Journal of Information Technology Education: Innovations in Practice

An Official Publication of the Informing Science Institute InformingScience.org

JITEiip.org

Volume 21, 2022

USING DESIGN-BASED RESEARCH TO LAYER CAREER-LIKE EXPERIENCES ONTO SOFTWARE DEVELOPMENT COURSES

Christine Bakke*	University of Minnesota, Crookston, MN, United States	cbakke@crk.umn.edu
Rena Sakai	University of Minnesota, Crookston, MN, United States	sakai017@crk.umn.edu

* Corresponding author

ABSTRACT

Aim/Purpose	This research aims to describe layering of career-like experiences over existing curriculum to improve perceived educational value.
Background	Feedback from students and regional businesses showed a clear need to increase student's exposure to career-like software development projects. The initial goal was to develop an instructor-optional project that could be used in a single mid-level programming course; however, the pilot quickly morphed into a multi-year study examining the feasibility of agile projects in a variety of settings.
Methodology	Over the course of four years, an agile project was honed through repeated Design Based Research (DBR) cycles of design, implementation, testing, communication, and reflective analysis. As is common with DBR, this study did not follow single methodology design; instead, analysis of data coupled with review of literature led to exploration and testing of a variety of methodologies. The review phase of each cycle included examination of best practices and methodologies as determined by analysis of oral and written comments, weekly journals, instructor feedback, and surveys. As a result of participant feedback, the original project was expanded to a second project, which was tested in another Software Engineering (SE) course. The project included review and testing of many academic and professional methodologies, such as Student Ownership of Learning, Flipped Classroom, active learning, waterfall, agile, Scrum, and Kanban. The study was homogenous and quasi-experimental as the population consisted solely of software engineering majors taking required courses; as based on va-

Accepting Editor Benson Soong │ Received: March 31, 2022 │ Revised: May 25, June 2, 2022 │ Accepted: June 4, 2022.

Cite as: Bakke, C, & Sakai, R. (2022). Using design-based research to layer career-like experiences onto software development courses. *Journal of Information Technology Education: Innovations in Practice, 21,* 25-60. https://doi.org/10.28945/4988

lidity of homogenous studies, class sizes were small, ranging from 8 to 20 students. Close interactions between respondents and the instructor provided interview-like settings and immersive data capture in a natural environment. Further, the iterative development practices of DBR cycles, along with the inclusion of participants as active and valued stakeholders, was seen to align well with software development practitioner practices broadly known as agile.

Contribution This study is among the first to examine layering a career-like software development project on top of a course through alteration of traditional delivery, agile development, and without supplanting existing material.

Findings In response to industry recommendations for additional career-like experiences, a standalone agile capstone-like project was designed that could be layered over an existing course. Pilot data reflected positive perceptions of the project, although students did not have enough time to develop a working prototype in addition to completing existing course materials. Participant feedback led to simultaneous development of a second, similar project. DBR examination of both projects resulted in a simplified design and the ability to develop a working prototype, if and only if the instructor was willing to make adjustments to delivery.

After four years, a solution was developed that is both stable and flexible. The solution met the original charge in that it required course delivery, not course material, to be adjusted. It is critical to note that when a working prototype is desired, a portion of the lecture should be flipped allowing more time for guided instruction through project-focused active learning and study group requirements. The results support agile for standalone software development projects, as long as passive delivery methods are correspondingly reduced.

Recommendations for Practitioners Based on the findings, implementation of a career-like software development project can be well received as long as active learning components are also developed. Multiple cycles of DBR are recommended if future researchers wish to customize instructional delivery and develop complex software development projects. Programming instructors are recommended to explore hybrid delivery to support development of agile career-like experiences.

Small class sizes allowed the researchers to maintain an interview-like setting throughout the study and future studies with larger classes are recommended to include additional subject matter experts such as graduate students as interaction with a subject matter expert was highly valued by students.

Recommendations for Researchers Researchers are recommended to further examine career-like software development experiences that combine active learning with agile methods; more studies following agile and active learning are needed to address the challenges faced when complex software development is taught in academic settings.

Further testing of standalone agile project development has now occurred in medium sized in person classes, online classes, independent studies, and creative works research settings; however, further research is needed. Future research should also examine the implementation of agile projects in larger class sizes. Increasing class size should be coupled with additional subject matter experts such as graduate students.

Impact on Society This study addresses professional recommendations for development of agile career-like experiences at the undergraduate level. This study provides empirical evidence of programming projects that can be layered over existing curriculum, with no additional cost to the students.

Initial feedback from local businesses and graduates, regarding agile projects with active learning, has been positive. The area business that refused to hire our underprepared SE graduates has now hired several.

Future Research Future research should explore layering agile projects over a broader range of software development courses. Feedback from hiring professionals and former students has been positive. It is also recommended that DBR be used to develop career-like experiences for online programming courses.

Keywords Agile, Scrum, software development, programming, Student Ownership of Learning, active learning, iterative development, information technology, software engineering

INTRODUCTION

In 2016, there was a complete turnover in our software engineering (SE) staff, at the same time a major IT employer in our region let our office know that our SE graduates were underprepared for programming careers. These events led the new staff to closely examine existing SE curriculum for alignment with both academics and professionals. At first glance the curriculum appeared to follow best practices through inclusion of the common technology core, multiple programming languages, two career-like experiences, and an emphasis on mathematics (Alperowitz et al., 2016; Mahnic, 2012; Rico & Syani, 2009; Schilling & Klamama, 2010). However, while our curriculum aligned with academic best practices, it did not align with professional recommendations. A meta-study by Sahin and Celikkan (2020) documented professionals in 24 countries finding underprepared programming graduates; further, the professionals recommended an academic shift toward agile and Scrum along with increasing technical discussions with academics, increasing exposure to project management, and incorporating teamwork requirements.

We noted that we were following recommendations of industry stakeholders and accreditation agencies, in that we included a career-like experience in an academic setting through a yearlong capstone sequence focused on developing complex software (Adkins & Tu, 2021; Alperowitz et al., 2016; Mahnic, 2012; Rico & Syani, 2009). A problem arose when it was noted that the capstone courses were designed around the traditional academic model of waterfall. In an examination of the rest of the SE courses we did not find any agile instruction, and, other than a few lab assignments, we found waterfall methodology and lecture-style instruction throughout.

Lack of software engineering preparedness in programming projects means that students need more career-like experiences, i.e., they need more practice developing software projects. This is different than lab experience where students encounter neat problems with clear answers. Complex software development is the messy, unknown development that occurs when a software developer is tasked with creating new and original software designed to address the specific needs of a client. To provide another such experience without the ability to add or change course, we developed an agile pilot project that would be added to a course. The goal of such a project is the same as the goal of semi-capstone and capstone courses; students are tasked with creating a unique software prototype from scratch.

There were some advantages to having the project design rely on agile, rather than waterfall. One of the main differences between agile and waterfall development is the use of iterative, cyclic development (agile), rather than forward-only development (waterfall). Our capstone is a two series course, the first course focuses on project requirements, the second on development of working software based on the project requirements. During the capstone experience, development teams meet with the instructor once a week to present progress and discuss challenges. Some instructors provide a guide for such meetings, but a weekly meeting guide is completely optional. The pilot modified one of the weekly meeting guides to incorporate cyclic development and agile terminology to begin the first cycle of Design Based Research (DBR).

As the cycles continued adjustments were made based on stakeholder feedback, mainly the weekly report was modified, but soon adjustments to delivery method were also seen as essential for successful prototype development. At our institution, instructors have choice of delivery methods, but curriculum is determined collaboratively. It was determined that, to keep content consistent between instructors, the agile project must be flexible enough to be layered onto an SE course without displacing existing course requirements. This led to examination of recent agile studies where we found development of working prototypes in mid-level programming courses; however, in each case, the researchers had created either a new course or re-designed an existing course so that the course focus was on the project (Corritore & Love, 2020; Magana et al., 2018; Maxim et al., 2017). None of the studies addressed our need for a standalone agile project. Was it possible to develop an agile-based project for a mid-level course that layered over an existing, traditional course? We were unable to find studies that mapped out a solution, so we turned to an active form of research known as Design Based Research (DBR), whereby we were able to iteratively collect, examine, and test the design in a real classroom setting.

Each DBR cycle relies on data and literature as the basis for the design that will be tested in the next cycle. The design, also known as an intervention, is tested, and collected data is reviewed to determine if a solution has been reached. The cyclic nature of DBR, combined with examination of data/literature and regular testing, make it a good fit for instructional development. Some of the challenges of incorporating DBR include multiple methodologies, extensive time commitment, and clear dissemination of data. DBR cycles start with identification of a problem, followed by repeated cycles of planning/design, implementation of the design, testing in a natural setting, communication with stakeholders and review of literature, and reflection and analysis. Each cycle culminates with a reflection stage where data is analyzed to determine whether a stable, flexible solution has been reached or if further adjustments are needed (McKenny & Reeves, 2013). DBR cycles included review of data alongside an examination of both academic and professional literature.

Throughout the study, research was guided by the following questions:

> **RQ1:** How can career-like experiences be successfully layered over existing course material?
>
> **RQ2:** What are student perceptions of courses with career-like experiences layered on top of existing course content?

This paper begins with an overview of relevant literature followed by four years of DBR iterations, including discussions of student survey responses and instructional observations. The research contributes to applied scholarship through development of a career-like complex software project that has the potential to be layered on top of existing course material. In alignment with DBR, the project did not rely on a single methodology, rather several project-focused methodologies were examined during each cycle. Participant feedback resulted in adjustments toward independent coding and testing of a second project design. The following literature review highlights only those academic and professional methodologies used to develop the proposed project solution: Student Ownership of Learning, Flipped Classroom, active learning, waterfall, agile, Scrum, and Kanban.

LITERATURE REVIEW

A survey of academics, IT professionals, and employers in 24 countries reported perceptions that programming instruction frequently placed too much emphasis on theory, while employers and professionals reported graduates needed additional training in soft skills, analytical thinking, hands on projects, teams, and project management. Professional recommendations for addressing these issues in an academic setting include increasing exposure to agile development, embedding newer technologies into the curriculum, providing interactive environments that encourage technical discussions between academics and students, and increasing experiences in project management and teamwork (Sahin & Celikkan, 2020). In the United States alone, there are around 1.5 million software developers

(Bureau of Labor Statistics, 2018), and the most common framework reported to be in use by professionals is agile, with industry adoption rates ranging between 71% (9th Global Project Management Survey, 2017) and 95% in 2020 (Digital.ai, 2020) and with 80% of major federal IT projects self-describing as either Agile or iterative (Viechnicki & Keikar, 2017). An examination of our courses revealed a need for change, as a majority of the instruction was through passive delivery and focused on the waterfall method.

First, we examined key factors in successful software development, as perceived by professional developers. In 2004, Steve McConnell, a professional programmer and author of Code Complete, described software development as often ill-defined and complex, insomuch that students and instructors face many challenges if they are to include complex projects in undergraduate education (2004, p.75). But what are these ill-defined and complex challenges? Beginning in 1985, the Standish Group began researching and collecting data on software engineering cases; today, they maintain a research database of over 50,000 software development projects from which they are able to provide premiere software development advice. In their Chaos report (Standish Group, 2015), the Standish Group listed just three critical criteria for successful software development: user involvement, executive management support, and clear requirements. This list can be compared to Sheffield's (2019) listing of 10 key factors for software development success: develop for intended users, create a detailed strategic plan, use a team of expert developers, require project planning, follow agile project management best practices, be clear and consistent in communication, create wireframes, develop a risk log with a corresponding action plan, follow best practices, and complete scheduled reviews. It can be seen that both lists highlight user involvement, managerial support, and clear communication through planning.

It was noted that the emphasis of professional developers had some overlap with academics. Academic agile software development literature was found to include voluntary, self-organized, cross functional teams in time-intensive academic settings (Zhang & Dorn, 2012); timeboxed competitions such as hackathons and coding bootcamps (Fronza et al., 2020; Gama et al., 2018) eXtreme Programming practices encountered during coding bootcamps (Fronza et al., 2020); and self-regulated learning (Parsons & MacCallum, 2019). Both professional and academic literature addressed complex software development by incorporating team approaches. As expected, professional software developers were using methods and tools that are in development, so that many are yet to be thoroughly tested and documented in academic settings.

With the foundational need for career-like projects already determined and an existing project as a model, the primary goal for the students was to design and build a working software prototype. We modeled the project after the capstone in order to provide a similar career-like experience at an introductory level, the main adjustment being agile methodologies, rather than waterfall, with the goal of experiencing the software development lifecycle at least one time. In order to create an agile project that emulates the career-like experience of the capstone, a pilot project was first tested in Introduction to Software Engineering. The project varied from that of Magana et al. (2018), which had redesigned an Introduction to Software Engineering course around a mid-level career-like experience, as we added the project onto existing material and did not alter the underlying course design. When the department determined that it was not desirable for a programming prerequisite to be enforced, we examined studies of courses designed expressly for beginning programmers (Corritore & Love, 2020; Figueiredo & García-Peñalvo; 2019).

Starting with the findings of previous studies, two frameworks formed the basis for the pilot project: Student Ownership of Learning (SOL) and agile methodologies. The researchers selected SOL for the pilot in order to provide an agile-like academic setting through meaningful learning and practical life skills through self-directed active participation. Agile was the initial professional framework; however, Agile's most popular method, Scrum, was found to be a better fit in our setting. No prior studies were found where researchers were able to add a capstone-like project onto a mid-level course, excepting studies where an entire course was modified or created with a project as the central focus of the course. The following sections review the primary methodologies used in the development of our agile projects.

SOFTWARE DEVELOPMENT METHODOLOGIES

A study of current research moved the pilot design away from waterfall and toward agile. Figure 1 shows a subset of data from a multi-year comparison of 50,000 IT cases (Standish Group, 2020) that revealed success rates for waterfall and project management to be significantly less than those of agile. When considering the best way to add a complex software development project to an existing course, it was crucial to note that professionals reported less challenges and greater successes with agile over all other methods.

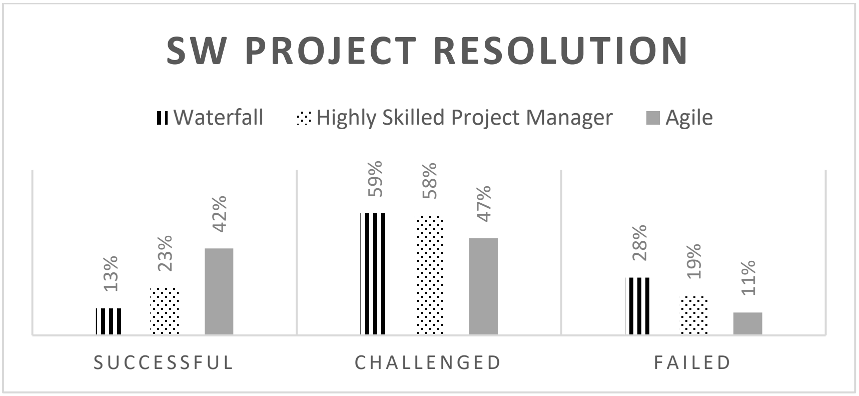


Figure 1: Software Project Resolution Rates
Authors' representation of data from the Standish Group (2020)

WATERFALL

Waterfall is a traditional academic model composed of distinct, sequential steps which is often misattributed to Royce (1970). In 1970, Royce recommended adjusting the then prevalent "waterfall" method toward a more iterative approach, but his adjustments were not implemented and the prevalent waterfall method continued. It is interesting to note that today's academic and business "waterfall" diagrams include multi-directional flow (Herawati et al., 2021; LucidChart, 2017; ReQtest, 2019). Traditionally, waterfall development isolates and forward-loads the requirements of a software development project. Although Royce is the first professional programmer to have proposed changes, he is not the last. With companies reporting the most common reasons for project failure being lack of user input, incomplete specifications, and changing requirements (Standish Group, 2015), it is clear that traditional waterfall methods should not be the primary software development instructional model.

AGILE PHILOSOPHY

Agile practices have been used for many years by professional software developers (Bakke, 2013; McConnell, 2004); however, it was not formally gathered into an organized philosophy until a group of 17 software developers drafted the Agile Manifesto in 2001 (Agile Alliance, n.d. -b). This published document defines the heart of the agile movement to prioritize individuals, working software, customer collaboration, and response to change over processes, abundant documentation, contracts, and extensive planning (Agile Alliance, n.d. -b). Over time, agile practices have been further defined and developed, so that agile now refers to a broad range of practices that emphasizes stakeholder involvement, soft skills, and creating value for the customer. Agile has become the most prevalent development philosophy, with private industry adoption rates ranging between 71% (9th Global Project

Management Survey, 2017) and 95% (Digital.ai, 2020); while 80% of major federal IT projects self-describe as either agile or iterative (Viechnicki & Keikar, 2017).

Agile is broad and, for the purposes of this study, it refers to a flexible software development philosophy that encourages customer communication, iterative development, and teamwork. A broad range of studies have been published on agile topics in both the academic and business realm from voluntary, self-organized, cross-functional teams in time-intensive academic settings (Zhang & Dorn, 2012) to timeboxed competitions such as hackathons and coding boot camps (Gama et al., 2018). Agile frameworks have been studied including eXtreme Programming practices during coding boot camps (Fronza et al., 2020) and self-regulated learning (Parsons & MacCallum, 2019).

The study began by examining the agile glossary, a dictionary of the most popular agile frameworks, tools, and practices (Agile Alliance, n.d. -a.). This led to a "library" approach, whereby the online professional community became accepted as mentors, able to provide trouble-shooting assistance, resources, and coding tips as students worked to develop custom projects. DBR cycles led to simplification of the design to primarily focus on Scrum, an agile framework which had been previously studied in both project-based learning (Dinis-Carvalho et al., 2018; Saadé & Shah, 2016) and semi-capstone experiences (Magana et al., 2018; Maxim et al., 2017).

SCRUM FRAMEWORK

Originally developed by Sutherland and Schwaber, the Scrum framework (Figure 2) involves regular collaboration between management, stakeholders, and the development team; these interactions guide decision making, reduce waste, emphasize essentials, and place value on experiences (Scrum Guides, 2017). Over 70% of agile companies report Scrum to be their preferred framework because of its strengths in managing changing priorities (70%), business / IT alignment (65%), delivery speed (60%), team morale (59%), increased team productivity (58%), project predictability (50%), software quality (46%), engineering discipline (44%) (Digital.ai., 2020).

Scrum is designed to guide teams in the development of software solutions; it is lightweight and helps teams generate value as they work to solve complex problems (Digital.ai., 2020). Scrum incrementally moderates project risk through teamwork and communication, tracking project progress through backlogs, daily stand-ups, and actionable items. Scrum timeframes (sprints) are fixed but may range from one week to one month depending on company preferences. During each sprint, project tasks are organized, prioritized, and addressed through the four key events of team planning, regular meetings, product reviews, and retrospectives. An example of the Scrum framework adapted to an academic setting can be seen in Figure 2.

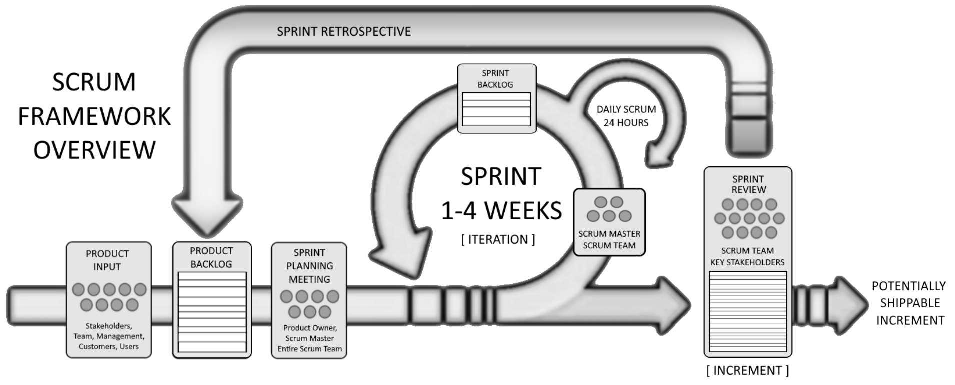


Figure 2: Scrum Framework Overview
Source: Authors' interpretation

Team planning takes up no more than eight hours per sprint and involves communication with stakeholders to clarify and prioritize sprint goals and product backlog. Each day, the scrum team meets for a daily stand-up where they briefly discuss progress and challenges. A practice known as "three questions" helps to guide team members through a discussion of accomplishments, challenges, and upcoming tasks (Scrum.org, n.d.). Three questions can be worded in many different ways, for example:

1. What did you complete yesterday?
2. What do you plan to complete by tomorrow?
3. Is there anything blocking your progress?

Scrum emphasizes flexibility by valuing stakeholder input, encouraging adjustments to requirements, and self-analysis (Scrum.org, n.d.). Scrum teams are flexible and adjust quickly to project changes through artifact creation, backlog tracking, viability testing, and regular stakeholder communication. Scrum practices rely on feedback, clearly defined roles, and priorities through goal setting, team organization, timeboxed increments, real-life experiences, and reduction of waste (California Project Management Office [CA-PMO], 2017). During development, teams regularly hold product reviews and team retrospectives during which they present the results of each sprint to stakeholders and management. During a product review teams rely on stakeholder feedback to adjust project backlog and determine priorities for the next sprint. Such changes may also result in managerial issues such as budget adjustments. While reviews focus on the product, retrospectives focus on improving the team's quality and effectiveness. Retrospectives help the team discuss successes and note areas for improvement (Scrum Guides, 2017).

Kanban

Kanban is an agile tool for managing project workflow that has been documented to increase team productivity by providing an overview of tasks while helping to minimize waste and balance availability of resources (Kanbanize, n.d.). Kanban boards provide teams with real-time task visualization of project tasks and rapid assessment of productivity bottlenecks. Teams and management benefit from this simple communication tool which displays a snapshot of current tasks; quickly revealing project needs, successes, and challenges. Kanban cards and columns help teams organize a complex project into a single chart that can quickly provide an overview of all major backlog tasks. A representation of a Kanban board that students might set up for their projects is shown in Figure 3.

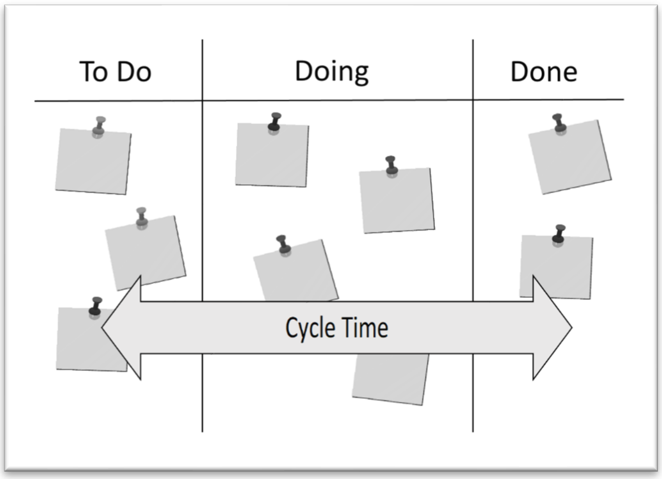


Figure 3: Kanban board
Source: Authors' interpretation

Kanban boards were modeled after industry, so that teams and instructors could quickly see an overview of both individual and team progress. Kanban is a simple and efficient tool that helps teams visualize the entire project, improve lead time estimates, and quickly see both challenges and flow. Management also benefits as the entire project is tracked over time and an entire overview of the project can be viewed at any time (Radigan, 2019). Two professional Kanban boards were found to be free for small groups: Jira and Trello. Teams followed professional practice through use of professional tools, and instructors found that Kanban charts were simple, needing little explanation. Project implementation mirrors professional teams in that each person moves their backlog items across a team progress chart divided into meaningful categories such as do, doing, and done. Because teams use a single board, a single snapshot provides assessment documentation for everyone on the team.

ACADEMIC RESEARCH METHODOLOGIES

SELF-SELECTION

In an academic setting, self-selection has been reported to be a stressor at the onset of a project; however, as students become vested, the freedom to make project decisions is appreciated and seen as beneficial (Md Rejab et al, 2019). This finding is similar to feedback from professional agile developers who state that the fastest and most efficient way to form productive, small, cross-functional software development teams is with facilitated self-selection (Mamoli & Mole, 2014).

Self-selection can also be practiced through teams that self-organize tasks. It has been shown that self-organizing teams more readily adapt to changes when they are given authority to make decisions as they work together toward a shared goal (Mamoli & Mole, 2016). Using a bottom-up estimation and planning, self-organized teams lead the decision-making process. This is seen at the sprint level by peer balancing of workload, teams that proactively address tasks, and team identification of hours needed. Additionally, it is common for professionals to employ a Scrum master to provide any needed education, facilitation, and guidance (Mandonca, 2016).

DESIGN-BASED RESEARCH (DBR)

Design-Based Research (DBR) is a form of active research whereby researchers iteratively test and refine “curricula, practices, software, or tangible objects beneficial to the learning process” (Armstrong et al., 2018), but may also examine the intangible such as contextually developed claims to better understand or advance theory (Collins et al., 2004). In DBR, participants play an active and integral role, which is much different from studies that view participants as subjects to be observed or experimented on. Throughout the research process DBR subjects are immersed in the study through collaborative key roles that encourage them to share in the investigation and improvement process (Barab et al., 2004, p.3).

A DBR study is initiated by defining a problem, its context, and relevant theories (McKenny & Reeves, 2013). Clear problem identification starts a cycle of collaborations between practitioners and participants in an effort to better understand, document, and evaluate the issues. Based on initial analysis, an intervention is designed and implemented. During implementation, data is often collected through methods such as observations, surveys, or interviews. DBR encourages data authentication through participant interaction in a natural setting along with communication and testing. At the completion of each cycle, a period of reflection examines relevant literature to assure best practices have been incorporated into the design. Researchers reflect on the study in an attempt to connect actions with results as they modify each intervention with the goal of developing a flexible, stable solution (Armstrong et al., 2018). During the review phase, collaborators determine if the design needs further modifications, or if a solution has been reached (Design-Based Research Collective, 2003, pp. 5–6). A representation of the DBR cycle, as employed in this study, can be seen in Figure 4.

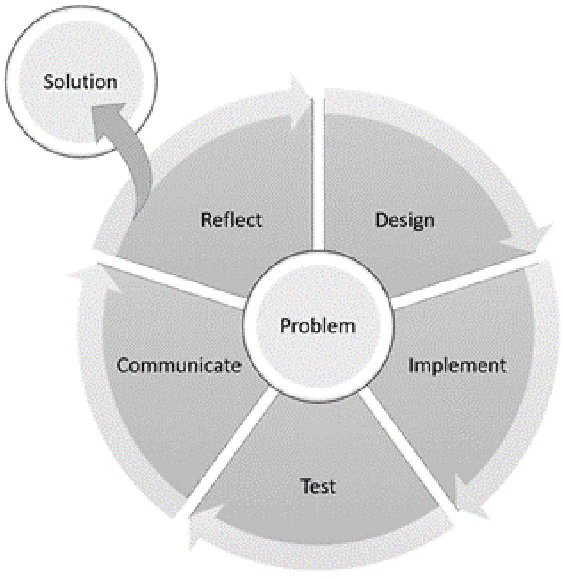


Figure 4. Design-Based Research Cycle
Source: Authors

Active Learning

When pilot data revealed a need for additional in-class activities to clarify connections between the project and the course material, a methodology was sought that could guide us through an impossible task: increase course value without adding time. We were already adding a project, now we were asked to add in-class activities. We found a possible solution in project-based studies that replaced lectures with active learning, and found they increased perceptions of educational value (Freeman et al, 2014; Pattanaphanchai, 2019). This meant we would need to cut back on lecture time. This led to a meta-analysis where active learning was found to be significantly more effective than traditional methods regardless of class size, student quality, or instructor identity; so much so that students in traditional lecture courses were found to be 150% more likely to fail than those in courses delivered through active learning methods (Freeman et al, 2014).

These studies led to development of hybrid lessons that combined traditional lecture with active learning. Active learning involves students in dynamic and creative activities such as group work, problem-solving, presentations, cooperative learning, case studies, and classroom responses (Klinger et al., 2022). Lecture instruction had been shown to be destructive to STEM so much so that failure rates were 55% greater in lecture classes over the same course delivered with some active learning components. If this failure rate were to be extrapolated to count each failure uniquely, it would calculate into 840,000 STEM course failures, simply because the courses do not include active components (National Science Foundation, 2014). Active learning tools are well documented in academic literature through studies on peer-assisted learning, cooperative learning, problem-based learning, group presentations, a think-pair-exercise discussion, and classroom response tools such as clickers (Bishop & Verleger 2013; Hawks 2014; Talbert & Mor-Avi, 2019).

Active learning generally refers to any form of learning that engages students in doing something other than listening (Duffany, 2015). Case studies are a form of active learning that encourages problem solving, real experiences, and interactions that increase content discussions between learners and content experts (Maxim et al., 2017). When active learning is sustainable, it has been shown to provide more value than traditional lectures and assessments (Brown et al., 2016; Wong et al., 2017). Another benefit to active learning was experienced by minorities and first-generation students who were seen to increase time spent preparing for class while feeling a greater sense of community (Eddy &

Hogan, 2014); these benefits are similar to those seen in capstone-like experiences (Jazayeri, 2015; Marques et al., 2018).

In an academic setting, facilitators of active learning provide formative feedback and pose complex problems that encourage examination of multiple sources (Carless & Boud, 2018). In a professional agile environment, active learning is manifested through self-organizing teams which in turn increases ownership and empowers decision making (Lemos et al., 2014; Mamoli & Mole, 2016). By designing our projects and activities around agile, we could expect to see a rise in peer engagement along with increases in creativity, flexibility, and self-directed learning (European Commission, 2020.

Student Ownership of Learning (SOL)

SOL is an applied learning framework rooted in the works of Dewey (1966). In SOL students' skills are developed starting with an instructional shift toward facilitation, directed learning, and student engagement. Beginning with a level of understanding, students' progress to a level of ownership, which is demonstrated through the ability to articulate strategies and outcomes of personal learning. The apex of student ownership can be seen when teachers and students collaboratively facilitate learning through a mutual exchange of ideas and strategies (National Science Foundation, 2014).

Courses that incorporate SOL increase student engagement by building on existing experiences and knowledge while encouraging student choices through project identification, goal setting, and prioritization of project tasks. As students develop expertise and confidence in their abilities through repeated interactions with projects, ideas, and important concepts, they demonstrate content mastery and project ownership (Corritore & Love, 2020).

Such an environment is created when instructors co-facilitate the learning process by engaging students in active leadership and self-directed learning (National Institute for Excellence in Teaching [NIET], 2021). Ownership of learning is typically observed in active learning settings that allow students to generate new knowledge as they become self-directed (Graus et al., 2022). To facilitate student ownership of learning, the role of the teacher must change from lecturer to coach. A teacher in this role becomes a valued resource, helping students achieve goals through active participation in decisions, choices, content application, observations, and evaluations (Chan et al., 2014; Weekly Tip, 2019).

Methodology

Background of the Problem

Software development is at the heart of SE; in other words, the goal of software engineering is to develop software. The SE program requires students to complete two career-like experiences: an internship and a capstone. Periodically, we examine our curriculum in light of professional trends as reported by researchers and area businesses. As a result of the examination process, we began a multi-year DBR study focused on devising another career-like project, modeled after our existing capstone. The goal was to develop a mid-level project whereby students would develop a complex software project from start to finish; this career-like experience would be similar to the capstone but would employ industry best practices of agile, rather than our current academic instructional methodology: waterfall. In technical fields, capstone projects are designed to emulate career experiences; throughout this study career-like experiences refer to any project where students experience the entire software development life cycle. To avoid adding graduation requirements or changing course material, it was decided that a successful project would be one that could be layered over an existing course without significant adjustments to the underlying material. It was also determined that such projects must remain instructor optional. Perceptions of participants were tracked to assure students were not overwhelmed by the addition of a complex project. The use of participant perceptions as a critical factor in determining the success of the project is grounded in DBR theory where participants are viewed as stakeholders and co-researchers, rather than subjects to be observed and documented by an outside specialist. In valuing participant feedback, researchers were led to examine active learning methodologies which often occur in natural settings and have been shown to increase participant initiative, decision making skills, and intellectual engagement (BU Center for Teaching and Learning, n.d.). We also considered the role teamwork might play, as professional developers often complete complex projects by working in teams. In academic settings adequate teacher-student ratios, cooperative learning, versatility, and quality support are seen to build student leadership skills (Mendo-Lázaro et al, 2018). In professional settings, the physical work location of team members is often flexible with 81% of software developers reporting they work with team members at different locations and 71% of co-located developers reporting working in geographically dispersed teams (Digital.ai, 2020). To increase the likelihood of success, the pilot was deployed in Introduction to Software Engineering, a mid-level course, covering the entire SDLC and requiring a foundational knowledge of object-oriented programming. The original project was designed as a simplified capstone in order to emulate a career-like experience. Concerns that the project would be perceived to add a significant work load were addressed by employing DBR methods and small class sizes.

When pilot data was analyzed, the instructor was encouraged to further develop a second project for another course, so that there were times when two DBR interventions occurred in the same cycle. Because another mid-level course with a clear correlation to the entire SDLC did not exist, development of a second complex agile project was placed in the next course that the students would encounter: Software Construction. There were many challenges to including an agile project in this course as it emphasized the low-level design documents of waterfall development. While it was expected that the additional workload created by the addition of a software development project would play a significant role in feedback, the additional workload was never reported as a concern. Rather, participants recommended adjustments to course delivery in order to provide an overlap between the project and course material. It is important to note that, while course delivery was adjusted, no material was removed. Additionally, the adjustment to course delivery coincided with major project successes that were previously unseen: nearly all students were able to successfully demonstrate working prototypes.

The study was homogenous and quasi-experimental as the population consisted solely of software engineering majors taking required courses. During each cycle, surveys were offered to all students for minimal extra credit points. Class sizes ranged from 8 to 20 students; with an overall average of 11.7. The validity of the study data is consistent with the findings of Boddy (2016) where homogeneous settings of 12 result in saturation. Repeated data collections increased data validity over the course of seven cycles; with three interventions conducted in the first course and four in the second. Case studies that allow close interactions between respondents and instructor provided us with an interview-like settings that allowed in-depth study and immersive data capture in a naturalistic, inquiry setting (Crouch & McKenzie, 2006). Further, the expansion of the study to two sequential courses strengthened case-oriented analysis (Sandelowski, 1996).

Project Goals

In an effort to address recommendations for increased professional development, the initial goal was to develop a complex project that could be overlaid on a mid-level programming course (Peters & Tripp, 1977; Rittel & Webber, 1973). Data from the pilot supported further development, as well as revealing a desire for a second project; this led to two projects rather than one. Over the course of four years and seven iterations, two questions remained the driving factors for project development:

RQ1: How can career-like experiences be successfully layered over existing course material?

RQ2: What are student perceptions of courses with career-like experiences layered on top of existing course content?

Because the courses already existed, we developed the career-like project based on the need for software developers to make working software, thus the goal of a career-like experience for a software developer is at least a working prototype. As the project was developed to be layered over an existing course and was intended to be assigned as a mid-level experience, a successful career-like experience would be if a majority of the students developed working prototypes. Further, a successful project should be feasible for students, in so much that students not only report the project as valuable but also do not reflect that the course load is too much. Thus, the success of a career-like experience would be gauged based on completion of prototypes and perceived course value.

The researchers viewed participant feedback as critical to the success of the study, so that DBR was well aligned with project goals. When a significant project is added to only one section of a course, students' perceptions are viewed as a critical driving factor in the courses success. DBR involves all stakeholders, not as subjects to be observed, but as valued fellow researchers. Students were told that their feedback was valued and their recommendations would be examined during the development process, and many were able to later observe how their feedback had been implemented when the projects were introduced to other courses. The researchers placed value on student perceptions of the course in order to guide against excessive workload.

The pilot was designed to provide students with a mid-level capstone experience that would emphasize teamwork, soft skills, and all, or nearly all, of the software development life cycle (SDLC). The first two cycles found that the addition of a complex software development project on top of existing curricula was problematic, and perhaps impossible. Yet, by the third DBR cycle, implementation of stakeholder recommendations brought unexpected success to students. The goals remained the same throughout DBR cycles, and project development progressed through a series of refinements based on feedback combined with an examination of the literature. In the course of analysis and testing, delivery methods came under scrutiny for best practice. As a result, it is recommended that when projects are to be overlaid onto existing curriculum, delivery adjustments away from passive learning should be planned. Repeated examination and adjustments led to an agile software development experience that has been well received by students.

DBR Findings

Pilot Intervention

A sophomore-level course, Introduction to Software Engineering, was selected for the pilot project, as the course relied on traditional-style instruction, covered the entire software development lifecycle, and required prior knowledge of object-oriented programming. The initial project goal was to provide an introduction to the complexities of a career-like experience steeped in Student Ownership of Learning (SOL) and agile practices. Table 1 shows how the capstone experience incorporated current literature, with notable influences from Corritore & Love (2020) and NIET (2021).

Table 1. Pilot Design of Mid-level Career-Like Project

Intro to Web Development	Into to SE	SE Capstone
• Weekly presentations • Independent coding • Instructor determines weekly requirements • No teams, no group work • No prerequisite	• Random selection of small group (3-4 students) • Weekly presentations • Weekly team meeting • One team project. • Individuals determine and prioritize tasks • Division of project into "equal", independent parts.	• Self-selected teams of 3-4 • Weekly team meetings • Weekly instructor meeting • One team project. • The team determines tasks, goals, and priorities. • Iterative development • Pre-requisite: senior status

Intro to Web Development	**Into to SE**	**SE Capstone**
	• Iterative development • Prerequisite: object-oriented programming	
SOL: Thinking Teachers integrate weekly team communication and reflection tools. Students apply their learning to a real software development project.	**SOL: Control** Self-selection of • Software project • Determine project features • Set goals • Prioritize project tasks	**SOL: Problem-Solving** Team: set project and weekly goals. Individual problem solving Team meetings: team problem-solving.
Agile in Practice: Scrum Iterative development, rotating scrum master Presentations: share project; discussions with team and class	**Agile Tool:** Stand-up, goals, three questions, backlog, artifacts Facilitation: random selection of team members by the instructor, the project was limited to designing desktop software.	**Agile Terminology** Artifact, backlog, Definition of Done, Scrum, Agile

DBR Iteration One: Fall 2018

Prior to the start of class, an agile-like template was developed as a guide for incremental software development. During the first week of class, students were tasked with proposing unique software that might be created; then, the instructor assigned them to development teams based on similarity of proposals. Each team discussed the proposed projects of the members and a single project was then selected for each team. Students were then tasked with dividing their project into independent parts, so that the parts might later be put together into a single product. This varies slightly from our senior project, where the team sets their own goals and divides the project as they wish.

Throughout the semester, teams met weekly to correlate and troubleshoot their project, and the small class size allowed teams to briefly report in class. The value of small class size and weekly presentations soon became apparent when several students said they were struggling because they had not completed the programming prerequisites. At this point, a discussion took place to consider dropping the project from the course and replacing it with a more traditional requirement or continuing. The instructor was surprised when students overwhelmingly requested to continue project development.

The discussion led to adjustments that the researchers hoped would provide a level of success for the students who faced the challenge of developing software with no prior programming experience. The primary recommendation was for a portion of each lecture to be replaced with in-class assignments that would guide project development; this was possible from a curriculum standpoint as the course covered the entire software development life cycle, however, it also led to an additional workload for the instructor. From this point on, whenever possible, the instructor developed in class activities that aligned with both the project and the curriculum.

In this way, the interview-like setting of the pilot led to adjustments not only to the project, but also to course delivery. After the discussion, active learning began to be viewed as an essential component of any course that included a complex software development project. Throughout the course the small class size led to many informal discussions, which provided insights into the cycle of software development for new programmers. This allowed the instructor to create timely curriculum-based activities that complimented the students' project development needs.

Many challenges had been encountered in the pilot. The small class size was ideal for testing such a project and led to many informal discussions where it was often observed that students had selected a major they were not only interested in, but passionate about. The end of course survey listed in Appendix A, was offered to the students in order to collect their thoughts and recommendations regarding the value of the project. Responses revealed common threads of concerns and recommendations, and representative participant responses are listed below.

ADDRESSING RESEARCH QUESTIONS

RQ1: How can career-like experiences be successfully layered over existing course material?

Question: As an instructor, what can I do to structure the student-to-student learning, so it is more beneficial?

- *It is more beneficial if instructor selects the "best fit" groups, rather than random selection.*
- *Give category options for app development, and place students into groups by their category selection.*
- *Give some goals for students to accomplish in group cooperation time.*

Question: Do you have suggestions or ideas for changing or improving assignments in this course? Please explain.

- *Assign an app to develop that will introduce students to different aspects of mobile app development.*
- *Teach more about Android studio.*
- *Keep presentations and give the students more to work off of so they can actually start getting more in depth with their projects.*

Instructor observations:

- The course required object-oriented programming; however, many students had not taken the prerequisite; students without the prerequisite struggled the most.
- Poor attendance at study group meetings and no prior programming experience were the most commonly reported group problems.
- There were several requests for increased guidance.
- Some students were able to get a portion of their software working.

DBR notes: A new intervention would be designed to address the main themes of student and instructor feedback. Instructor feedback includes feedback from independent research projects where students provided feedback on ways to improve the tools used throughout this study.

RQ2: What are student perceptions of courses with career-like experiences layered on top of existing course content?

Question: Select all that apply to your learning in this class: [I learned a lot in this class; I learned a few things in this class that I did not already know; I did not learn anything new in this class]

- *I learned a lot in this class (83%)*
- *I learned a few things in this class that I did not already know. (17%)*
- *No negative responses*

Question: I would recommend this course to other students? 97% (yes)

DBR notes: students reported value for a project that was layered over existing material; however, additional iterations are needed to address feedback that pertains to RQ1.

Methodology Review

DBR involves methodological review as part of the design modification process. The pilot revealed a need for further examination of active learning methods and agile practices. Student feedback led to more specific project guidance as well as the creation of in-class active learning assignments. Current literature was examined to address feedback requesting adjustments to the teamwork selection process and the high levels of team friction. It was found that the instructor arbitration can be minimized by shifting some of the management responsibilities to the team (Mamoli & Mole, 2016).

Both academics and professionals report self-selection to be a positive experience; however, approaches to self-selection differ. In the professional realm facilitated self-selection is preferred, while the academic study examined self-selection without facilitation. The participant recommendations aligned not with the prior academic study, but with the professional preference for facilitated self-selection, so that it was decided the next cycle would design and test self-selection similar to a professional environment. This change to a more active instructor role was hoped to result in improvements to productivity, team motivation, and group stability (Agile Alliance, 2021).

Alterations were made to adjust for a second level project to be incorporated in to the next course in sequence, Software Construction. The first course would also be adjusted by incorporating additional guidance for those who may be struggling with foundational skills (Corritore & Love, 2020; Figueiredo & García-Peñalvo, 2019). The next curriculum cycle offered both of the targeted courses: Software Construction (SE 2300) and its prerequisite, Introduction to Software Engineering (SE 2200) using the designs shown in Table 2. The two altered designs were implemented in tandem which provided a unique opportunity to observe both project variations at the same time.

Table 2. Two-Course Intervention

Into to SE (pilot)	**New Intervention**
• Random selection of small groups • Weekly presentations • Weekly team meeting • One team project with the division of the project into "equal", independent parts. • Individuals determine and prioritize tasks • Project: game development	• Self-selection of a preferred project from a project list • Teams determined by instructor based on self-selected project choice(s) • Weekly presentations • Project: app development
Agile Active Learning Add group meeting guidance by incorporating agile practices, terminology, and tools **Scrum** • Addition of three questions	

Iteration Two: Spring and Fall 2019

The spring semester contained slight adjustments as few students provided meaningful data during the COVID disruptions. Due to the minimal responses, spring data was combined with fall, which had returned to in-person instruction resulting in more robust feedback. Minimal feedback during spring led to analysis of RQ1 only; during fall analysis was complete allowing analysis of both RQ1 and RQ2.

Combined feedback: SE 2200 & SE 2300, representative student comments

RQ1: How can career-like experiences be successfully layered over existing course material?

- *Explain portions of code*
- *Allow more time to work on in class activities.*
- *Reinforcing and making the connection between course material and the project early on would be helpful.*
- *Less emphasis on coding hours or more emphasis on analysis and design in the beginning may help reinforce the software development life cycle.*

Feedback from the COVID semester led to further development of embedded active learning experiences as well as further experimentation with the frequency of presentations and team meetings, which had been highlighted in the pilot data. In this way, the benefit of COVID was the gift of time to further develop agile components and active learning lessons. The intervention adjustments for both courses are shown in Table 3.

Table 3. Two-Course Intervention, Version 2

SE 2200	**SE 2300**
• Bi-weekly presentations • Bi-weekly team meetings • Add code discussion requirements to the weekly presentation • Game development Develop more active learning to more clearly connect the project to course material	• Four presentations • Weekly team meetings • Add code artifact and discussion requirements to weekly reports • App development • Develop more active learning to more clearly connect the project to course material
Agile Active Learning Add individual participation requirements to team meetings and group reports. Include rotating Scrum Master requirement. **Agile and Scrum** Change wording from Scrum three questions to Agile three questions	

Feedback from previous interventions recommended decreasing lecture time, so the new design in both courses emphasized further development of active learning lessons that would continue to build the connection between software development and the required course material. During this cycle, the use of professional agile resources greatly decreased time spent in active learning development. The following example shows an active lesson incorporating a common agile development technique known as Given When Then.

Active learning examples: Introduction to Software Engineering

Lesson focus: User Interface.

- Traditional lecture: Discussion of UI Best Practices.
- Active learning: students worked with their team to select color palettes, fonts, and menu layout; teams determined a cohesive interface design that would be used by all members.

Lessons on Software Construction, focus on code design and quality

- Traditional lecture: The importance of high-quality design
- Agile active learning: Given When Then (work with a team)
    - Create a Given When Then user story for your software
    - Each student's Given When Then is unique.

ADDRESSING RESEARCH QUESTIONS

Intro to Software Engineering: SE 2200, representative student comments

RQ1: How can career-like experiences be successfully layered over existing course material?

Question: As an instructor, what can I do to structure the student-to-student learning, so it is more beneficial?

- *Find a way to connect what we learn in class to what we work on outside of class.*
- *Presentation: doing it every other week is too much.*
- *Ask about group discussions.*

Question: Do you have suggestions or ideas for changing or improving assignments in this course? Please explain.

- *Activities were very efficient.*
- *Make it a rule to have only one meeting in each week.*
- *Honestly felt the class was engaging: less lecture time and more time for students to discuss in class.*
- *Teams need to agree develop on the same game engine instead of multiples, so it will be easier to cooperate with other classmates.*

Software Construction: SE 2300, representative student comments
Question: As an instructor, what can I do to structure the student-to-student learning, so it is more beneficial?

- *The only thing I can think of is that sometimes the instructions for how to format the presentations and what to include changed from time to time, but it's not a large problem.*
- *Nothing I can think of, you do a good job of mixing up the groups in class and the out of class meetings are run by us.* (a majority of the responses were similar to this or blank)
- *More guidance on how we could make our project better.*
- *Presentations are important. I like to share my work to my classmates, it's pretty cool, I have a sense of achievement when others see my project.*

Question: Do you have suggestions or ideas for changing or improving assignments in this course? Please explain.

- *More requirements for working with each other in a software engineering team.*
- *Require a team meeting once every two weeks instead of every week.*
- *Give more code exercises and examples.*
- *When classes just talk about textbook content, it is kind of boring.*

DBR notes: Changes were attributed to the addition of group guidance, instructor facilitation, and adding code discussion to presentations. The next intervention will adjust so that members in the same group will be working on similar projects. Presentations will be reduced to 4 per semester.

RQ2: What are student perceptions of courses with career-like experiences layered on top of existing course content?

Intro to Software Engineering: SE 2200, representative student comments
Question: Select all that apply to your learning in this class: [I learned a lot in this class; I learned a few things in this class that I did not already know; I did not learn anything new in this class]

- *I learned a lot in this class (67%)*
- *I learned a few things in this class that I did not already know. (22%)*
  - Two students selected both of the above
- *No negative responses*

Question: I would recommend this course to other students? 97% (yes)

Software Construction: SE 2300, representative student comments

Question: Select all that apply to your learning in this class: [I learned a lot in this class; I learned a few things in this class that I did not already know; I did not learn anything new in this class]

- *I learned a lot in this class (64%)*
- *I learned a few things in this class that I did not already know. (21%)*
    - Two students selected both of the above
- *No negative responses*

Question: I would recommend this course to other students? 95% (yes)

DBR notes: The data supports layering of a career-like project over existing, software development-related material. Additional DBR cycles are needed to address feedback to RQ1 to determine if the project is stable and flexible. Data for RQ2 will continue to be collected.

Instructor observations:

Bi-weekly meetings and reports were noticeably less productive than weekly reports and meetings; no significant project completion differences were noted between bi-weekly presentations and four presentations; however, four presentations were preferred by both participants and the instructor. The agile glossary (Agile Alliance, n.d. -b.) provided resources for many active learning lessons that were seen to clearly tie course material to student projects, with little or no modifications. A chart with the agile tools we incorporate most often is listed in Appendix B.

Methodology Review

The similarities between DBR and agile philosophies became apparent during this cycle with both employing cycles of iterative development in natural settings and placing a high value on stakeholder involvement (Agile Alliance, n.d. -b; Haagen-Schützenhöfer & Hopf, 2020). It appears that proponents of agile and DBR have independently created similar processes, designed around cyclic testing and valuing all participants. This correlation may be of interest to researchers. (See Table 4.)

The Agile Alliance glossary (Agile Alliance, n.d. -a) was seen as an invaluable aid to rapid development of active learning assignments for software development projects. We have included a listing of agile tools that easily translated to active learning lessons: agile manifesto, backlog, CRC cards, daily meeting, Definition of Done, do-doing-done, epic, extreme programming, facilitation, frequent releases, given when then, incremental development, iteration, iterative development, Kanban, Kanban board, Niko-Niko Calendar, Personas, Scrum, Scrum Master, Sprint, Sprint Backlog, Sprint Planning, Story Mapping, team, timebox, user stories, user story template, velocity.

Table 4. Adaptable Intervention

Intervention
• Implementation of Do, Doing, Done through a minimum of three goals per week, one achieved task, one in progress task, and one task not yet started. • Weekly team meetings required documentation and goal setting for the project, discussion of weekly progress, and trouble-shooting. • Weekly report: added artifacts of either video or picture. • Added code sample and explanation
Active Learning and Agile • Agile research and application were added to group requirements • Many active learning experiences incorporated agile or scrum practices.

Iteration Three: Spring 2020

Each person selected their top three projects from an instructor provided list. Based on student choices and instructor preferences, students were divided into study groups of three or four. Each student would develop a unique project; however, all projects in the same group would be on the same topic. For example, for those selecting Photoshop-like projects, all members needed to build unique filters, but no two filters would be the same. One student might develop an inversion filter and a color adjuster, while another could develop a pixilation and blur filter.

Grading became nearly independent, so that, while participation in group work was graded, the students each developed their own work. Individual grades were able to be based on individual submissions and participation in a study group. Individuals were graded on their own contribution and their own project development, so that this iteration involved breaking down requirements to the individual level.

There was also a change to presentation requirements, which freed up class time for more lecture and active learning. The length of the weekly report requirements increased to include project details, coding slides, and resources in an attempt to address requests for more structure and guidance. The group additions included studying Agile, Smart Goals, Burndown Chart, and the additional individual requirements included Smart Goals, Gantt Chart, user stories, backlog, velocity, bugs/fixes, an artifact, and velocity estimation. Students commented that the added requirements increased frustration rather than educational value, which resulted in adjustments to the weekly report requirements which allowed teams to research their choice of agile methodology and apply it to everyone's project. The individual report was simplified by having students address the three Scrum questions, goals were simplified to a Kanban-like overview of do, doing, done, and project code was either demonstrated or three coding terms were explained. For each of the four presentations, students would highlight their weekly journal along with a challenge. Spring 2020 resulted in a few changes to the report. Group meeting documentation included a paragraph of each project and an overview for each person during the weekly meeting and attendance in group meetings was noticed to be reflective of project quality.

Addressing Research Questions

Software Construction: SE 2300, representative student comments
RQ1: How can career-like experiences be successfully layered over existing course material?

Question: As an instructor, what can I do to structure the student-to-student learning, so it is more beneficial?

- *Continue with the same methodology of teaching.*
- *Going forward and assuming this type of classroom format will persist it would be helpful to putting students in the same time-zone into the same group. But, if the classroom format is going to be going back to normal, then allowing students to choose their group will be more beneficial for learning in my opinion. Yes it is true that you will have to work with people you don't like or don't work well with, but when learning how things work you're likely to learn better when you're learning with people you get along with and work well with.*

Question: Do you have suggestions or ideas for changing or improving assignments in this course? Please explain.

- *My project was very valuable, and I'm glad to have gotten a chance to get credit in class for something I wanted to do on my own.*
- *I think an improvement for this course would be to have a project/continuation of a project you're working on for the final instead of a final consisting of rote memorization. It's pretty clear that students learn the most from projects.*
- *I learned the most from making a project.*

- *Keep the project and keep us working as a team to solve problems together.*

DBR: Feedback was positive as summarized above. There were no comments for changing the project. A few adjustments were made, and DBR data was collected on one more cycle to assure adaptability and stability of layered career-like projects.

RQ2: What are student perceptions of courses with career-like experiences layered on top of existing course content?

Question: Select all that apply to your learning in this class: [I learned a lot in this class; I learned a few things in this class that I did not already know; I did not learn anything new in this class]

- *I learned a lot in this class. (40%)*
- *I learned a few things in this class that I did not already know. (40%)*
- *No negative responses*
- *Some students left this question blank*

Question: I would recommend this course to other students? 100% (yes)

DBR: Feedback continued to support development of active learning in conjunction with the project. In reviewing current literature, slight adjustments were made to project topic list to better align projects with the three of the recommendations of Fronza et al. (2020): *craftsmanship:* instructors should guide students from project inception through the release of a working project; *goal-setting:* instructors should facilitate teamwork by identifying high-level goals and breaking them down into clear tasks; *accountability*: instructor supervision is crucial to avoid one member sustaining a heavy workload.

METHODOLOGY REVIEW

The feedback from this cycle appeared to support a stable, yet flexible design. It was determined that data would be collected during one more cycle that incorporated the same interventions and active learning assignments. To assure flexibility had been reached, new project topics would be determined. What had seemed unattainable had been accomplished; students had developed a working prototype on top of existing curriculum. If student and instructor feedback during the next iteration continued to be supportive of the project and students developed working prototypes, the intervention would be considered a success.

ITERATION FOUR: SPRING 2021

For this iteration of Software Construction, a topic was selected for the entire class, but each project would continue to be unique. For this round, each student would develop an escape room based on an educational sub-topic of their choice such as history or math and targeted a different age group, for example, math – addition 1st grade, math – subtraction 3rd grade, math- division 4th grade, etc. There were no further changes to the course methodologies or projects.

To verify intervention stability and flexibility course material remained consistent with prior iterations and active learning remained in place. Analysis from implementation and testing phases resulted in no significant changes or recommendations for alterations. Group requirements and weekly reports continued to follow the template, and discussions guided study groups through common development steps. The course continued to require four presentations which were referred to as checkpoints. Once again, fully engaged students successfully developed a prototype game. All prototypes were able to be played by other students.

Once the project was determined to have satisfied the research questions and achieved both stability and flexibility, an overview could be developed and included. The examples and summaries included in Appendix C and D reflect previous cycles as a new theme is used each time the project is taught.

So far, this has resulted in software designs that are unique, without the need for client or stakeholder input. Appendix C includes an overview of the project along with example weekly report and presentation requirements. Appendix D includes previous software topics and representative discussion assignment examples.

Addressing Research Questions

Software Construction: SE 2300, representative student comments

RQ1: How can career-like experiences be successfully layered over existing course material?

Question: As an instructor, what can I do to structure the student-to-student learning, so it is more beneficial?

- *This way the learning effect is better because students are assigned to a group that suits them. Vice versa.*
- *Keep the weekly report. I like to learn this way.*
- *I enjoyed the coding project. I learned more about programing in a different application.*

Question: Do you have suggestions or ideas for changing or improving assignments in this course? Please explain.

- *Keep the independent learning and choices. I learned a lot about a coding language that was new to me.*
- *App development is a good thing to learn, and Android Studio.*

RQ2: What are student perceptions of courses with career-like experiences layered on top of existing course content?

Question: Select all that apply to your learning in this class: [I learned a lot in this class; I learned a few things in this class that I did not already know; I did not learn anything new in this class]

- *I learned a lot in this class (50 %)*
- *I learned a few things in this class that I did not already know. (50 %)*
- *No negative responses*

Question: I would recommend this course to other students? 100% (yes)

Discussion of Findings

Addressing Research Questions

After a review of both professional and academic literature, there were no studies found demonstrating stand-alone, agile software development projects that could be layered on top of an existing curriculum without re-designing a course. The pilot project was tested in Introduction to Software Engineering, where a small class size and an SDLC focused curriculum were hoped to align well with the project. The class size was small (eight students) which resulted in a natural, interview-like setting throughout the semester. This provided ample opportunities for instructor-student discourse to assure data accuracy (Crouch & McKenzie, 2006).

The design began by focusing academic methods on SOL and agile professional practices. Over time SOL was expanded to encompass the broader active learning framework and the most common agile framework, Scrum, was emphasized. To address project goals, a brief mixed-methods survey was developed which combined two open-ended questions with two quantitative questions tht were based on the Net Promotor Score (NPS) or the Ultimate Question (UQ). The UQ had been used in a previous academic study (Corritore & Love, 2020) and was well known in NPS calculations that had been instrumental in driving turnarounds in companies such as Facebook, eBay, Jet Blue, LEGO, and Apple (Reichheld & Markey, 2011; Thomas, 2014). The interview-like setting was seen to strengthen consistency between participant responses and instructor observations. Validity was further examined through analysis of responses to the two quantitative questions demonstrating perceptions of course value.

Four Question Survey

> **Question:** As an instructor, what can I do to structure the student-to-student learning, so it is more beneficial?
>
> **Question:** Do you have suggestions or ideas for changing or improving assignments in this course? Please explain.
>
> **Question:** Select all that apply to your learning in this class: [I learned a lot in this class; I learned a few things in this class that I did not already know; I did not learn anything new in this class]
>
> **Question:** I would recommend this course to other students?

The pilot project presented many challenges but was positively received. The biggest surprise was when several participants requested a second project be overlaid on another course prior to the capstone. DBR data analysis led to improvements which were implemented through alteration of the design and development of active learning components. The adjustments allowed for testing of two project variations, and better preparation as all students would have the prerequisite programming knowledge prior to taking the second course.

The next cycle incorporated a variation in the first course to address the possibility of students with no programming experience, while the second course was designed with the underlying presumption that students would have some level of programming experience. The first course increased programming guidance and reduced project variation through selection of app development and use of my first app tutorials from Android and Apple. During this cycle, group meetings became part of weekly presentations, weekly presentations continued as requirements, and there was an increase in the agile requirements. To address conflicting comments for more guidance, too many presentations, and the benefits of presentations; one intervention tested weekly presentations and the other implemented four presentations.

The design began as a simplified version of the capstone with emphasis on self-selection and individual choices (Agile Alliance, 2021; Mamoli & Mole, 2014; Md Rejab et al, 2019) and the final version retained similarities to the capstone while including significant differences. The initial version included a weekly journal to better track individual and group progress; there were several major modifications to the weekly journal. The pilot revealed several issues, the most significant being a soft prerequisite to the course; students with no prior programming experience faced greater challenges in completing the software development project. From the onset, students' perceptions of the career-like project were positive and feedback helped to streamline the design. Repeated iterations involving agile and) methods helped to meld data into a flexible and stable solution. Adjustments primarily occurred during the first three cycles, while data from the last two cycles supported design stability.

The design for the third intervention simplified the presentations to be similar to Scrum standups, while broadening group requirements to include the study and application of agile. This cycle also saw development of a topic list to allow some self-selection, while small groups were determined by the instructor. Students developed unique projects, but all projects revolved around a common topic and development style such as educational games. This cycle also included a clarification of the coding requirement, which became a picture and an explanation of a section of code. The benefits of DBR were clearly seen when every student with good attendance developed a working prototype. Table 5 shows a side-by-side comparison of the original waterfall capstone and the new agile project design which is considered stable and has been incorporated into many pre-capstone courses.

Table 5. Comparison of Agile Project to Waterfall Capstone

Agile (New Project)	Waterfall (Capstone)
Individual software development	Team-based software development
Course add-on	Entire course
Guided study group trouble-shooting	Team determined tasks
Development of related software prototypes	Development of a single software product
Weekly: team meetings and progress report	Weekly: team meetings and progress report
Three to four presentations (all class)	Weekly presentations (all teams)
Weekly project-focused discussions	
In-class agile & active learning components	
Agile terminology/practice (study group/individuals)	

The fourth cycle verified stability and flexibility of the design. This was achieved through selection of different project themes. The class project was to develop educational escape rooms, and teams were randomly selected by the instructor. Each term agreed to develop based on an educational topic, such as math, history, reading, or music. After determining the common topic, individuals selected a unique sub-topic and age level for their personal development, such as grade 2 addition. The project requirements remained the same as the third iteration: weekly report, four presentations, weekly group meetings, and in–class inactive learning assignments. The project was considered both stable and flexible, based on participant data, instructor feedback, and all active students achieving working prototypes. Below we summarize the career-like agile project by comparing it to the originating course, the SE capstone.

Conclusions and Future Work

Modern software development is challenging, complex, and requires flexibility in fast-paced environments; such is the nature of career experiences for the professional software developer. As a result, professionals are recommending that academics increase the robustness of educational practices by replacing waterfall instruction with a variety of agile tools and practices. With the intent of including an additional career-like experience in software development, we developed a project that has been used to guide many students through the complexities and challenges of software development. To assure efficacy of the project, data and literature were examined through seven DBR interventions which occurred in two classes over the course of four years.

Implementation of DBR, allowed organized collection of data, examination of literature, planning, and product testing, followed by a thorough review phase. Similar to a case study, the researchers employed DBR to collect and analyze empirical evidence from quasi-experimental, interview-like settings. The original goal of developing a completely standalone project was not achieved, possibly due to time constraints imposed by the addition of a project. The need to supplant something was noted by the participants early in the study, with a recommendation to partially flip the existing lecture materials, so that only a portion of each class period was spent in passive learning, while the remainder was supplanted with active learning that connected the project with the course materials. When the project was adjusted to include agile based active learning, all of the active students achieved the goal of working prototypes. The student recommendation to include active learning was supported in literature, where it has been seen to improve teamwork and development skills through collaborative community, building of leadership skills, and encouragement of lifelong learning (Coorey, 2016).

The first two DBR cycles led to many adjustments in design based data analysis and further literature review. For example, when additional guidance was requested, review of literature found the issue has been documented previously in complex software projects in remote teams (Johansson et al., 1999). When students recommended keeping presentations, prior studies supplied evidence that our class was considered safe enough for students to benefit from interaction with their peers (Steen-Utheim

& Foldnes, 2018). In the instance where no corroborative studies were found for the optimal number of project presentations, it was determined that both three and four class presentations worked well. When additional project guidance was requested, a portion of face-to-face time shifted to active learning through guided discussions, Scrum tools, agile practices, and group learning (Srivatanakul & Annansingh, 2022). It was unexpected that a second project would be requested, and it was a pleasant surprise to find that a single standardized template could be combined with active learning to effectively add projects to both courses.

Until a stable solution was developed, the review phase of DBR resulted in intervention modifications. Each review examined literature (for academic and professional best practice), project data, survey responses, and instructor observations. There were also a few times when an adjustment was necessary mid-way through the course; one such time occurred when we learned that several students had taken the project course without first completing the programming pre-requisite. During the second year of the study, participant feedback led to testing interventions that included a change in delivery that resulted in a partial flipped-classroom, whereby a portion of lecture time became outside reading and the remainder was replace with agile and active learning. This allowed all existing course material to remain in place but did not allow flexibility of project completion alongside lecture instruction. The adjustment to a partially flipped classroom for the third cycle, was significant. All students were able to complete prototypes, and all feedback reported the project as valuable. DBR involves testing a potential solution for flexibility and stability, so that a fourth cycle was documented. The instruction and project were similar in the third and fourth cycles, and the findings were also consistent. After reviewing data from all cycles, the project combined with active learning, evidenced a stable solution had been reached.

Using scrum tools and practices, we were able to develop an agile project similar to the capstone, in order to provide students with another career-like experience. As the study concluded, stakeholder feedback had expanded inclusion of agile projects into multiple courses. It must be noted that in each class it was found that flipping a portion of lecture and replacing it with active learning was critical to completion of working prototypes. Without some adjustments away from lecture, students were unable to develop a working prototype. In some institutions, students' perceptions of a course are important, so we note that once the project stabilized, the instructor did not feel it significantly increased their workload, but they did note a significant increase in student ratings of the course after the project was added. Like Gunyou's (2015) experience with flipped classrooms, we believe replacing a portion of lecture with active learning was the key that allowed time to develop a working prototype. It is important to note that when students first encounter project-based software development projects in courses they may feel overwhelmed; however, our graduating seniors have frequently mentioned that the inclusion of these agile projects is one of their most valuable educational experiences.

Studies have indicated that increasing the number of complex software development projects will result in broader development of professional skills. A related study found that 97% of users use mobile or desktop devices (Statcounter Global Stats, 2019), so that the researchers recommend complex software development experiences in both desktop and app development. Recommendations for further study include expansion to online courses and implementation in larger classes. The researchers have now tested the proposed solution in class sizes up to 25 students, online courses, independent studies, and creative works project with similar results. This experience, and the development of such a flexible, yet stable solution, has resulted in a change to our instruction which now commonly includes both agile and active learning in many courses.

References

Adkins, J., & Tu, C. (2021). Online teaching effectiveness: A case study of online 4-week classes in a graduate information systems program. *Information Systems Education Journal, 19*(3), 31-37. https://files.eric.ed.gov/fulltext/EJ1301232.pdf

Agile Alliance. (n.d. -a..). *Agile glossary*. https://www.agilealliance.org/agile101/agile-glossary/

Agile Alliance. (n.d. -b). *What is Agile*?. https://www.agilealliance.org/agile101/

Alperowitz, L., Dzvonyar, D., & Bruegge, B. (2016). Metrics in Agile project courses. *Proceedings of the 38th International Conference on Software Engineering Companion*. https://doi.org/10.1145/2889160.2889183

Armstrong, M., Dopp, C., & Welsh, J. (2018). Design-based research. In R. Kimmons (Ed.), *The students' guide to learning design and research*. EdTech Books. https://edtechbooks.org/studentguide/design-based_research

Bakke, C. (2013). *Perceptions of professional and educational skills learning opportunities made available through K-12 robotics programming* [Ph.D. thesis, Capella University]. https://www.learntechlib.org/p/120073/

Barab, S. A., Thomas, M. K., Dodge, T., Squire, K., & Newell, M. (2004). Critical design ethnography: Designing for change. *Anthropology Education Quarterly, 35(*2), 254–268. https://doi.org/10.1525/aeq.2004.35.2.254

Bishop, J. L., & Verleger, M. A. (2013). *The flipped classroom: A survey of the research*. Paper presented at the 120th American Society for Engineering Education Annual Conference and Exposition, 30, 1-18. https://doi.org/10.18260/1-2--22585

Boddy, C. R. (2016). Sample size for qualitative research. *Qualitative Market Research, 19*(4), 426-432. https://doi.org/10.1108/QMR-06-2016-0053

BU Center for Teaching and Learning (n.d.). *Experiential Learning*. Boston University. https://www.bu.edu/ctl/guides/experiential-learning/

Brown, G. T. L., Peterson, E. R., & Yao, E. S. (2016). Student conceptions of feedback: Impact on self-regulation, self-efficacy, and academic achievement. *British Journal of Educational Psychology, 86*(4), 606-629. https://doi.org/10.1111/bjep.12126

Bureau of Labor Statistics. (2018, April 13). *Occupational outlook handbook*. https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm

Carless, D., & Boud, D. (2018). The development of student feedback literacy: Enabling uptake of feedback. *Assessment & Evaluation in Higher Education, 43*(8), 1315–1325. https://doi.org/10.1080/02602938.2018.1463354

California Project Management Office. (2017, September 22). 3*.8.1 Agile Tools and Techniques*. California Department of Technology, State of California. https://projectresources.cdt.ca.gov/agile/agile-tools-and-techniques/

Chan, P. E., Graham-Day, K. J., Ressa, V. A., Peters, M. T., & Konrad. M. (2014). Beyond involvement: Promoting student ownership of learning in classrooms. *Intervention in School and Clinic, 50*(2), 105-113. https://doi.org/10.1177/1053451214536039

Collins, A., Joseph, D., & Bielaczyc, K. (2004). Design research: Theoretical and methodological issues. *Journal of the Learning Sciences, 13*(1), 15-42. https://doi.org/10.1207/s15327809jls1301_2

Coorey, J. (2016). Active learning methods and technology: Strategies for design education. *International Journal of Art & Design Education, 35*(3), 337–347. https://doi.org/10.1111/jade.12112

Corritore, C. L., & Love, B. (2020). Redesigning an introductory programming course to facilitate effective student learning: A case study. *Journal of Information Technology Education: Innovations in Practice, 19*, 91-135. https://doi.org/10.28945/4618

Crouch, M, & McKenzie, H. (2006) The logic of small samples in interview-based qualitative research. *Social Science Information, 45(4)*, 483-499. https://doi.org/10.1177/0539018406069584

Dewey, J. (1966). *Democracy and education: an introduction to the philosophy of education*. The Free Press.

Design-Based Research Collective. (2003). Design-based research: An emerging paradigm for educational inquiry. *Educational Researcher, 32*(1), 5-8. https://doi.org/10.3102/0013189X032001005

Digital.ai. (2020). *15th annual state of agile report.* https://stateofagile.com/#ufh-i-615706098-14th-annual-state-of-agile-report/7027494

Dinis-Carvalho, J., Ferreira, A. R., Barbosa, C. S., Lopes, C., Macedo, H., & Tereso, P. (2018). Effectiveness of SCRUM in Project-Based Learning: Students View. *Proceedings of the Regional Helix'18 Conference – International Conference on Innovation, Engineering and Entrepreneurship*, Guimarães, Portugal on June 27–29, 2018 https://doi.org/10.1007/978-3-319-91334-6_154

Duffany, J. (2015). *Engineering education facing the grand challenges, What are we doing?* Paper presented at the 13th LACCEI Annual International Conference. https://doi.org/10.18687/LACCEI2015.1.1.246

Eddy, S. L., & Hogan, K. A. (2014). Getting under the hood: How and for whom does increasing course structure work? *CBE—Life Sciences Education, 13*(3), 453–468. https://doi.org/10.1187/cbe.14-03-0050

European Commission. (2020, October 21). *Communication to the Commission. Open Source Software Strategy 2020-2023: Think Open.* https://ec.europa.eu/info/sites/default/files/en_ec_open_source_strategy_2020-2023.pdf

Figueiredo, J., & García-Peñalvo, F. J. (2019, October). Teaching and learning strategies of programming for university courses. *Proceedings of the 7th International Conference on Technological Ecosystems for Enhancing Multiculturality* (pp. 1020–1027). León, Spain: ACM. https://doi.org/10.1145/3362789.3362926

Freeman, S., Eddy, S., McDonough, M., Smith, M., Okoroafor, N., Jordt, H., & Wenderoth, M. (2014). Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the National Academy of Sciences*, 111(23), 8410-8415. National Academy of Sciences. https://doi.org/10.1073/pnas.1319030111

Fronza, I., Corral, L., & Pahl, C. (2020). End-user software development: Effectiveness of a software engineering-centric instructional strategy. *Journal of Information Technology Education: Research, 19,* 367-393. https://doi.org/10.28945/4580

Gama, K., Alencar Goncalves, B., & Alessio, P. (2018). Hackathons in the formal learning process. *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (pp. 248–253). https://doi.org/10.1145/3197091.3197138

Giorgdze, M., & Dgebuadze, M. (2017). Interactive teaching methods: Challenges and perspectives. *IJAEDU-International E-Journal of Advances in Education, 3*(9), 544-548. https://doi.org/10.18768/ijaedu.370419

Gunyou, J. (2015) I flipped my classroom: One teacher's quest to remain relevant. *Journal of Public Affairs Education, 21*(1), 13-24. https://doi.org/10.1080/15236803.2015.12001813

Graus, M., van de Broek, A., Hennissen, P., & Schils, T. (2022). Disentangling aspects of teacher identity learning from reflective blogs: The development of a category system. *Teaching and Teacher Education, 111*, 103624. https://doi.org/10.1016/j.tate.2021.103624

Haagen-Schützenhöfer, C., & Hopf, M. (2020). Design-based research as a model for systematic curriculum development: The example of a curriculum for introductory optics. *Physical Review Physics Education Research, 16*(2). https://doi.org/10.1103/physrevphyseducres.16.020152

Hawks, S.J. (2014). The flipped classroom: Now or never? *Education News.* https://www.semanticscholar.org/paper/The-Flipped-Classroom-%3A-Now-or-Never-EDUCATION-NEWS-Hawks/1ccc20a540c78cf9f89c440b29958f550b7fdf4b

Herawati, S., Negara, Y. D. P., Febriansyah, H. F., & Fatah, D. A. (2021). Application of the waterfall method on a web-based job training management information system at Trunojoyo University Madura. *E3S Web of Conferences, 328*, Article 04026. https://doi.org/10.1051/e3sconf/202132804026

Jazayeri, M. (2015) Combining mastery learning with project-based learning in a first programming course: An experience report. *Proceedings of the IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE),* pp. 315–318. https://doi.org/10.1109/ICSE.2015.163

Johansson, C., Dittrich, Y., & Juustila, A. (1999). Software engineering across boundaries: Student project in distributed collaboration. *IEEE Transactions on Professional Communication, 42*(4). 286-296. https://doi.org/10.1109/47.807967 .

Kanbanize. (n.d.). *What is Kanban? Explained for beginners*. https://kanbanize.com/kanban-resources/getting-started/what-is-kanban

Klinger, M, Swaby, M., & Walbridge, E. (2022). *Flipped classrooms*. Derek Bok Center for Teaching and Learning. Harvard University. https://bokcenter.harvard.edu/flipped-classrooms

Lemos, A. R., Sandars, J. E., Alves, P., & Costa, M. J. (2014). The evaluation of student-centeredness of teaching and learning: A new mixed-methods approach. *International Journal of Medical Education, 5,* 157–164. https://doi.org/10.5116/ijme.53cb.8f87

LucidChart. (2017). *The pros and cons of waterfall methodology*. https://www.lucidchart.com/blog/pros-and-cons-of-waterfall-methodology

Magana, A., Seah, Y., & Thomas, P. (2018). Fostering cooperative learning with Scrum in a semi-capstone systems analysis and design course. *Journal of Information Systems Education, 29*(2), 75–92. https://jise.org/Volume29/n2/JISEv29n2p75.pdf

Mahnic, V. (2012). A capstone course on agile software development using Scrum. *IEEE Transactions on Education, 55*(1), 99–106. https://doi.org/10.1109/te.2011.2142311

Mamoli, S., & Mole, D. (2014). Self-selecting teams part 1 - Why you should try self-selection. *Methods & Tools*. http://www.methodsandtools.com/archive/selfselectingteams.php

Mamoli, S., & Mole, D. (2016). Creating great teams. *How Self-Selection Lets People Excel*. Agile Alliance. https://www.agilealliance.org/resources/experience-reports/creating-how-self-selection-lets-people-excel/

Mandonca, C. (2016, March 3). *About self-organizing teams*. Scrum.org. https://www.scrum.org/resources/blog/about-self-organizing-teams

Marques, M., Ochoa, S. F., Bastarrica, M. C., & Gutierrez, F. J. (2018) Enhancing the student learning experience in software engineering project courses. *IEEE Transactions on Education, 61*(1), 63–73. https://doi.org/10.1109/TE.2017.2742989

Maxim, B., Acharya, S., Brunvand, S., & Kessentini, M. (2017). WIP: Introducing Active Learning in a Software Engineering Course. *American Society for Engineering Education,* ASEE. Paper #17715. https://peer.asee.org/wip-introducing-active-learning-in-a-software-engineering-course.pdf

McConnell, S. (2004). *Code complete* (2nd ed.). Microsoft Press, USA.

Mckenney, S., & Reeves, T. (2013). Systematic review of design-based research progress Is a little knowledge a dangerous thing? *Educational Researcher, 42,* 97-100. https://doi.org/10.3102/0013189X12463781

Md Rejab, M., Noble, J., & Marshall, S. (2019). Agile self-selecting teams foster expertise coordination. *Interdisciplinary Journal of Information, Knowledge, and Management, 14*, 99-117. https://doi.org/10.28945/4280

Mendo-Lázaro, S., León-del-Barco, B., Felipe-Castaño, E., Polo-del-Río, M.-I., & Iglesias-Gallego, D. (2018). Cooperative team learning and the development of social skills in higher education: The variables involved. *Frontiers in Psychology, 9*. https://doi.org/10.3389/fpsyg.2018.01536

National Science Foundation. (May 12, 2014). *Enough with the lecturing* [News Release 14-064]. https://www.nsf.gov/news/news_summ.jsp?cntn_id=131403

National Institute for Excellence in Teaching. (2021) *Learning acceleration resources: Fostering Student Ownership through Thinking and Problem-Solving*. https://www.niet.org/assets/Resources/student-ownership-thinking-problem-solving.pdf

9th Global Project Management Survey. (2017). *Success Rates Rise: Transforming the high cost of low performance.* PMI's Pulse of the Profession. https://www.pmi.org/-/media/pmi/documents/public/pdf/learning/thought-leadership/pulse/pulse-of-the-profession-2017.pdf

Parsons, D., & MacCallum, K. (Eds.). (2019). *Agile and lean concepts for teaching and learning: Bringing methodologies from industry to the classroom*. Springer. https://doi.org/10.1007/978-981-13-2751-3

Pattanaphanchai, J. (2019). An investigation of students' learning achievement and perception using flipped classroom in an introductory programming course: A case study of Thailand higher education. *Journal of University Teaching and Learning Practice, 16*(5), Article 4. https://doi.org/10.53761/1.16.5.4

Peters, L. J., & Tripp, L. L. (1977). Comparing software design methodologies. *Datamation, 23*(11), 89-94.

Radigan, D. (2019). *What is Kanban?* Atlassian. https://www.atlassian.com/agile/kanban

Reichheld, F., & Markey, R. (2011). *The ultimate question 2.0. How net promoter companies thrive in a customer-driven world*. Bain & Company. https://www.bain.com/insights/the-ultimate-question-2/

Rico, D.F., & Sayani, H.H. (2009). Use of agile methods in software engineering education. *Proceedings of the 2009 Agile Conference,* pp. 174-179. https://doi.org/10.1109/AGILE.2009.13

Rittel, H. W. J., & Webber, M. (1973). Dilemmas in a general theory of planning. *Policy Sciences, 4,*155-169. https://doi.org/10.1007/BF01405730

ReQtest. (2019, November 20). *What is waterfall methodology? Long live waterfall!* https://reqtest.com/agile-blog/what-is-waterfall-methodology/

Royce, W. W. (1970). Managing the development of large software systems. *Proceedings of IEEE WESCON,* 26, 328 - 388. https://www.semanticscholar.org/paper/Managing-the-development-of-large-software-systems%3A-Royce/4afe47371b891778c6cc6fa401bfc1673ea0d63f

Saadé, R. G., & Shah, S. (2016). Exploring an agile learning activity to teach agile project management. *Proceedings of Informing Science & IT Education Conference (InSITE) 2016,* 95-101. https://doi.org/10.28945/3454

Sahin, Y. G., & Celikkan, U. (2020). Information technology asymmetry and gaps between higher education institutions and industry. *Journal of Information Technology Education: Research, 19*, 339-365. https://doi.org/10.28945/4553

Sandelowski M. (1996). One is the liveliest number: The case orientation of qualitative research. *Research in Nursing & Health, 19*(6), 525–529. https://doi.org/10.1002/(SICI)1098-240X(199612)19:6%3C525::AID-NUR8%3E3.0.CO;2-Q

Schilling, S., & Klamma, R. (2010). The difficult bridge between university and industry: A case study in computer science teaching. *Assessment & Evaluation in Higher Education 35*(4), 367-380. https://doi.org/10.1080/02602930902795893

Scrum.org. (n.d.). *What is scrum*? https://www.scrum.org/resources/what-is-scrum

Scrum Guides. (2017). Scrumguides.org. https://scrumguides.org/scrum-guide.html

Sheffield, M. (2019, June 25). *10 Key Factors to Ensure Software Project Success*. Seamgen. https://www.seamgen.com/blog/key-factors-software-project-success/

Srivatanakul, T., & Annansingh, F. (2022). Incorporating active learning activities to the design and development of an undergraduate software and web security course. *Journal of Computers in Education 9*, 25–50. https://doi.org/10.1007/s40692-021-00194-9

Standish Group. (2015) *CHAOS Report 2015*. https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf

Standish Group. (2020). *Chaos 2020: Beyond Infinity*. https://www.standishgroup.com/news/49

Statcounter Global Stats. (2019). Desktop vs mobile vs tablet market share worldwide. https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet

Steen-Utheim, A. T., & Foldnes, N. (2018). A qualitative investigation of student engagement in a flipped classroom. *Teaching in Higher Education, 23*(3), 307-324. https://doi.org/10.1080/13562517.2017.1379481

Talbert, R., & Mor-Avi, A. (2019). A space for learning: An analysis of research on active learning spaces. *Heliyon, 5*(12), e02967. https://doi.org/10.1016/j.heliyon.2019.e02967

Thomas, J. W. (2014). *The Ultimate Question® and the Net Promoter® score*. Decision Analyst. https://www.decisionanalyst.com/whitepapers/ultimatequestion/

Viechnicki, P., & Keikar, M. (2017, May 5). *Agile by the numbers: A data analysis of Agile development in the US federal government*. Deloitte. https://www2.deloitte.com/us/en/insights/industry/public-sector/agile-in-government-by-the-numbers.html

Weekly tip: Ownership of learning. (2019, Jan 25). *Faculty Insider*. Washington State University. https://li.wsu.edu/2019/01/25/ownership-of-learning/

Wong, S. W. L., Miao, H., Cheng, R. W., & Yip, M. C. W. (2017). Graphic novel comprehension among learners with differential cognitive styles and reading abilities. *Reading & Writing Quarterly, 33*(5), 412-427. https://doi.org/10.1080/10573569.2016.1216343

Zhang, X., & Dorn, B. (2012). Accelerating software development through agile practices-A case study of a small-scale, time-intensive web development project at a college-level IT competition. *Journal of Information Technology Education: Innovations in Practice, 11*, 25-37. https://doi.org/10.28945/1545 .

APPENDICES

APPENDIX A: STUDENT FEEDBACK

Student Survey

- Question: As an instructor, what can I do to structure the student-to-student learning, so it is more beneficial?
- Question: Do you have suggestions or ideas for changing or improving assignments in this course? Please explain.
- Question: Select all that apply to your learning in this class: [I learned a lot in this class; I learned a few things in this class that I did not already know; I did not learn anything new in this class]
- Question: I would recommend this course to other students?

APPENDIX B: PROFESSIONAL TOOLS

Professional Tools

Project need	**Software examples**
Kanban team task board	Trello, JIRA, Monday.com
Shared code repository in the cloud	Github, BitBucket
Communication	Slack, Discord
Weekly meetings	Zoom, Messenger, Hangouts
Facilitator meetings & report	Report template, zoom, online team documents

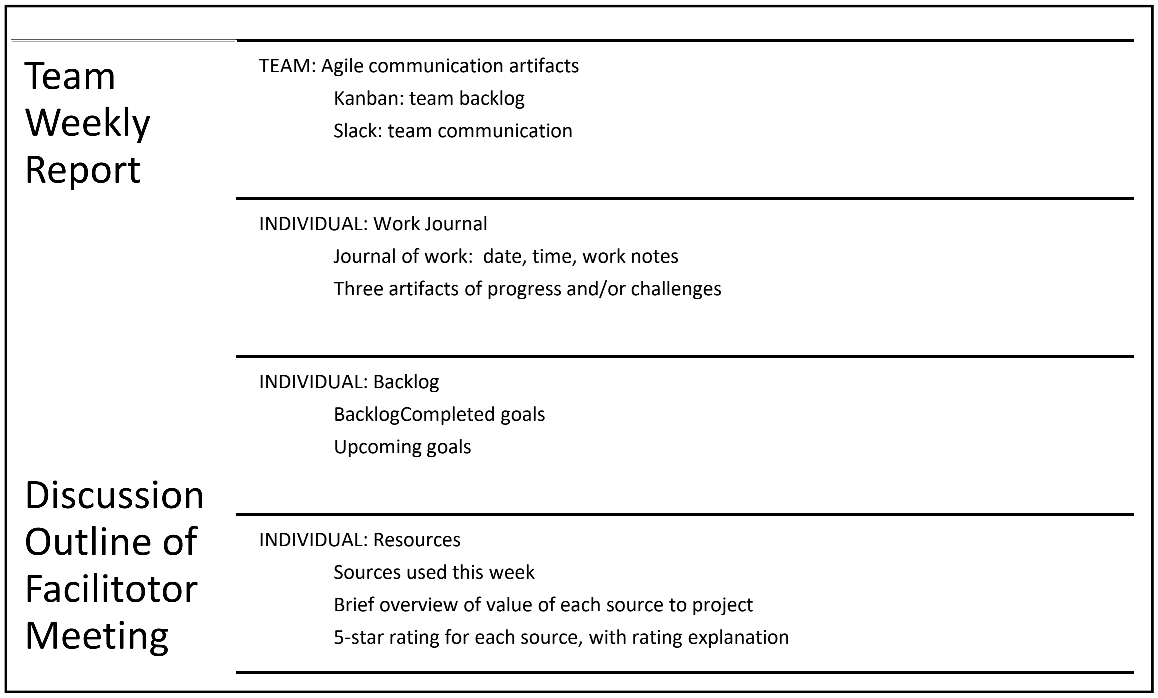
APPENDIX C: PROJECT OVERVIEW

Project Overview

	Pilot Design	DBR Design
Teams	Instructor selected	Team selection based on individual project topic
Project topic	Student selected, no list provided	A course focuses on one of the following: app development, desktop development, website development. Topic preferences are selected by each student from a provided list. Study teams of 3 – 4 are determined based on topic selection. Study groups determine individual projects that cannot overlap one another. Example: Educational Escape Rooms. Group selection: History. Individual selections: Music, Religion, Rome, Agriculture.

Agile terminology and practices	Active learning, in class	Study groups meetings are guided through all-class discussions that guide software development toward prototype through a pre-determined series of goals.
Weekly report	Group meeting summary Individual project progress, this week's goals, project sample, problems or fixes, goals for next week, sources	Group meeting summary Individual project progress: three questions, artifact, challenge and solution, top source.
Weekly study group	The time and topics to be covered are determined by the group	Agile research and implementation are embedded in weekly discussions.
Project presentations	Two, midterm and last week of class. Working prototype required for the last presentation	Four checkpoints. Requirements for each checkpoint are divided into weekly study group discussions.

Example of Weekly Report Requirements



Team Weekly Report

TEAM: Agile communication artifacts
- Kanban: team backlog
- Slack: team communication

INDIVIDUAL: Work Journal
- Journal of work: date, time, work notes
- Three artifacts of progress and/or challenges

INDIVIDUAL: Backlog
- BacklogCompleted goals
- Upcoming goals

Discussion Outline of Facilitotor Meeting

INDIVIDUAL: Resources
- Sources used this week
- Brief overview of value of each source to project
- 5-star rating for each source, with rating explanation

Example of Presentation Deliverables

Presentations	Software Development Goals
1: Preparation, Design	Instructions (1), UI designs (2), points(3), leaderboard (4)
2: Three Screens	Start screen (1), instructions (2), win screen(3), music (4)
3: Movement & Interactions	User movement and interaction with an object (1-2) Solving 2 puzzles activates win screen (3-4)
4: Final Prototype	Working score (1), Solving 3 puzzles (2) -> Win Screen and Leaderboard (3). Check: students play other student's games

Appendix D: Software Topic Examples

Examples of prototyped apps:

- Escape Rooms
- IoT Home Security Systems
- Smart Home devices
- "Photoshop" filters
- Bus routing system
- Welcome to our Campus
- Social Connections
- Grocery Shopping
- Speaking – disability

Active Learning: Representative Discussion

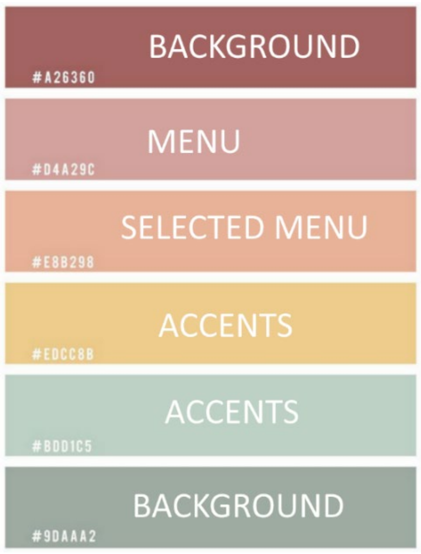
WEEKLY TEAM MEETING REQUIREMENTS

In your weekly small group meeting, discuss the following. Add both questions and answers to your group meeting notes. Also include the date, time, and group participants.

Group meeting: Tuesday, 4:00 pm. We met using zoom.

WEEKLY MEETING GOAL #1: Determine colors and fonts

1. *Determine a color palette that will be used by all group members. Label background color(s), state main font color(s), accent color(s), and colors that will be used for navigation. Each color is to be labeled with its hexadecimal number.*



We will use white for all words / fonts.

2. *Determine the font family that will be used by all group members.*

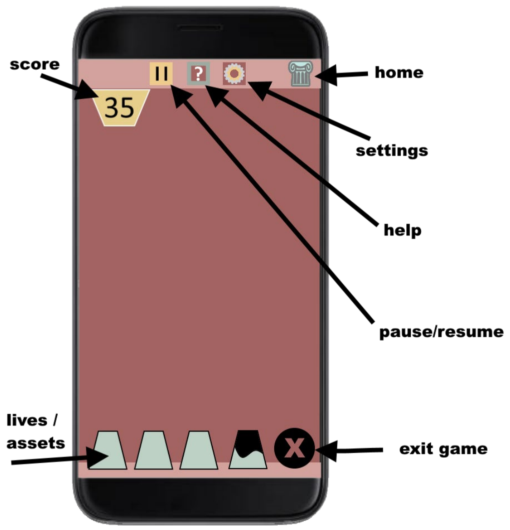
{ Helvetica, Calibri, Arial, Verdana }

WEEKLY MEETING GOAL #2: Develop common UI design

3. *Design and explain how the team logo will be common to all projects.*

Our group chose the history of the ancient Roman Em-pire. The Logo is a column. The logo will be dark green (#9DAAA2) and light green (#BDD1C5). It will be the home button in the upper right corner of all pro-jects.

4. *Determine the basic layout for the game play screen, this layout will be used for all team members. Determine at least 5 common features such as score, assets, pause/resume, save, home, exit, help, settings…. Hand sketch (or use a drawing program) the layout everyone will incorporate.*



WEEKLY MEETING GOAL #3: Each person's requirements

Determine an overview for your game. All team games must be on the common, selected theme. Explain gameplay and score. How is your game unique from other team members?

Jackson: My game will be based on Sparta. Players will compete in events such as running, javelin throwing, obstacle courses, and chariot races. Players can play on the computer or with friends online. Points are earned for answering trivia questions, based on response speed. Incorrect answers cost points.

Lillinette: Temple is a game where players build temples and homes. Everyone starts with basic tools and supplies. To get more supplies or decorate the players must answer questions about city-states and ancient Roman towns. If they want to build more, they have to also buy land. The game does not have a time limit and there are no aggressors, but if the questions are answered incorrectly, they lose supplies.

Eli: Players search for ancient coins in the catacombs. They might encounter creatures, centaurs, or other aggressors. The catacombs are a maze and they cannot see the whole thing, so they might get lost. They must complete the game within the time limit and without losing to a creature. When they encounter something dangerous, they have to answer questions about ancient Rome. Wrong answers lose time, correct answers add time.

MEETING GOAL #4: Keep a summary of the team meeting (submit for discussion)

List the time/day of your study-team meeting
List notes for everyone who attended.
Summarize teamwork, communication, and challenges addressed during the meeting.

{this show the information in goal 1, 2, 3 and may contain other meeting notes}

AUTHORS

Dr. Christine Bakke is a Software Engineering and Information Technology instructor at the University of Minnesota, Crookston. Her Information Technology PhD degree specialized in educational robotics; while her undergraduate was in Computer Science, programming, and mathematics. Her professional experience includes 18 years as an IT professional with specializations in networks, cybersecurity, database, and programming. Her research focuses on active learning techniques that combine academic and professional best practices into agile active learning experiences. Recent projects include Scrum-like development of chatbots, speech assistant software, and custom IoT devices with software.

Rena Sakai is an honors student at the University of Minnesota, Crookston. She is working to complete her bachelor's degree in Software Engineering, with minors in Information Technology Management and Cybersecurity. Her research interests center around new ways to learn, innovative and immersive methods, and project-based instruction. Her previous creative works studies include Flutter programming, Student Ownership of Learning, and instructional design founded in active learning strategies.