutual trust and respect between team members.

- **Coordinator**: This role is played by a student. Their primary responsibility is to coordinate communication between the Product Owner and the team, ensuring that there is constant communication with the Product Owner throughout the sprint and that any questions, challenges, or doubts that the student team may have are brought up to the Product Owner. This student is a direct connection between all the team members and the professor, establishing a loop of mutual trust, support, and respect. In addition, the Coordinator is also responsible for leading the effort of listening to the Product Owner’s
requests and, when applicable, the changes or new requirements, and presenting these to the rest of the team. However, the rest of the student team is still responsible for updating and understanding the new requirements. The Coordinator works closely with the Scrum Master to ensure that the project is driven by the Product Owner’s requests. During the JAD workshops, the Coordinator is the student who plays the role of Session Leader as well.

- **Session Leader**: This role only exists during a JAD workshop. The Coordinator assumes this role. It is this student’s responsibility to lead the workshop session, ensuring that all team members are participating in asking questions to the Product Owner, and that all requests, answers, and feedback from the Product Owner are being taken into account by the team. In addition, this student is responsible for redirecting the team to the task at hand and ensuring that every team member is focused on the session. This member is responsible for speaking the most during the JAD workshops, facilitating the communication between the team and the Product Owner. Last, the Session Leader should encourage the quieter members of the team to participate, and prevent the louder members of the team from dominating the session. After the JAD workshop, the Session Leader transitions back into the Coordinator.

- **Mediator**: The primary responsibility of the Mediator – played by a student – is to mediate any conflicts and/or challenges that may be hindering project progress. As such, this team member is responsible for overseeing that all team members have a role and a responsibility throughout the entire sprint. They work closely with the Scrum Master to coordinate the sprint backlog and ensure that all team members are contributing. If, however, a team member is found not to be contributing to the project, it is the
Mediator’s responsibility to reach out to the team member to offer support and guidance. If needed, the Mediator and Scrum Master can reach out to the professor to get them involved for guidance. The Mediator can keep an informal document as an aid meant to be presented to the team during the sprint retrospective in which they keep notes related to each team member’s progress and contributions, and any challenges/conflicts that arose. In this way, the Mediator can help lead the conversation about team productivity, effort, what worked well, and what did not for the sprint retrospective.

In addition to these roles, this model encompasses the following practices and artifacts:

- **Sprint**: This methodology utilizes Scrum sprints time-boxed to 2 weeks. Each sprint follows the traditional flow as in Scrum, with one addition at the beginning: a JAD Workshop. After the workshop, the flow continues as normal, with daily stand-ups, sprint reviews, and sprint retrospectives.

- **JAD Workshop**: This workshop is primarily focused on the elicitation of requirements from the Product Owner. It is intended to be a structured meeting during which the team asks questions to the Product Owner about the product, developing an initial sprint backlog. This workshop takes place at the beginning of every sprint, and can last anywhere between 30 minutes to the entire duration of the class time, depending on the number of student teams in the course. Since there might be multiple teams in one class, a single workshop can take for all student teams together at the same time, with the Coordinator of each team ensuring their team’s participation during the workshop.

- **“Flap”**: Adapted from eduScrum, this methodology also utilizes the “Flap”, which should be a physical board in which students divide tasks into three columns: “To Do”, “In Progress”, and “Done”. The idea behind a physical board is to encourage students to
have as many face-to-face interactions as possible, even outside of class time. Students can use sticky notes to denote tasks and place them on the board on the appropriate column. Ideally, sticky notes can be color-coded, where each student is assigned a color, and the sticky notes of that color correspond to the tasks assigned to that student. A “Flap” board would be ideal in situations in which students have access to a room reserved only for their use, such that the boards can be left there and accessed at any point during the day by the team members. If, however, this is not possible, it is still encouraged to have students use some sort of visual, physical board, but online tools can also be used to supplement or replace this board.

- **Product Backlog**: The same as in Scrum; it is an ordered list of items needed for the completion of the product. Only the Product Owner should update/modify the Product Backlog.

- **Sprint Backlog**: The same as in Scrum; driven by the sprint goal, it is the plan that the team has agreed to follow throughout the duration of the sprint. Items from the sprint backlog should be visualized on a physical board, similar to the “Flap”, and can belong to the “To Do” column of the “Flap”. The Scrum Master is in charge of organizing and monitoring the sprint backlog, but all team members are allowed to modify it as needed, with team approval.

- **Definition of Done**: The same as in Scrum; this is a clear, transparent understanding of what constitutes an item from the sprint backlog as “done”. This needs to be defined at the class level by the professor and the students during the initial JAD workshop session.

- **Initial Workshop Documentation**: This document is only created by the team (with help by the professor) after the JAD workshop from the first sprint. The intent of this
document is for students to synthesize the information from that first workshop onto a written document which includes all of the product requirements (which should mirror the product backlog). It should include a list of questions asked by the team and their answers, any doubts the team might still have, as well as the contributions of each team member during the workshop. The professor should aid the students in their creation of this document, and it needs to be approved by the professor (in order to ensure that the students are headed on the right path) prior to allowing the students to move on to the next phase.

- **User Guide Documentation**: This document is updated during each working cycle of every sprint. It is meant to contain a user-friendly guide for the end user of how to use the product that is being developed. As such, during each sprint, each team member is responsible for updating the document with the tasks that they implemented, detailing how the feature is meant to be used. This User Guide should be created with the thought that the product will be shipped to an end-user who is not familiar with technical terminology, but can read the User Guide and understand how the developed product is intended to be used.

- **Maintenance Documentation**: This document mirrors the User Guide; it is also updated during the working cycle of each sprint. The main difference between the User Guide and this document is that this is meant to be a technical documentation of the product, such that once the product is finished, another software development team can pick up the Maintenance Documentation and provide maintenance to the working software. It should be comprehensive enough that any other software developer understands the tools and practices used.
In addition to the roles, artifacts, and events described above, the new methodology is broken up into 4 main parts: (1) Preliminary Steps, (2) Sprint 1: Structured Sprint, (3) Sprint Cycles, and (4) Final Sprint. Each part will be explained in the following subsections.

a. Preliminary Steps

Preliminary Steps consist of introducing the students to the Agile values and principles in order to start guiding them towards adopting an Agile mindset. Scrum and JAD are then introduced at a theoretical level to students. Examples of Scrum and JAD in practice should be provided to students in order to expose them to how Scrum and JAD can and have been used in industry. Students should then be divided into self-organized teams of 5-6 students. This size is within with the range suggested by the PBL principles (3-6 students) and suggested by eduScrum (4-6 students).

It is the professor’s decision how to assign students to teams, whether by allowing students to select their own teams, being assigned to teams at random, or any other method that the professor might choose. After the teams are created, the project should be introduced. Once again, it is up to the professor to decide whether a single project will be created by all teams, or if multiple different projects will be created, such that each team might be working on different projects. Given the strong involvement from the part of the professor with this methodology, it is suggested that the professor select a single project for all student teams. For the purposes of introducing this methodology, a singular project will be used. The project should be presented to the students by the professor, who will play the role of the Product Owner. The professor is also in charge of developing the product backlog and presenting this to students.

The Preliminary Steps phase should take part during the first week of the course. The main goal behind this phase is to get students familiar with Agile, Scrum, JAD, and self-
organized teams at an abstract level. The professor needs to be heavily involved during this first week, ensuring that students are understanding the theoretical background of Agile methodologies and practices, while also assigning them to their teams. Figure 46 shows a visual representation of this phase.

Fig. 46: Preliminary Steps

b. Sprint 1: Structured Sprint

The very first sprint should assume that students are not familiar with Scrum nor JAD. As such, rather than allowing students to jump directly into these Agile methodologies and into the Agile mindset, the structured, sequential model of Waterfall was adapted to create a Structured Sprint in which the professor plays the roles of Product Owner, Mentor, and Scrum Master. The only two roles that are assigned to students are those of Coordinator and Mediator, which are, for this sprint, overseen by the professor. Further, the strengths of comprehensive documentation are implemented in this Structured Sprint as well, requiring an additional document to be produced as compared to the regular sprint cycles.

In this Sprint, the self-organized team is hierarchical, contradicting the Agile principle in order to allow for the professor to take on a lead role within the student team. This case only applies during the first sprint. The professor is in charge of introducing students to what an Agile self-organized team looks like, while also showing the students what the Scrum Master is responsible for. Once roles are assigned, the next phase in the sprint, the JAD Workshop, starts.
It is the professor’s responsibility to choose how teams should be formed. Each phase in this Structured Sprint is described below:

- **JAD Workshop:** Just like in a JAD workshop session, this is meant to be a structured meeting – led by the Session Leader (the professor) – in which the students ask the Product Owner a series of driving questions meant to help in the creation of an initial sprint backlog. Since the professor plays the role of Session Leader and Product Owner during the workshop, it is their responsibility to initiate the session, instructing students on how a JAD workshop functions. The professor can provide example questions for the students to think about, and even guide them towards thinking about potential different scenarios about the product that the customer wants. It is also the professor’s responsibility to ensure that all team members are participating in this session, and that any conflicts that may arise between team members during the session are resolved. In addition, the professor should work with the team to come up with a Definition of Done that is clear and understood by all team members. This Definition of Done that is established during this first workshop will remain throughout the entire lifecycle. The workshop should last anywhere between 30 minutes and the entire class duration, allowing for each student team to ask the Product Owner questions. If a single project is assigned to all student teams, the professor can run the workshop with all student teams at the same time, with each team taking turns to ask questions. Once the workshop is over, students are expected to create and submit an Initial Workshop Document to the professor for approval. The professor needs to approve this document prior to allowing the students to move on to the next phase. If it is not approved, the professor can request
changes to be made in order to ensure that the students are correctly understanding the purpose of the workshop and have listed the initial requirements of the product.

- **Working Cycle**: With approval from the professor, the student team is allowed to move on to the working cycle part of the sprint, in which they start from assigning items from the sprint backlog to each team member and begin to work on the development of the product. The professor, as the Scrum Master, is responsible for guiding the students throughout the entire process, ensuring that everyone understands their responsibilities and tasks, participates in daily stand-ups, and updates the “Flap” appropriately. Although the professor is part of the student team as the Scrum Master, they will not be taking any project development tasks from the sprint backlog. Their main role is to oversee, like a manager, team progress, and guide the team towards sprint success by ensuring that all activities are understood by all team members. They are meant to be a guide for students who are new to these Scrum practices. Halfway through the sprint, the team will perform a Half-Sprint Check-In with the professor. This is meant to provide the professor with a more comprehensive and formal update of the team’s current progress and any challenges or conflicts that may have arisen and hindered team progress. It is the professor’s job to once again, guide the students and help them resolve any conflicts. It also allows for the professor to get the student team back on the right track if it is the case that they have deviated from the Scrum practices, or are not adhering to the Agile principles. At the end of the Working Cycle, the student team will produce initial versions of 2 documents: User Guide Documentation and Maintenance Documentation.

- **Sprint Review**: Taking place towards the end of the sprint, this is meant to be the meeting in which the student team presents parts of the working product (which they worked on
throughout the sprint) to the professor as the Product Owner. The team should present a visual demonstration of the working product in tandem with their initial documentation from the User Guide Documentation and Maintenance Documentation. Given that this is the sprint review of the first sprint, it may not be the case that student have a working product to present. If this is the case, students should still present their progress to the Product Owner, along with their documentation, in order that the professor can give them feedback on their work. The professor must approve the team’s presentation and documents prior to being able to move on to the next phase. If it is not approved, the professor must request changes (which, given the amount of professor involvement throughout the sprint, should be minor).

- **Sprint Retrospective:** Just like in Scrum, the team will have a retrospective – led by the professor – to evaluate the team’s performance and find areas of improvement. The professor guides the team to show them what a retrospective is and how it should be carried out. This should be a smaller, less formal meeting, allowing for mutual support and feedback from the professor (as both the Product Owner and the Scrum Master) to the student team.

Figure 47 below shows the Structured Sprint:
c. **Sprint Cycles**

During this part of the methodology, students are given more freedom, adopting the PBL principles of balance. The professor steps back into assuming only the roles of Product Owner and Mentor, while a student from the team is assigned the role of Scrum Master. It is up to the professor to decide if they prefer roles to be reassigned amongst the students in every sprint, or if
the roles that students are assigned at this point will remain their same roles throughout the entire course. It is encouraged for students to change roles in every sprint in order that each student can experience every different role. The phases in the Sprint Cycles are as follows:

- **JAD Workshop**: Once again, every sprint is initiated with a workshop session, this time led by the student Coordinator who takes on the role of Session Leader during the duration of the workshop. The intent is to elicit requirements from the Product Owner. This time, however, the Product Owner has the option to introduce new or changing requirements. If the Product Owner changes or adds any new requirements, they will also update the product backlog to reflect these changes. As such, the Session Leader facilitates the communication between the team and the Product Owner, making sure that all of the team members are participating in the session and are updated on the current product backlog. The JAD Workshop is where the professor will also help guide the students towards what items are important to be worked on for the upcoming sprint. However, the professor should ensure that the students are also given the freedom to select, as a team, items from the product backlog that will be part of their sprint backlog. At the end of the workshop, the students should have an initial sprint backlog. In order to give students more freedom and flexibility, they are no longer required to seek approval from the professor before moving on to the next phase.

- **Working Cycles**: Similar to the first sprint, the working cycles represent the cycles between working on items from the sprint backlog, the daily stand-ups, and updating the “Flap”. During these Sprint Cycles, however, it is the student Scrum Master who is responsible for overseeing the team’s progress, all while ensuring that the Scrum practices are being followed correctly. It is also their responsibility to oversee and update
the “Flap”, making sure that it is accurately portraying the current state of the sprint progress. During the working cycle, the Coordinator should maintain communication with the Product Owner to report on progress, potential conflicts that may arise, or even to ask clarifying questions throughout the development process. This can be taking some class time to go up to the professor (the Product Owner) and ask questions that the team has while the team works. The Coordinator is also responsible for hearing the Product Owner’s input and feedback and conveying these to the team. Last, the Product Owner has the option to introduce changing or new requirements during any point in the working cycle. As such, it is the Coordinator’s responsibility to communicate these changes to the team, and ask any necessary questions. During the working cycle, it is the Mediator’s responsibility to ensure that all team members are participating and contributing towards project progress. In the case that any team member is not participating, the Mediator must help mediate the situation by first communicating with the team member, involving the professor when deemed necessary. The Mediator should also seek support from the Scrum Master. Just like in the Structured Sprint, the User Guide Documentation and Maintenance Documentation are also produced, with these documents simply being updated (rather than creating new versions of them) to include the new information relevant to the sprint.

- **Sprint Review:** Similar to the Structured Sprint, the Sprint Review is a formal presentation of the working product produces during the sprint to the Product Owner, who provides feedback and guidance. However, during the Sprint Cycles, the review is led primarily by the student team, with the Coordinator leading the communication with the customer, and the Scrum Master facilitating the presentation with the student team.
The professor primarily plays the role of Product Owner, offering guidance and suggestions to the team as part of the Mentor role as well, but prioritizing giving the team as much freedom as possible to lead their own presentation and review. The team also presents the User Guide Documentation and Maintenance Documentation briefly, as the main focus of the review should be on presenting the working product. The Product Owner also has the ability, during the review, to introduce or change any requirements (mainly in the form of feedback), for which the team will, in the upcoming sprint, take into consideration in their sprint backlog. The product backlog is updated by the Product Owner when they introduce new/changing requirements. The professor no longer needs to approve the presentation for the team to move on to the retrospective. Instead, any features that were not approved by the Product Owner should be put back into the sprint backlog, but the team is allowed to move to the next step, just like in Scrum.

- **Sprint Retrospective:** The retrospective is led by the student team. The Scrum Master should facilitate and guide the meeting, ensuring that all members of the team are participating, and focus on feedback/observations made by the Mediator. The Mediator takes a central role in the retrospective, reporting on the observed progress of each team member and providing feedback and support to each team member. If it is the case that a team member is not contributing to the team, or they are hindering team progress, it is the Mediator’s responsibility to professionally address this issue during the retrospective such that the entire team can decide how to proceed. If needed, the professor can be involved in the retrospective. The professor’s main role during the retrospective is to take a step back and allow the students to lead their own retrospective. This should be a more
relaxed meeting in which the team members can bond, establish mutual trust in one another, and together assess how they can improve for the next sprint.

Figure 48 shows the Sprint Cycles:

For the Sprint Cycles, the green circles (see Fig. 48) represent moments in which the Product Owner reserves the right to introduce changes or new requirements. However, this is up to the professor to decide when and how to do so. It is not required to introduce changes or new requirements in each phase. It is encouraged that the professor does introduce or change requirements at least once during a sprint in order to get students into the Agile mindset of responding to change, becoming adaptable, and handle ambiguity.
In addition, the professor should also guide the team through continuous testing of the working software. As the Mentor, the professor should ensure that the teams are continuously testing each increment of the product. However, it is up to the professor to decide when, where, and how testing should be done. In addition, the professor can provide example test-cases to the teams in order to show them good testing practices.

d. Final Sprint

During the Final Sprint, the professor shifts the focus and priority of the students to finishing the project and testing the final product. As such, no changes nor requirements should be introduced during this last sprint. The professor, as Mentor, will take a stronger role in the student team, ensuring that the team is nearing product completion and guiding them towards prioritizing those items on the “Flap” that are “In Progress”. Last, the professor should also emphasize extensive testing, and provide example test-cases to the students for testing the product. The Final Sprint is broken up as follows:

- **JAD Workshop:** The workshop is the same as in the Sprint Cycles, the only difference being that the professor, as the Mentor, should guide the team to prioritizing certain things from the product backlog. This is to ensure that the team can deliver a working product, even if it was not possible to fully implement all items from the product backlog. As such, the session, led by the Coordinator as the Session Leader, is focused on asking the last clarifying questions, and the professor should guide the students to prioritizing certain items over others. This can vary by team, depending on each team’s own progress. Given that each team has had the flexibility to develop their own sprint backlogs and self-organize their own work up to the final sprint, it may be the case that each team has different features implemented. As such, the professor plays a more crucial
role in ensuring that each team is capable of delivering the most important aspect of the product that he, as the Product Owner, has asked for. He helps each team develop their final sprint backlog.

- **Working Cycle**: The working cycle is the same as in the previous sprints, the only difference is that the professor should, once again, guide the team towards project completion. As such, the professor should be more involved in the team's progress, while still ensuring that the team has the freedom to self-organize their tasks. The Scrum Master drives the focus to moving all items from “In Progress” from the “Flap” to “Done” for each team member. In addition, the Scrum Master should also ensure that if one team member is finished with his tasks, the team member takes on another task from the “Flap” that is under the “To Do” column (meaning no other team member is currently working on that item). If there are no items under “To Do”, the team member should then be assigned to help any other team member currently still working on their tasks in order to ensure mutual support between team members.

- **Preliminary Presentation**: Halfway through the sprint, each team should have a preliminary presentation where they present their prototype (even if unfinished) to the Product Owner, who again offers feedback and guidance. Both of the documents – the User Guide Documentation and Maintenance Documentation – are also presented for feedback. This should be structured as a practice presentation of the product to the Product Owner, allowing the professor to better guide the students to ensure that they can finish the product in the next half of the sprint.

- **Final Presentation**: After the Preliminary Presentation, students go back to the working cycle for the next half of the sprint. Towards the end of the sprint, the Final Presentation
takes place. This is the formal presentation of the final product and documents to the Product Owner, led by the Coordinator and facilitated by the Scrum Master, similarly to the previous sprint reviews.

Figure 49 shows the Final Sprint:
The entire model can be visualized with Figure 50 below, which generalizes each of the parts described above. This model is read from left to right, with the Preliminary Steps first, followed by the Structured Sprint, the Sprint Cycles, and ending with the Final Sprint.

Fig. 50: AgileFlow Model
Chapter 6: Conclusions

This chapter will focus on concluding discussions regarding this research work. The chapter will also serve to discuss the answers to the research questions of this work and summarize how the goals were achieved.

1. Research Question 1

What are the gaps between industry and academia in the software engineering field with respect to software development practices?

According to the literature, recent graduates from the computer science discipline have been found to be lacking areas/skills that are necessary in Agile methodologies, such as teamwork, project management, and communication, among others [7]. Figure 16 shows the top 6 most identified knowledge deficiencies in recent graduates from a study conducted by [7]. The literature states that universities have typically delivered technical skills well, while the soft skills have usually had less support in computer science programs [23], [31], [59].

In order to test this statement, Question 5 from Discovery Survey 1 was asked to students in order to identify which areas they believe themselves to struggle the most in. Figure 31 provides the results for this question, showing that the top 3 skills students believe themselves to struggle the most in were testing, written communication, and software tools. Out of these 3 skills, only one is a soft skill: written communication. Teamwork, the fourth skill (right after software tools), is another soft skill with which student believed themselves to struggle the most in. As such, based on the literature and the results from the survey, it can be seen that students believe themselves to struggle in a mix of both technical and non-technical skills.

As such, Research Question 1 was found to include different gaps between industry and academia pertaining to technical and non-technical skills in students that are necessary for
software development practices. Teamwork, testing, written communication, software tools, and all other skills from Figure 31 are important for software development. Specifically related to Agile Software Development, communication and teamwork practices are crucial.

Question 3 from Discovery Learning 2 also revealed that, students who had, at the moment, not been taking an Agile course are statistically significantly less familiar with self-organized teams than those who were currently taking an Agile course. This reveals yet another gap between academia and industry, since students are not being exposed to self-organized teams in academia prior to heading towards industry. With a growing use of Agile Software Development in industry, students must have the knowledge of working in self-organized teams.

2. RESEARCH QUESTION 2 AND RESEARCH QUESTION 3

RQ2: How can Agile Software Development practices be leveraged to create a synergistic learning environment that enhances student learning within software engineering courses that use project-based learning?

RQ3: How can utilizing the strengths of existing software development lifecycles aid in the creation of a novel educational software methodology that incorporates the principles of Agile Software Development?

i. How are different Agile methodologies currently leveraged in software engineering education?

Institutions of higher education have historically been using passive learning teaching approaches in their classrooms [6], [8], [9], [10], [11]. Through the surveys, it was found that students learn better through the use of active learning activities (see Figure 42), and that students are generally exposed to group work in classes of higher classification (see Figures 33 and 34). Since Agile requires active learning activities, it is important to consider the results
from Figure 42 when answering Research Question 2, and that students are generally more exposed to group work in Senior-level courses.

In order to further answer Research Question 2, it is also important to understand the PBL principles, given that the software engineering course that would leverage the Agile Software Development practices would use project-based learning. Table 23 provides a mapping between the Agile values and the PBL principles. This table was instrumental in developing a new educational methodology that would leverage ASD in a PBL course.

Last, in order to answer Research Question 3, it is also important to understand the literature on how ASD practices have been used in institutions of higher learning, in order to understand the challenges that arise. Throughout the literature, there have been different proposals and experiments to introduce Agile Software Development to computer science students. Each come with their own benefits and challenges, of which one of the major areas of concerns was the importance of the professor’s involvement throughout the process in order to guide the students towards an Agile mindset. Adopting an Agile mindset is, in itself, a very challenging process [17], [37]. As such, it is imperative that any methodology or model used to introduce students to Agile focuses on continuous guidance, support, and mentorship from the professor to the students as they adopt an Agile mindset.

Agile principles were mapped to educational objectives in Table 22, providing a guidance on how Agile practices can be leveraged to achieve these educational outcomes in computer science students. AgileFlow is a direct response to the challenges outlined in the literature pertaining to how to teach Agile to students. First, by mixing the strengths of Waterfall and Agile (see Chapter 3, section 2, subsection vii, and section 3, subsection iii), it provides a structured way to introduce students to Agile, providing a lot of professor guidance in the initial stage. Just
like in Waterfall, students go through the first phases in a structured manner, requiring professor approval before finishing one phase and moving on to the next. In this way, students are slowly introduced to both Scrum and JAD.

Once this Waterfall phase is over, the professor takes a step back and allows students the freedom and flexibility to explore their own Agile process, self-organizing themselves and planning/leading their own meetings. The professor is still there to provide guidance, but they take a much more relaxed role, guiding students – but allowing them the freedom – in all of their Scrum/JAD practices.

The methodology should be complemented with small lectures as necessary, prioritizing most of the class time in allowing students to work on their projects. As such, a mix of passive and active learning activities are used. Lectures are helpful towards students’ learning (see Figure 42), but these should not become the central aspect of the course, which is project-based. Professor feedback, the most helpful activity in student learning as seen in Figure 42, is imperative throughout the entire process of the methodology.

The methodology also makes use of constructive and interactive learning, since students are required to work in teams and together, construct their own knowledge and understanding of Agile Software Development through experiences. As such, the methodology aims at increasing student learning in the classroom and allowing them to learn/practice numerous soft skills, such as project management, communication, and teamwork, by combining the strengths of Agile and Waterfall.

3. DISCUSSION

The literature provided a good understanding of where the current gaps and challenges for introducing Agile Software Development into the classroom lie. The surveys further
confirmed some of the information provided by the research, such as the areas in which students struggle the most in. Further, with a greater understanding of the challenges in software engineering education, and the different learning processes, AgileFlow was developed as a model for introducing students to Agile Software Development.

It is important to offer students a combination of passive and active learning techniques, as every student learns differently. However, students should have more activities that fall under constructive and interactive learning, which are more beneficial in helping students construct their knowledge. As such, one of the priorities of the new methodology is to center most of the class meetings on teamwork, allowing students the time to interact and work on their projects.

Project-based learning is an effective form of constructive and interactive learning, allowing students the opportunity to solve real-world problems through hands-on experiences and discovery. In addition, the PBL principles map well with the values of Agile, making it a good fit to introduce students to Agile practices. The project is guided by a mix of Agile methodologies and Waterfall in order to allow for a structured approach to teaching Agile. Professor guidance and mentorship throughout the process is heavily emphasized, all while ensuring that students have the freedom and flexibility to self-organize themselves and lead their own team and project work.

4. LIMITATIONS

This section will describe the different limitations of this study. Specifically, any time a survey is used and distributed, there is always a risk of personal bias that may affect the results. In addition, students are also subject to misinterpreting certain terms for which definitions were not provided in the surveys. For example, in Questions 3 and 4 of Discovery Survey 1, no definitions for any of the software development lifecycles were provided. As such, students may
have selected lifecycles they believed to have used on a project based on a flawed definition of that lifecycle. Similarly, Question 3 and Question 4 from Discovery Survey 2 are subject to misinterpretation since no definition of a self-organized team is provided. Students may have inferred their own definitions for self-organized teams that are not the same as the definitions used in this research.

Another limitation of this study is that the surveys were only presented to students at one institution of higher learning throughout 2 semesters limiting the sample size for data collection from the surveys. In addition, no surveys were distributed to organizations in industry, limiting the data collection from the side of industry.
Future Work

This section describes the future work relevant to this research. Future work would include carrying out an experiment to test the AgileFlow model within a classroom, and assess different factors, such as student learning and motivation, in order to see if the methodology is beneficial for introducing students to Agile Software Development. To have a direct comparison, two courses can be carried out simultaneously, one utilizing the new methodology and the other using any other existing model, such as eduScrum. In this way, a more comprehensive comparison can be done to test the effectiveness of the new model against an existing model that has a similar goal.

In addition, more work is needed to understand more of the gaps that exist between academia and industry. In order to understand the gaps from the side of industry, surveys can be created and distributed to different organizations from the software engineering industry to gather more information on what other gaps between recent graduates and the expectations of industry there might be. Similarly, additional surveys can be created to assess more information from the side of academia, and future work would include distributing surveys to more institutions of higher learning to gain a broader understanding of the current challenges in software engineering education.
References


Vita

Montserrat Molina was born and raised in El Paso, Texas. She graduated from Loretto Academy High School in 2018 and from her Bachelor of Science in Computer Science in fall 2022 from the University of Texas at El Paso. Throughout her undergraduate education, Montserrat worked part-time as an Instructional Assistant and Research Assistant and completed 4 software engineering internships during the summers. She began her Master’s in Computer Science in January 2023. During her time as a graduate student, Montserrat was a Teaching Assistant for fundamental computer science courses.

Contact Information: mgmolina3@miners.utep.edu