Implementing Agile as an Instructional Methodology for Low-Code Software Development Courses

Laura F. Poe
lpoe@richmond.edu

Lionel Mew
lmew@richmond.edu

University of Richmond
Richmond VA

Abstract

The objective of traditional software development courses focuses on competencies of the programming languages and technical tools. Project methodologies and software development are typically taught as separate courses in Information Systems undergraduate programs and are not incorporated until the final Capstone course. Rather than teaching project methodologies as secondary to the learning phase of software development, these methodologies can be actively incorporated into the software development course, applying the theoretical concepts in the classroom. This research measures the outcome of instituting the project methodology, Agile, as an instructional tool for a low-code software development course using the Mendix platform.

Keywords: Software development; Agile; pedagogy; rapid delivery; instructional methodology

1. INTRODUCTION

This paper reports on development of a new program combining Agile methodologies with a low-code platform to provide a synergistic experience for non-traditional students in a six week summer course. Using a low-code platform as a tool to support the course is possible due to the emergence of Model Driven Development (MDD) tools such as Mendix in a new wave of Computer Aided Software Engineering (CASE) tools with expanded capabilities. With the continued adoption of Agile methodologies throughout industry, the combined experience is designed to add value to student portfolios as they prepare to enter the workforce.

CASE tools are software tools used to help design and generate applications. This occurs at a higher level of abstraction than application development using traditional linear programming methods (Halpern & Tarr, 2006). From the advent of CASE tools in about 1970, the ability of these tools to easily and reliably generate applications did not emerge until decades later.

The CASE tools of the 1980's allowed less technical practitioners to generate database applications by facilitating higher levels of abstraction – automating and simplifying application development using the context of domain models, with the tool generating development documentation, code, and in some cases, fully functional applications.

While many professionals used CASE tools as aids to the development process, few of them used the tools for complete database application generation. Several reasons impacted the decision. The cost and complexity of installing

and maintaining the tools, training and paying the higher salaries of their users, coupled with the tool's abilities and failure to perform as predicted, led to the tools having little commercial impact during the 1980's and 1990's (Schmidt, 2006). Jones (2002) notes that as many as 70 percent of CASE tools were not being used.

Reasons for the first-generation CASE tools' lack of acceptance include unrealistic performance expectations and inadequate training. The emergence of better tools for MDD, evolving from first generation CASE tools, have improved performance such that they are now being used for medium to large scale development projects. Efforts are being made to address the training issues. Students entering the workforce must have some exposure to these new and innovative tools. Incorporating the Mendix platform into this Agile course is an effort to provide students with exposure to a development project using an MDD tool, in addition to fostering the students' understanding of working in teams using an Agile methodology.

The selection of Agile was based on the rise in practitioner usage of the Agile methodology across the enterprise. Understanding software development as well as project management methodologies are core skills for Information Systems programs. Traditional 'Systems Analysis and Design' courses focus on the Systems Development Life Cycle, which primarily uses a Waterfall methodology approach. Waterfall methodologies, unlike Agile, follow a systematic advancement from requirements, design, development, testing, and implementation over a long period of time and cover large functional/programmatic changes. Agile's focus on smaller, incremental changes in functionality and programs allows teams to work in cycles of two to three weeks. These short cycles can be easily incorporated into a classroom setting and allow for software development projects to be successfully finished in the duration of a semester. Students are then able to learn both the software development tools and concepts along with the project methodology.

2. STUDENT POPULATION

The student population at the School of Professional and Continuing Studies (SPCS) consists of mostly non-traditional students. Although no commonly accepted definition for a non-traditional student exists, some insight may be gained by examining SPCS student demographics. The average student age is 37, although the information systems students are closer to 30 as a group. Previously, the majority of students have been male, although the number of females in the program is slowly increasing, with females accounting for more than half of enrollments in the past semester. Experience levels and goals of female students are similar to those of the male students, in which a wide variance applies equally. Some who are career switchers have little or no knowledge or experience, while others have been working in the field for years.

Eighty-one percent of students are part-time. Both part-time and full-time students are working on either Bachelor of Science in Professional Studies degrees with a major in Information Technology Management or Information Security, or a Post-Bachelor Certificate in Applied Studies, Information Systems, or Information Security. As previously mentioned, student experience varies, with some having Associate's degrees or at least some community college work, and have immediately transferred to SPCS with a desire to complete their Bachelor's degree. Others have been in the workforce for some time and need a degree for promotion, yet others are trying to break into the information systems field with significant life experience and success in other fields. The program can be seen as a degree-completion program, since most new students have 45 – 60 credit hours in transfer.

The factors which make SPCS students unique lead to a wide variety in student understanding, experience and ability. All major core courses are classroom-based courses, although some are offered in hybrid format. No completely online information systems courses are offered, but some non-major courses may be taken online. Most of the students live in the local area, and the majority of students stay in the area after graduation. Courses are generally capped at 15 students with overrides up to 20 students, allowing significant individual attention and interaction with instructors. The Agile Low-Code course is an elective 3-credit course in hybrid format offered during a 6-week summer session. This particular iteration started with 14 students, of which two withdrew early in the session.

The diverse student population presents numerous opportunities as well as challenges. Much of the student body brings life and work experiences to the classroom while facing the challenges of family and work obligations. The applied aspects of the course arguably add more value to this student population, whereas the traditional students continue to mature and learn to think critically during their degree programs.

Continuing students in this program are expected to focus more on professional competencies, with the primary focus for instructors to help students grow professionally.

To facilitate learning amongst this target population, the high-level philosophy is to provide an applied aspect for each course. This student population likes hands-on work. This special topic course in Agile Low-Code development is an exciting endeavor, since the expectation is that the higher level of abstraction allows more students to develop competencies in the course material. Students taking this course vary in level of experience with some students having previous background in systems development courses and other students with no software development background.

3. MODEL DRIVEN DEVELOPMENT

Background
Model Driven Development (MDD) is a software development methodology that uses model architectures instead of coding to raise the level of abstraction such that less technical expertise is required to develop applications, with business knowledge being the key ingredient for users.

According to Henkel and Stirna, modeling support consists of language(s) for modeling constructs in the specific domain, supported by key underlying areas of abstraction, understandability, executability and model refinement. Regarding development support, or support for development processes, Henkel and Stirna found the literature suggests six areas: observability, turn-around-time, collaborative development support, integration, developer competence support and reusability.

When evaluating Mendix against these criteria, Henkel and Stirna found Mendix to be suited for web-based development of small to medium complexity and for small projects with short delivery times. These qualities were determined to make Mendix a good fit for supporting this and other traditional information systems courses.

The FFIEC IT Examination Handbook Infobase (n.d.) proposes four areas of risk when considering implementation of CASE tools, including inadequate standardization, unrealistic expectations, inability to implement quickly, and weak repository controls. Mendix was chosen based on its ability to manage these risks.

Mendix

The Mendix tool was used for the course due to its functionality, simplicity, and the support offered by Mendix. The platform is the leading low-code solution recognized by analyst reports, such as Gartner, and large enterprise companies, SAP and IBM. The full-stack platform is designed to build applications rapidly. The platform abstracts and automates the various application development layers from front end to back end. For example, the data structure is built using the Unified Modeling Language (UML). The business logic uses Business Process Modeling Notation (BPMN), and the user interface is built with widgets following a 'what you see is what you get' (WYSIWYG) model.

The platform allows for business and IT to collaborate and build applications that add business value and provides 6 main functions:

- Collaboration
- Data Structure and Domain Model
- Business Logic
- User Interface and Experience
- Security and user authentication
- Deployment

Collaboration
Collaboration is key to building successful applications that solve business problems. The business understands the critical business problems and needs digital solutions to fix those problems; whereas, the IT department needs to support the business by providing the solutions that work. Communication across these two departments has always been challenging. In order for both business and IT to collaborate, speaking the same language, delivering on time, and delivering under budget are key success factors. In addition, solutions delivered months and years after the original business problem has been identified are unreasonable and reduce the realization of the solutions' return on investment.

To deliver applications rapidly, in weeks versus months, a process change is needed. Agile methodology follows the iterative process and business can have a minimal viable product within weeks with the ability to iterate as needed (Frydenberg, Yates & Kukesh, 2017). The Agile methodology allows for iterative development and for the business to provide input and shape the product before it is delivered.

When students create a Mendix project, the collaboration workspace is automatically created with all the built-in Agile process features. For example, students can capture their user requirements and add user stories and sprints.

They can create sprints that run for a fixed set of time and manage the backlog of stories and work to be done. In addition, using the feedback widget, they can gather feedback from the business or professor and implement additional features and functionalities. The application they build in the first sprint will be vastly different than the application they deliver at the end of the semester. Students can see the process and workflow as they build the solutions out. The project manages the code repository and code check in and out process which allows for multiple students to work on a diverse set of user stories.

4. SCOPE AND RESEARCH OBJECTIVES

The objective of this study is to examine the use of Agile as the course delivery model for software development courses and determine the effectiveness of Agile as a teaching methodology. Previous research has indicated successful usage of Waterfall and Agile methodologies when teaching Capstone, project-based courses. The objective of software development courses focuses on mastery of the programming languages and technical tools; methodology is secondary to the learning phase of the software development. Previous research indicates that self-regulated learning increases student motivation through engaging classroom environments (Linden 2018). Agile as a pedagogy promotes self-regulated learning and self-managing teams. Studies of Agile have concluded that overall performance is linked to the effectiveness of team coordination in software development teams (Moe 2009).

The results of the study provide educators with an alternative course delivery method that prepares students to create software development solutions that could be usable artifacts in the industry and familiarizes them with the practical applications of the Agile project methodology. Industries hiring graduates of information systems and computer science programs expect intrinsic knowledge of project management methodologies, extending beyond software development knowledge. Utilizing Agile as a pedagogy is a response to the current demand for skilled workers with a sound understanding of the project methodology that is rapidly overtaking the Waterfall and formerly used methodologies.

Pedagogical principles that were followed during this course included 1) integrated learning environment and 2) cooperative learning. The integration learning principles focuses on incorporating the technology with the Agile methodology. Students are able to learn both the content of the software development tool and functionality and the subject area of Agile. Cooperative learning allows students to work collectively, as a group, to learn from each other through hands-on practices. As identified by Niess and Gillow-Wiles, educators shift toward building a pedagogical reasoning that integrates technologies as teaching and learning tools (2017). The communication achieved through the Agile ceremonies provides the assimilation of systems learning in a pseudo-practitioner environment.

The following research questions are explored:

1. Is the Agile Methodology an effective method for teaching software development courses?
2. What is the impact on team dynamics and performance using Agile as a technological pedagogical approach?
3. Do students perceive their level of software development learning was increased, decreased, or unchanged by using Agile as a pedagogical form of course delivery?

Research Method

An exploratory, six-week custom developed course in information systems was created, titled Agile Low-Code Development, with a minimum enrollment of twelve and maximum enrollment of fifteen. The course enabled participants to develop applications using the Mendix Low Code Platform and the Agile project methodology. The use of Agile facilitated students' learning of the widely used software development methodology where requirements and solutions evolve through structured team collaboration. The Mendix Low Code Platform does not require coding experience, giving both technical and non-technical students the skills to build web and mobile applications. Using the Mendix tool provided additional advantages with its built-in Agile processes. The results of the final products built by the teams and the qualitative feedback gathered from students at the end of the course were used to determine the effectiveness of Agile as an instructional methodology for software development.

The Agile tool selected for creating stories and tracking progress was GitHub, a free online tool used by numerous government entities and private industry organizations. GitHub was applied to support Agile through story tracking, assignment, and team monitoring activities. Project teams used GitHub's storyboard, ZenHub,

to display and update the project board. Activities, such as moving stories from 'ready' status to 'in progress', 'testing', 'done', and 'closed' were performed on a daily basis during team scrum ceremonies. These scrum ceremonies were daily meetings, synonymously referred to as stand-ups, where each team member provided the following information: what was accomplished previously, what will be accomplished by the next meeting, and any impediments preventing the completion of the work.

To promote team independence, a hybrid model was instituted, having one in-person class per week and allowing for any remaining meetings to be held by the teams at their convenience. The course achieved three primary goals: 1) teach basic concepts of Agile from a holistic view; 2) develop the knowledge and ability to use the Mendix Low-Code Platform; and 3) develop an application using Agile and Mendix in a group project. In order to achieve the three goals, the delivery model of the course operated as an Agile project using iterative development based upon weekly sprints.

The course was guided by two professors, one of whom focused on Mendix development while the other focused on ensuring the Agile fundamentals were taught and ceremonies were followed. Students worked in teams to develop a custom application based on predetermined parameters. Students were able to select a project of their choosing within the course parameters.

Rather than allowing self-forming teams, professors randomly assigned four teams of three as well as the student roles on the team. While the role of the professors was instructional, professors, also, acted as customers of each of the Agile teams. The three-person teams included one member with had a combined role of scrum master and product owner and two developer/tester roles.

The first week of the course was instructional and allowed for team norming with the remaining five weeks divided into weekly sprints. During the initial week, fundamental concepts and instruction on both Agile and Mendix were delivered by the professors in addition to an introduction to the GitHub and Mendix tools. Students were seated with their teams, and time was allotted for teams to develop their team norms. The development of team norms is an essential component of Agile and stipulates a team's expectations of team members as well as meeting cadences.

At the end of the first week, students were expected to take the Mendix developer certification to indicate competence in the tool, essential for developing their applications. Weeks two through five repeated a cycle of sprint review, retrospective, Mendix training and tutorials, and sprint planning. This order was repeated on the first weekday meeting of the course. Each of the Agile ceremonies, i.e. sprint review, retrospective, and sprint planning, were held during class and facilitated by a professor acting as a Release Train Engineer (RTE), a role held by a leader in charge of multiple Agile teams. Each Agile team used the time given by the professor to complete the ceremonies. Teams were expected to have the output for sprint review and sprint planning ceremonies visible in the GitHub tool. Teams were, also, expected to have regular scrums throughout the week to track progress against assigned stories.

Weekly deliverables were set forth by the professors and became an essential component of teams' stories. Grading rubrics for each deliverable were provided in advance, and teams received a collective grade for the majority of the coursework, Appendix Table 1. Grades were measured not only on the application functionality but on team ability to create stories with measurable and testable acceptance criteria, estimated story points, owners assigned, and proper story tracking throughout the sprint. Detailed feedback was provided weekly, and students could use this feedback to make corrections in future weeks. Product owners were responsible for ensuring that stories were complete based on the acceptance criteria and moved each story to the status of closed during sprint review. If a story was not complete, the team included the story as part of upcoming sprint planning, outlining the missing or nonfunctioning components of their application that must be corrected during the subsequent sprint. The final week, sprint 5, of the course was focused on application security and correcting any errors found during testing or the product owner's review. As a final deliverable, each team provided a demo of their applicable to the class.

5. RESULTS

All teams satisfactorily produced a software development project that met the previously provided criteria of the project rubric. The software developments projects all had a similar level of complexity and functionality based on the requirements provided by the professors.

ISSN: 2473-4901
v5 n4951

The level of Agile maturity increased each week as teams became accustomed to the weekly Agile ceremonies. Teams progressed in the development of the application at a more regular pace than in a traditional course delivery approach due to the required weekly team collaboration. Using Agile forced teams to plan in advance with weekly deliverables and have consistent communication. Procrastination was significantly minimized due to the visibility of the application's progress and story tracking, Figures 1 and 2. As shown in Figure 2, teams were able to view their progress from sprint to sprint based on team velocity, which captures the quantity of work completed each sprint. Work quantity is measured using story points, an exercise that provides a numeric value to estimate the difficulty and time required to complete the task.

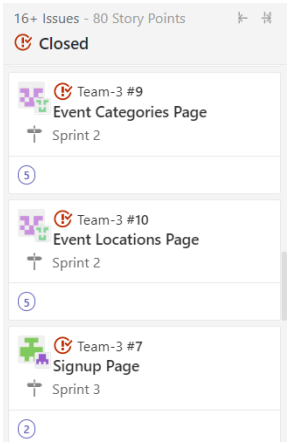

Figure 1
Closed stories in GitHub

Student Feedback

The team's product demos were the final deliverable and revealed the overall success of the students and the pedagogy. Of the four teams, two met the criteria expected in the application, and the other two lacked some of the required components.

As part of the course evaluation, the professors facilitated a retrospective of the course with the students, gathering in depth feedback. All students participated in the retrospective during the last session of the course. The retrospective followed the format of three basic questions:

- What did we do well?
- What should we continue doing?
- What should we change?

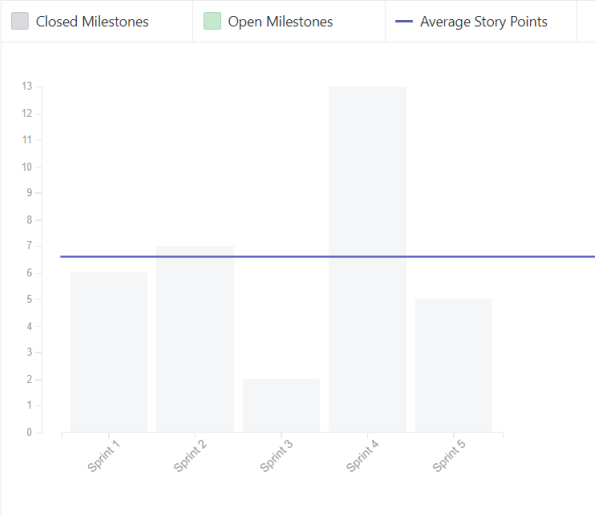

Figure 2
Team velocity

The student response was positive overall, and a detailed list of feedback can be found in Appendix Table 2. In terms of the level of difficulty, eight of the twelve students found the course to be moderately difficult, with the remaining four indicating a high level of difficulty. Feedback, also, indicated that the level of difficulty was reduced using the Agile methodology for course delivery. Breaking the course material into sprints and forcing the students to have weekly sprint planning during the classroom setting was one of the advantages identified by participants.

Challenges faced by the students on Mendix technical issues and questions caused the majority of uneasiness. The six-week duration of the course was insufficient for some students to develop the level of competence desired for app development, and feedback suggested that learning both Agile and Mendix competencies in six weeks proved to be difficult. Suggestions were to have the course offered in the normal 16-week semester format. While the issues with Mendix as an application were evident, these issues were separate from Agile as a pedagogy.

Research Question 1: Is the Agile Methodology an effective method for teaching software development courses? The results of the study indicated that use of Agile was positive and could be applied to all software development courses. However, course length is recommended to be in excess of six weeks to achieve maximum results. Students who normally lacked organizational skills or faced challenges prioritizing tasks benefited most from Agile pedagogy. Teams were forced to collaborate through regular planning sessions, and their output reflected

structured preparation, driving additional task prioritization. The Agile pedagogical approach simulated a practitioner's perspective of software development where development teams have customer demands and critical timelines.

Continuous learning is a key principle of Agile and forces students to self-reflect on their sprint work, changes that need to be made, and how to incorporate required changes into the new functionalities for the next sprint. These are real world challenges being faced in the classroom in a controlled environment where students can feel safe to make mistakes, correct, and progress. Continuous learning requires ownership of their learning, progress as a team and as an individual, and the observed student learning was achieved rapidly.

Research Question 2: What is the impact on team dynamics and performance using Agile as a pedagogy? Project-based teams shared the same end goal to develop an application, though the specific requirements for each application differed. The small sizes of the teams created closer team relationships and required team members to work together to solve challenges. Having teams of three limited the possibility of passive team members. Each individual was vital to the success of the project. The team could not succeed without everyone's contributions. Therefore, the overall learning of the software development tool was elevated for all participants.

DevOps represents the Agile association between development and IT operations and constitutes a variety of coding and testing practices designed to speed up product delivery. Two DevOps practices were observed during the class, pair programming and extreme programming. Pair programming is coding performed by two people on the same machine, while extreme programming makes continuous programmatical adjustments based on changing requirements. Teams using these methods are more productive and produce fewer defects (Rico et al. 2009). Without formally introducing pair programming or extreme programming concepts into the classroom, the students were actively performing both practices throughout the six weeks. Teams often had one laptop and three team members all reviewing the app development and making changes together. Some teams used the product owner to make recommendations while the other two team members made the functionality modifications. The reduced team size enabled rapid delivery of the application and showcased Agile and DevOps procedures being applied, often without instructor involvement.

Research Question 3: Do students perceive their level of software development learning was increased, decreased, or unchanged by using Agile as a pedagogical form of course delivery? Based on student feedback gathered during the retrospective and the end of course evaluations, students felt the level of learning was increased by using Agile despite the suggestions to lengthen the course. Course assignments used the sprint framework, and participants understood the weekly expectations. In addition, the teams were motivated to perform and took pride in the application they were building.

7. CONCLUSION

The overall results of the study support the utilization of Agile as an instructional methodology for low-code software development courses. The professors and students found that the results of the app development either met or exceeded expectations, and attributed the application of Agile across teams increased their ability to complete the project. Students were able to gain hands-on experience in Mendix while mimicking a real-world Agile project. The Agile pedagogical approach could be transferable to traditional undergraduate student populations and is suggested for low-code platforms in order to maximize the ability for the students to complete the project.

8. FUTURE RESEARCH

Recommendations for future study include expanding the use of Agile pedagogy to other software development courses for obtaining a larger sample size to measure impact and progress. Follow-up studies should review the impact to students after beginning employment in information systems related fields to determine the success of the Agile pedagogy relative to its application in industry. Evaluation of specific Agile tools was not performed as part of this study and could be further considered to enhance the instructional value.

7. REFERENCES

FFIEC (n.d.). Computer-Aided Software Engineering. FFIEC IT Examination Handbook Infobase. Retrieved from https://ithandbook.ffiec.gov/it-booklets/development-and-acquisition/development-procedures/software-development-techniques/computer-aided-software-engineering.aspx

Frydenberg, M., Yates, D., & Kukesh, J. (2018). Sprint, then Fly: Teaching Agile Methodologies with Paper Airplanes. *Information Systems Education Journal*, *16*(5), 22.

Hailpern, B., & Tarr, P. (2006). Model-driven development: The good, the bad, and the ugly. *IBM systems journal*, *45*(3), 451-461.

Jones, W. (2002). Case tool time. Retrieved from http://www.umsl.edu/~sauterv/analysis/488_f02_papers/CASE.html

Henkel, M., & Stirna, J. (2010, September). Pondering on the key functionality of model driven development tools: the case of mendix. In *International Conference on Business Informatics Research* (pp. 146-160). Springer, Berlin, Heidelberg.

Linden, T. (2018). Scrum-based learning environment: Fostering self-regulated learning. *Journal of Information Systems Education, 29(2)*, 65-74.

Moe, N. B., Dingsoyr, T., & Dyba, T. (2009, November 20). A teamwork model for understanding an agile team: A case study of a Scrum project. *Information and Software Technology, 52,* 480-491.

Niess, M. L., & Gillow-Wiles, H. (2017). Expanding teachers' technological pedagogical reasoning with a systems pedagogical approach. Australasian Journal of Educational Technology, 33(3), 77–95.

Rico, D. F., & Sayani, H. H. (2009). The Business Value of Agile Softare Methods: Maximizing ROI with Just-in-Time Process and Documentation. Fort Lauderdale: J. Ross Publishing.

Schmidt, D. C. (2006). Model-driven engineering. Computer-IEEE Computer Society, 39(2), 25.

Selic, B. (2003). The pragmatics of model-driven development. *IEEE software*, *20*(5), 19-25.

Teichroew, D., & Hershey, E. A. (1977). PSL/PSA: A computer-aided technique for structured documentation and analysis of information processing systems. IEEE transactions on software engineering, (1), 41-48.

Tuckman, B. W. (1965). Developmental sequence in small groups. *Psychological bulletin*, *63*(6), 384.

Yourdon, Ed (Jul 23, 2001). Can XP Projects Grow? *Computerworld,* 35(30), 28.

Appendix

Project - *this is the semester project. It will be graded on functionality, structure, adherence to requirements, etc. according to the rubric provided.*	**20%**
Sprint Deliverables – *User Stories, Domain Model, Business Logic, User Interface, Security and Application Deployment. Each deliverable increasingly incorporates Agile and Mendix concepts, and will be graded on these competencies separately according to rubrics provided.*	**60%** *(10% each - 5% Agile, 5%* Mendix)
Mendix Certification – *Students are required to become Mendix Rapid Developer certified and provide documentation of certification.*	**10%**
Presentation – *Students will demonstrate and present their projects.*	**10%**
Total	**100%**

Table 1
Grading Distribution

ISSN: 2473-4901
v5 n4951

Course Retrospective			
	What did we do well?	**What should we keep doing?**	**What should we change?**
Communication & Collaboration	Teams had consistent communication and collaboration among team members. Time was provided during class to work on the project. Teams had the ability to plan and resolve technical issues during class with the professors.	Direct feedback was given during class.	Include a working version of a Mendix app for student reference. Six weeks was not sufficient for fully learning Mendix. Reference exact pages in books and other materials during class.
Agile Pedagogy & Tools	GitHub's ZenHub project board allowed teams to assign stories to team members, giving everyone a responsibility and ownership. Ability to prioritize work using Agile and stories for clear definition of work. Agile provided a clear framework for teams to understand weekly expectations. Agile assisted the team in the ability to complete the project by providing motivation and the division of duties among the team members.	ZenHub boards in GitHub provided more organization for story tracking and should continue to be used for the Agile process rather than the Mendix platform's story tracking. Teams assigned by professors prior to the start of the class. Team roles assigned by professors prior to the start of the class. Some students were forced to develop team managerial responsibilities who would otherwise not have chosen this role. The order of the weekly class facilitated progress: sprint review, retrospective, Mendix tutorial, sprint planning.	Requiring the Mendix Developer Certification as the first Mendix deliverable during week 1 was difficult to complete. Have one professor rather than two in future offerings.
Mendix Tool	The Mendix support team was available for technical issues and questions. Weekly Mendix tutorials assisted students with progressing in their application development.	The course as an Agile project was hands-on and increased learning of the tools and processes.	Tutorials provided by Mendix skip some steps making it difficult to find information when facing technical issues. Provide a clearer definition of the application's requirements (e.g. what is considered a microflow).

Table 2
Retrospective results of Agile low-code development course