International Journal of Engineering Education Vol. 35, No. 4, pp. 999–1007, 2019
Printed in Great Britain

0949-149X/91 $3.00+0.00

Evolution of a Graduate Software Engineering Capstone Course—A Course Review*

PHILLIP A. LAPLANTE, JOANNA F. DEFRANCO and EVERTON GUIMARAES
Pennsylvania State University, 30 East Swedesford Road, Malvern, PA 19355, USA.
E-mail: pal11@psu.edu, jfd104@psu.edu, ezt157@psu.edu

In education projects, students sometimes have difficulty conveying their knowledge when asked to solve real problems especially when a software product is generated as an outcome. Faculty accumulated nearly 12 years of experience running a graduate software engineering capstone course perceived the need to better assist students on the learning process. Course adaptations are required, particularly considering the heterogeneity of student's background and work experience, as well as the use of current technologies and tools. The paper outlines an evolved graduate software engineering capstone course for part-time graduate professional students. The capstone course is intended to create a meaningful student experience while providing a productive environment to apply knowledge learned from the program. Moreover, the new proposal emphasizes agile methodologies and code as the primary artifact. Our main contribution is to present the evolution of this capstone course, from inception to coding, testing and deployment. The course revision includes new artifacts, such as a work breakdown structure and burndown chart, as means to improve the course learning outcomes based on lessons learned and student experience. For evaluation purposes, we selected groups of most recent capstone course sections totaling 175 students. Two learning objectives of the program were evaluated: teamwork and critical thinking. Critical thinking was assessed via the project plan artifact. Teamwork was assessed through discussion forums, team test plans and reports. The results were positive; however, they indicate that not all elements of project planning are present in the studio course.

Keywords: software engineering; education; capstone project; agile; graduate program

1. Introduction

One of the main goals of software engineering graduate programs is to prepare students to be ready for the job market, more specifically, industrial software development. Young professional software engineers, however, are not always prepared to solve real problems. Many universities attempted to address this concern through capstone courses in (under)graduate software engineering programs. A capstone course is quite common, particularly when the program has a professional (non-research) orientation. During the capstone course, students are expected to build a software application from conception through delivery utilizing concepts and techniques learned during the program.

Graduate software engineering capstone courses have been occasionally reported, however, many factors that affect learning objectives and student experience still need to be addressed. While there are several treatments of undergraduate studio discussions in the literature [1–5, 8], only a few report such experience for part-time graduate students. To the best of our knowledge, our work is the first to propose course adaptations for online delivery. The capstone course addresses these shortcomings and has evolved from a resident delivered, waterfall-based, instructor-led team project to a resident and online, agile, student-centered team project.

The present work reviews and evaluates a studio course for part-time, professional students completing a professional master's in software engineering (MSE) degree. This course was first reported in [11] and cited more than 20 times. The course has been delivered both in residence and online. The main contribution of this work is to review the lessons learned and revisions of this software studio experience. In addition to the retrospective discussion of the software studio design and delivery over the past 12 years, we will describe the evaluation and results of assessing course artifacts as part of a program assessment for the MSE degree. Although a meaningful studio experience has been provided to the MSE students, the assessment shows a few gaps that will prompt another course revision. These changes will also be discussed.

2. Background

Researchers explored different experiences for capstone courses in both undergraduate and graduate levels [4–7, 9]. The state of art shows that capstone courses in software engineering (SE) education usually run from 7 to 14 weeks and employ either waterfall of agile as development process. In some cases, industry customers are also involved during the development process as means to allow students a more immersive experience. Although most capstone courses adopt agile methodologies and prac-

* Accepted 11 March 2019.

tices, there are still many factors affecting the learning objectives in a capstone project, as well as problems faced by students during the development process. Next, we describe most recent studies and briefly discuss each of them.

2.1 Case studies on SE capstone courses

A study developed in [10] introduces a case study, developed at the University of Central Missouri, using agile approaches in student's capstone class. The course was designed to teach agile software development using SCRUM, and therefore, students are required to work in teams while on some occasions individual projects could be allowed. The course organization has the following roles: SCRUM Master (Instructor), Agile Mentor (Graduate Assistants), SCRUM Team (size of two to four students) and Product Owner (Domain Expert who proposed the project). The core lessons learned, and recommendations can be summarized as: estimation of effort and task planning; larger projects are attached to students; Git repositories are strongly recommended (source version control), and the product owner is critical for the success of the project. Ultimately, one of the projects developed using the proposed methodology eventually ended up being adopted by the university and released to more than ten thousand students.

In turn, Bastarrica et al. (2017) proposed a study for a capstone course at the Universidad de Chile. The course initially followed a waterfall process. Adopting a waterfall process in a capstone course makes the students not only feel disengaged, but the outcome in terms of documentation was not strictly necessary for all clients. After 5 years, the course curriculum had been reviewed, and the course structure change to adopt agile methodologies with a focus on three relevant practices: timeboxing, client on site and incremental development. The study aimed at measuring what students have learned during the capstone course using a survey of student perceptions of learning outcome. The study revealed that student's perceived soft skills as determinant factors for project success. Further, students' feedback of the analyzed aspects had a significant change in their initial and final perception, an intended learning outcome of the course.

Balahan and Sturn [5] proposed a software engineering course aimed to create lab conditions, so students are faced with challenging planning software engineering development and project management. The research outlines the ideals, principles, and goals when proposing a new lab course considering best practices in software engineering. Moreover, the software engineering lab course simulated development tasks and was comprised of the following features/practices: agile development, product validation, structured teamwork, maintenance tasks, and software engineering systematic development. Authors evaluated several aspects concerning the proposed course, especially measuring the student's experience regarding the software development process, development in teams, requirements management, software design, software implementation, and overall software quality. Finally, the results suggest a need to balance the team instruction and student grading better.

2.2 The challenge of team project courses

A study proposed by Ahtee and Poranen [6] evaluates the final report of software projects in two different universities in Finland. The study reported four significant risks and challenges students faced while developing a software project: tools, and skills to use available technology (61% of projects); technological problems (53%); project management (61%); and working/studying too many other courses during the project. Based on the reported results, authors suggest faculty and students should pay attention to support and training when using development tools and planning the project. Furthermore, authors conclude that a causality analysis between different risk items should be performed, as some categories are expansive, and should be divided into smaller categories for better understanding and mitigating project risk.

Vanhanen et al [9] performed a survey in a capstone project to understand the problems in software development. As a result, the authors reported that the main problems in software development were related to testing, task management, and technology skills. As the capstone project was in collaboration with industrial customers; however, problems related to task management and customer expectations had negatively affected customer satisfaction. The problems directly affecting customer satisfaction are related to technology skills, version control system, task management, team communication, and customer communication.

Finally, Koolmanojwong and Boehm [7] analyzed a graduate-level software engineering course at the University of Southern California. The course teaches the best software engineering practices as means to allow students to apply the knowledge learned from previous courses in developing real-client projects. As a result, the study shows how risks in graduate software project had evolved in nearly 20 years. For instance, new factors are considered in the top 10 risks in software engineering graduate projects, namely: COTS and independent involving systems, customer-developer-user team cohesion, and process quality assurance. The results showed that team project courses have

difficult tackling problems in that students need to do just-in-time learning. For example, students' struggles are mostly concerned about how to select and organize team members, how to interact with project clients to understand their needs and environment, how to negotiate feasible requirements, how to develop project plans, and how to define the system's architecture.

3. Program description

The MSE degree offered at Penn State University (PSU) is a 36-credit (12-course) program, which aims at preparing computer professionals to develop, maintain, manage, test, and understand software products and services more effectively. The program is delivered both online and in residence. The residency program is only offered at PSU's Great Valley Graduate Professional School located in Southeastern Pennsylvania. Although some students are full time, most are part-time software professionals. Many have 20 or more years' experience working in the avionics, medical and financial industries. Not all have recent hands-on software development experience; however, they all work in some aspect of the software development process such as project managers, testers, documentation specialists, technical support, database administrators, and so on. The online courses are conducted by the same faculty who teach face-to-face. The online program differs only in that the student's progress through the program in a cohort.

4. Course design

The primary intent of the capstone course in the Master of Software Engineering (MSE) program is to provide a capstone experience that synthesizes theory learned in predecessor courses into an industrial strength software project. Therefore, students are expected to design and implement a viable software product within 14 weeks. The capstone course is typically the last course taken by the students in the program, and the course objectives can be summarized as: (i) develop a significant software system from requirements through code deployment; (ii) learn new languages, technologies, environments, methodologies, and application domains; and (iii) learn to function effectively as a team member on a complex software project.

Creating an effective studio course for professional students with heterogenous experiences is challenging, especially in online delivery. The initial course approach using a waterfall methodology was revised and described in [11]. This revision emphasizes agile methodologies and code as the primary artifact. Nevertheless, there were four additional goals on the horizon of the next revision:

1. Allow students to discuss and conceive the software project, instead of providing them with a list of predefine projects. This strategy helps students meet their specific needs and take ownership of the project.
2. Manage student personalities, expectations, and attitudes through frequent instructor interaction.
3. Emphasize the use of open-source tools and frameworks, off-the-shelf code, and support for collaboration and communication.
4. Help students manage the time spent on the project, as well as track the project activities and individual contributions

Another objective of the project is for the student to extend their skill set as a software engineer. In other words, the student obtains experience in some technology or tool in which they are unfamiliar whether it is a new language, framework, web services development, or database. The teams collaborate on a project in which the instructor approves. Moreover, they should ideally meet with the instructor every other week where requirements and goals are refined as well as progress and any current and (un)resolved obstacles are discussed.

All these goals are adequately addressed in the current course design along with using agile principles and methodologies, such as test-driven development (TDD) [12] and SCRUM. Furthermore, all project documentation required from the students are living documents in that the student teams resubmit as changes occur in the design. Fig. 1 shows the course schedule distributed within 14 weeks, which are organized into 7 sprints, ultimately culminating in a fully documented viable product. The first sprint comprises the first two weeks, when students will organize themselves in groups of up to 5 members. Next, students need to define the project scope and outline a software requirement specification along with the test documents (following the TDD approach). Once requirements are documented and validated, each group should come up with an estimation of software in terms of time and complexity. At the end of each Sprint, students perform a team retrospective to revisit and discuss all documentation, and if needed, apply changes based on the instructor feedback.

Sprint 2 runs on weeks 3 and 4, when students need to work on the domain and class modeling. Next, teams need to evaluate and update the test document and reports, especially in cases of any changes on the requirements specification or even if the architecture design decisions require new tests to be created. Team retrospective will be held at the

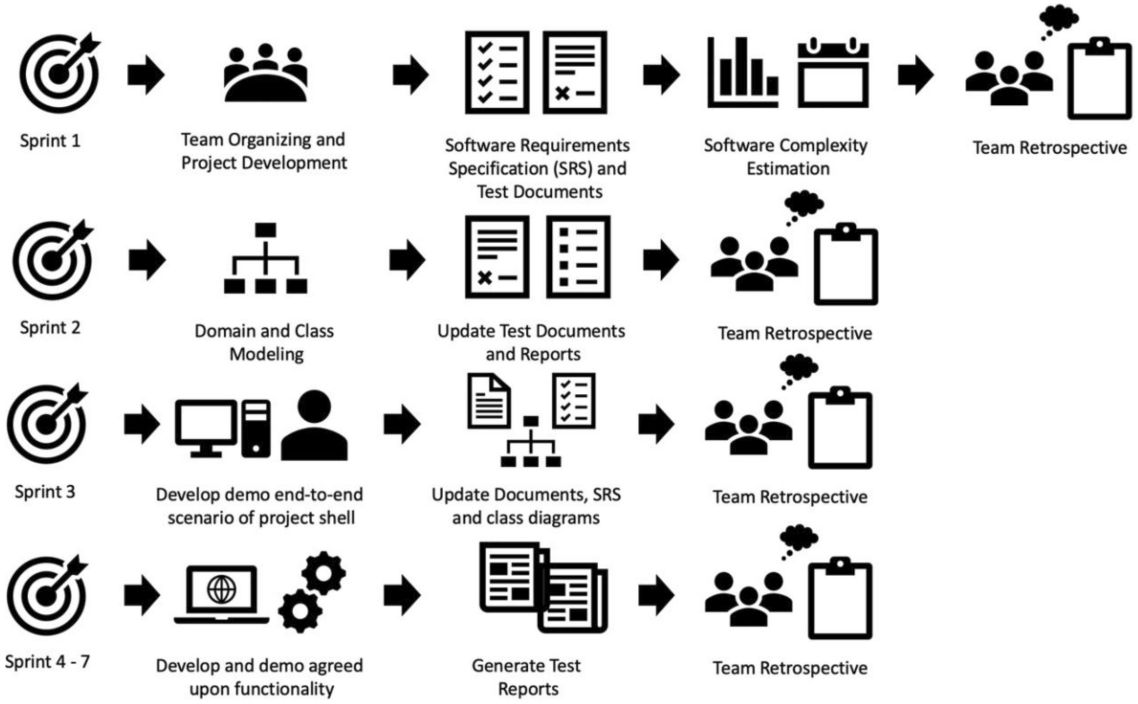

Fig. 1. Current Course Structure.

end of week 4. Sprint 3 runs from weeks 5 to 6, when teams are expected to develop and demo an end-to-end scenario of project shell. Based on the initial software prototype develop by each team, the software documentation should be updated. The team retrospective will serve as basis for teams to better plan the project and agree upon the functionalities the team will commit to deliver in the upcoming sprints. Sprints 4 to 7 will run for the remaining of the course, each one been 2 weeks in length. Groups are expected to update the software product using TDD practices along with SCRUM. For instance, when using TDD teams are expected to generate automated testing using a unit framework, and those tests can ultimately be used for regression testing as occur to the system [13]. Incomplete use cases return to the backlog, and teams plan the next Sprint based on the Team Retrospective.

5. Assessment

Based on the learning objectives of the MSE program *teamwork* and *critical thinking* were assessed. We collected data from seven consecutive sections of the capstone course, which has been taught by five different faculty members. Moreover, each section has, in average five teams, and therefore we evaluated 35 samples of each artifact (project plan, test plans and test reports). The assessment and description of each learning objective is discussed in the following sections

5.1 Teamwork

The *teamwork* learning objective is defined as "graduates being able to work effectively as part of a team that may be international and geographically distributed. Students will also have the ability to lead and manage software engineering projects with international and geographically distributed teams." From the studio course, the team work learning objective was measured by the discussion forums, and team test plans and reports.

The assessment was completed by taking an anonymous random sampling of the artifacts needed from the delivered sections of the capstone the year prior. The test plans, report artifacts, and discussion forums were ranked by the MSE faculty members in the following categories: *exceeds*, *meets*, *partially meets*, and *does not meet expectations.* The faculty assessed the artifacts and discussed the rankings based on the criteria shown in Table 1. After the ranking the *teamwork* objective, it was determined that 90% of the students effectively participated in the forum. The test plans and reports reviewed were found to have all (100%) used continuous testing which is inherent in agile TDD.

5.2 Critical thinking

The *critical thinking* learning objective is defined as "the graduates will be able to critically and creatively plan and manage the development of software-intensive systems using project management methods and tools." The student should understand

Table 1. 2017–18 criteria for software engineering program assessment

Criteria
1. Identifies all schedule activities.
2. Identified all dependencies.
3. Accounts for worst, best, average case completion times.
4. Test driven development is employed.
5. Working code is delivered each week.
6. Unit testing tools are employed.

software project management methods, processes, and tools to create and allocate work packages. In addition, the teams should be able to monitor their progress to meet cost, quality, and schedule constraints, as well as understand the project governance responsibilities and practices. In terms of performance expectations from the capstone course, the critical thinking learning objective measures the student's ability to: (i) write an effective project plan; (ii) apply techniques to monitor cost and schedule performance; and (iii) create a concept of operations statement (CONOPS). The measurements for this learning objective via the capstone course was the team and individual weekly plan postings (backlogs), test reports, and CONOPS statements.

To assess the learning outcome of the critical thinking objective, the teams use agile software development in which project plans revolve around a "living" backlog. The faculty evaluated the project plans using the same categories (*exceeds*, *meets*, *partially meets*, and *does not meet expectations*) and the criteria shown in Table 2. It was found that 96% of individual student posting included elements of traditional plan-driven development. Moreover, team test plans and test reports were reviewed and found that 100% contained an appropriate CONOPS.

6. Discussion

Next, we discuss the learning experience according to students' point of view. We gathered feedback from all teams at the end of each sprint to improve the learning process, as well as feedback has been provided through the project review.

6.1 Lessons learned

Experience in Team communication. All groups adopted different venues for communication, which ranges from CANVAS messages, group discussions, Google Hangout to frequent live meetings bi-weekly. As means to explore the relation between group communication and accomplishments, the more frequent the meetings, the earlier the team can identify and resolve any impediments, which better help to track the work progress. Teams that had meetings 3 times a week could accomplish more in terms of higher number of functionalities delivered each iteration (Sprint).

Experience in Team collaboration. As part of the capstone learning curve, some students may not be familiar with the technologies adopted by the team. However, collaboration is the key factor in agile methodologies. Reportedly, groups where most experienced developers helped other team members with specific technologies and techniques and could resolve errors/bugs more quickly. Moreover, some teams dedicated a good amount of time learning new technologies (i.e. API's, frameworks, testing tools) required to make a successful project.

Experience using SCRUM Boards. Groups adopted many different solutions for project management (i.e. Asana, Azure DevOps, Trello). Most groups did a great job keeping the work progress updated. Some groups efficiently organized the use cases / user stories into EPIC's, which not only improves organization but helps splitting out the tasks and better distribute the workload. For example, usually, each team member is responsible for handling use cases / user stories related to a specific EPIC. Examples of generic EPICs created are Database, UI, Exception Handling, Wireframes, Login, and so forth.

Experience in Project Roles. Teams are usually good when defining the project roles, even though they might have some problems at the beginning to understand what the responsibilities of each role in the SCRUM process are. Furthermore, teams are encouraged to change roles with other colleagues during the development, even though the literature and industry advise a cautious approach [14].

Experience with Peer Evaluation. Peer assessment is a valuable tool to evaluate the individual contributions of team members. While the instructor might be able to evaluate individual contributions primarily from observations of synchronous meetings, the assessment provides additional information on what might take place outside those meetings. Students are encouraged to take peer evaluation seriously, as means to provide an honest assessment. Students should also provide an assessment of their own contribution.

Experiencing in Software Design/Architecture. Each group should design an initial architecture

Table 2. 2017–18 criteria for software engineering program assessment

Criteria
1. Plans follow a standard format for Agile backlog (i.e. contains all requisite elements).
2. Plans are complete, consistent and readable.
3. Plans are realistic.
4. Plans are written in professional manner.

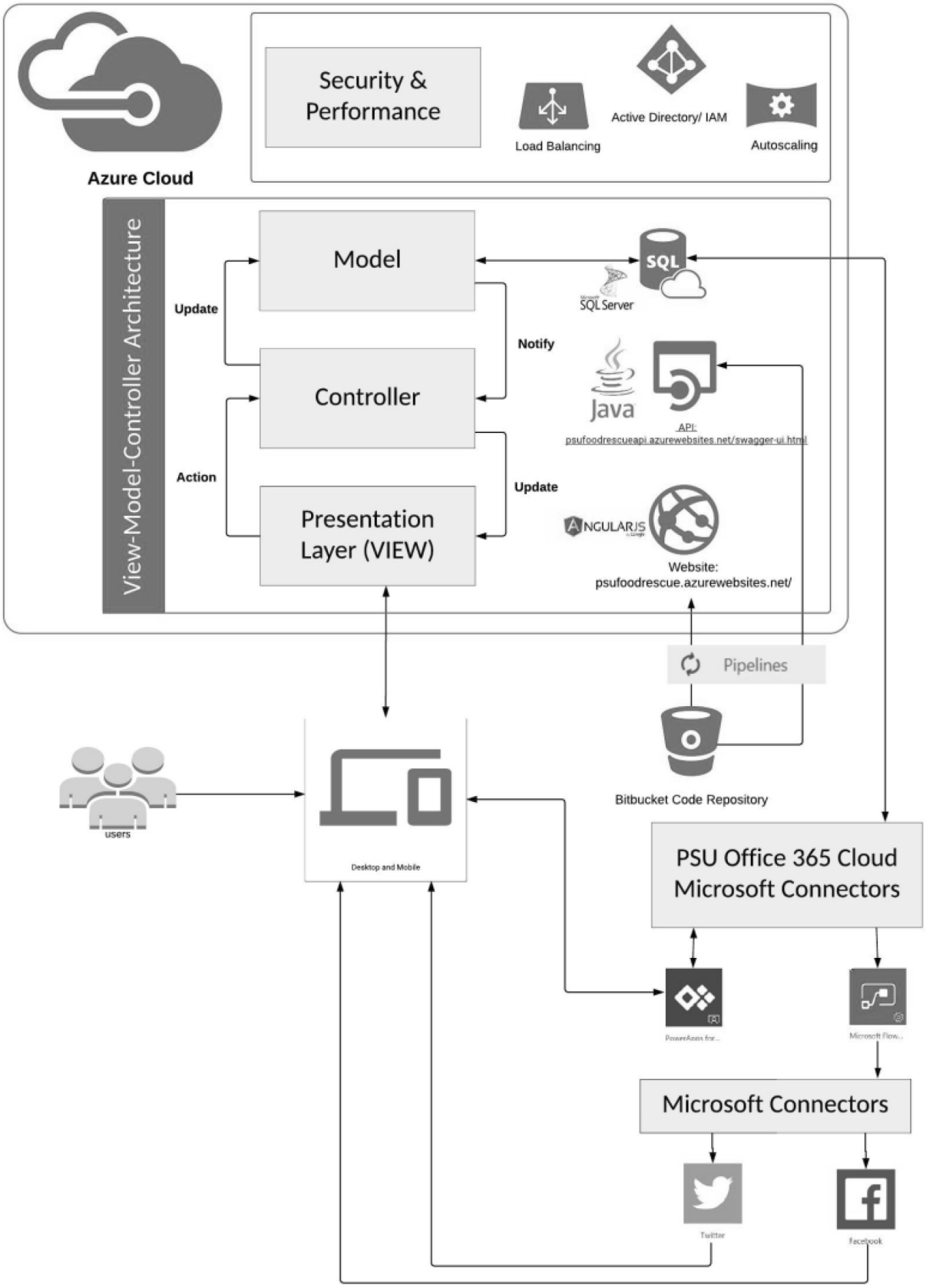

Fig. 2. High level architecture of the Food Rescue Application.

comprising all the fundamental elements of the application. The architecture described in Fig. 2 represents the architecture for one of the projects called Food Rescue. The application aims to be a bridge the gap between local food banks and businesses with a surplus of food and other items. In addition to supplying local families with necessities during hard times, donors can earn tax breaks from otherwise wasted inventory. The web application connects users in an efficient and easy to use interface. The popular Model—View—Controller (MVC) pattern was considered when designing our application. Data is housed in a Microsoft SQL database in the Azure Cloud and manipulated through JavaScript modules, Java services and stored procedures. Security architecture includes Microsoft Azure Active Directory and IAM for portal management and the use of Microsoft SQL for PSU Food Rescue site password encryption. All of this is hosted in the Microsoft Azure Cloud to be viewed responsively by users through multiple different devices thanks to the nature of AngularJS and the Bootstrap framework. Additionally, Microsoft Connectors are used to push out notifications containing user specific data.

Moreover, the team described the rationale concerning architecture decisions based on software quality attributes the solution provides support to (e.g., security, availability, scalability), and other design principles (e.g., SOLID, separation of concerns, patterns) contemplated in the architecture documentation. The solution demonstrates a good understanding and critical thinking on conceiving an architecture where students took into consideration good practices and current state of the art tools for building a software solution.

6.2 Limitations

Source code. One of the things impacting some projects is an inefficient configuration management process. Based on the data collected from Git repositories, only a few members are effectively collaborating. Moreover, teams with more than 3–4 members assigned as developers in Git could not properly manage conflicts in the code repository as it was found that portions of work have been overwritten or completely deleted. In some cases, changes were pushed to the master branch (used for production code) disregarding any testing activities, which should be of highest high priority given that teams should use TDD. Teams are strongly recommended to adopt configuration management measures as means to avoid conflicts and reduce the amount of rework. Moreover, teams are expected to appropriately test all new user cases/functionalities before pushing any changes.

SCRUM vs. Use Cases/User Stories. After reviewing all projects, a potential point of improvement is related to the requirements gathering. The first aspect would be reviewing the breakdown of user stories and how teams specify work breakdown structure. For example, user stories/use cases should ideally have well defined acceptance criteria as means to validate the requirements, and further it will also help when developing test cases. As most recent tools for project management supports the concept of EPICS, teams are encouraged to use EPICs as means of grouping user stories as well as have a proper work distribution. Each use case/user story can be further decomposed into smaller tasks, which gives the team a better sense of achievement. In some cases, the team might also want to adopt an issue tracking system and assign tickets to each subtask. Therefore, when all subtasks are completed, the ticket is then closed, and the use case/user story will be tested before integrating to the production version.

7. Recommendation

Two learning objectives of the MSE program were evaluated in terms of teamwork and critical thinking. Both were met with the course design as presented. The next course revision, shown in Fig. 3, will include artifacts (e.g., burndown chart) to address the lessons learned and improve this program learning outcomes from this program review.

An incidental finding of the course outcome review was that some students are not sufficiently capable in programming to fully participate in the project, even though students admitted to the program are expected to be competent. In response, the MSE program was modified to include an early software construction course. This course is essentially a mini capstone, with the difference that it requires individual work, and the project/requirements are already defined. Further in this course, the concentration/course objectives revolve around development/tools used in the capstone and programming.

8. Conclusion

Creating an effective studio course for professional students with heterogeneous experiences is challenging. The goal of the studio course was to create a meaningful experience while providing an environment for the students to apply knowledge learned from the program. Although the program objectives were met, the results indicate that not all elements of project planning are present in the studio course. The main contribution of this work is to share the evolution of this course since its inception 12 years ago and to describe the improve-

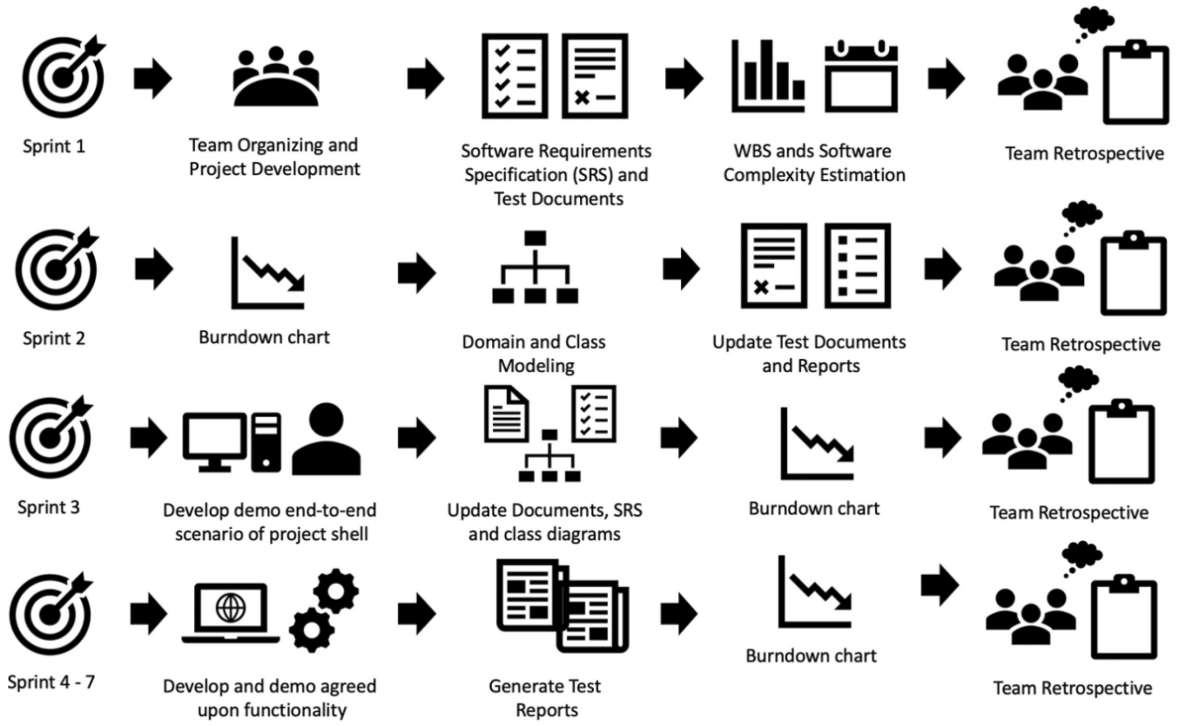

Fig. 3. Revised course structure based on program assessment.

ments made based on the results of the formal program assessment.

References

1. Viljan Mahnic, A Capstone Course on Agile Software Development using SCRUM, *IEEE Trans. on Education*, **55**, pp. 99–106, 2012.
2. J. Vanhanen, T. Lehtinen and C. Lassenius, Teaching Real-World Software Engineering through a Capstone Project Course with Industrial Customers, In *Proc. of the First Int'l Workshop on Soft. Eng. Education Based on Real-World Experiences*, pp. 29–32. IEEE Press, 2012.
3. I. Weissberger, A. Qureshi, A. Chowhan, E. Collins and D. Gallimore, Incorporating Software Maintenance in a Senior Capstone Project, *Int'l Journal of Cyber Society and Education*. **8**, pp. 31–38, 2015.
4. M. C. Bastarrica, D. Perovich and M. M. Samary, What Can Students Get from Software Engineering Capstone Course" In *Proc. of 39th Int'l Conf. on Software Eng*. (SEET-track), pp. 137–145, Argentina, 2017.
5. M. Balahan and A. Sturn, Software Engineering Lab: An Essential Component of Software Engineering Curriculum, In *Proc. of 40th Int'l Conf. on Soft. Eng*. (SEET-track), pp. 21–30, Sweden, 2018.
6. T. Ahtee and T. Poranen. "Risks in Students' Software Projects, In *Proc. of 22nd Conf. on Software Engineering Education and Training*, pp. 154–157, India, 2009.
7. S. Koolmanojwong and B. Boehm, A Look at Software Engineering Risks in a Team Project Course, In *Proc. of 26th Conference on Software Engineering Education and Training*, pp. 21–30, California, USA, 2013.
8. B. Bruegge, S. Krusche and L. Alperowitz, Software Engineering Project Courses with Industrial Clients, *ACM Trans. Comput. Education* (TOCE), **15**(4), 2015.
9. J. Vanhanen, T. Lehtinen and C. Lassenius, Software Engineering Problems and Their Relatioship to Perceived Learning and Customer Satisfaction Software Capstone Project, *Journal of Systems and Software*, **137**, pp. 50–66, 2018.
10. D. Ding, M. Yousef and X. Yue, A Case Study for Teaching Students Agile and SCRUM in Capstone Course, In *Proc. of Journal of Computing Sciences in Colleges,* **32***(5), pp. 95–101, May, 2017.*
11. P. Laplante, An Agile, Graduate, Software Studio Course, *IEEE Transactions on Education*, **49**, N1, November, 2006.
12. R. Sangwan and P. Laplante, Test-Driven Development in Large Projects, In *Proc. of Journal of IT Pro*, **8**(5), pp. 25–29, September, 2006.
13. D. Janzen and H. Saiedian, Test-Driven Development: Concepts, Taxonomy, and Future Direction, In *Proc. of Computer*, **38**(9), pp. 43–50, September, 2005.
14. K. T. Lyra, M. L. Alves, F. H. C. Silva, K. Souza and S. Isotani, An Agile Management Experience: Points of View of Graduate Students, In *Proc. XXXII Brazilian Symposium on Software Engineering (SBES)*, Sao Carlos, 2018.

Phillip A. Laplante, received a BS degree in systems planning and management, MEng degree in electrical engineering, and the PhD degree in computer science from the Stevens Institute of Technology, Hoboken, NJ, in 1983, 1986, and 1990, respectively, and the MBA degree from the University of Colorado, Colorado Springs, in 1999. He is a Professor of Software and Systems Engineering with Pennsylvania State University, Malvern. His research interests include real-time systems, real-time image processing, safety critical software systems and software quality.

Joanna F. DeFranco, earned her PhD in computer and information science from New Jersey Institute of Technology, MS in computer engineering from Villanova University, and a BS in Electrical Engineering and Math from Penn State

University. She is an Associate Professor of Software Engineering with the Pennsylvania State University. She has worked as an Electronics Engineer for the Navy as well as a Software Engineer at Motorola. Her research interests include technical teamwork, blockchain, and Internet of Things.

Everton Guimaraes, assistant professor of software engineering, holds a BS in information technology from the Federal Institute of Technology, Science and Education (IFRN), an M.S. in computing and systems from the Federal University of Rio Grande do Norte, and a PhD in sciences and informatics from the Pontifical Catholic University of Rio de Janeiro. Before joining Penn State Great Valley, Dr. Guimaraes worked as Assistant Professor at the University of Fortaleza, where he also joined the Software Engineering Research Group. He was also a Postdoctoral Fellow at Drexel University.