2021 ASEE ANNUAL CONFERENCE
Virtual Meeting | July 26–29, 2021 | Pacific Daylight Time
Paper ID #33201

Experiences of Teaching Software Testing in an Undergraduate Class Using Different Approaches for the Group Projects

Dr. Ingrid Buckley, Florida Gulf Coast University

Dr. Ingrid Buckley is an Assistant Professor in the Software Engineering Department at Florida Gulf Coast University. She holds a Ph.D. in Computer Science from Florida Atlantic University. Dr. Buckley's research interests include software engineering education, software quality, fault-tolerant system design, and cybersecurity education.

Dr. Peter J. Clarke, Florida International University

Peter J. Clarke received his B.Sc. degree in Computer Science and Mathematics from the University of the West Indies (Cave Hill) in 1987, M.S. degree from SUNY Binghamton University in 1996 and Ph.D. in Computer Science from Clemson University in 2003. His research interests are in the areas of software testing, software metrics, model-driven software development, domain-specific modeling languages, and computer science education. He is currently an associate professor in the School of Computing and Information Sciences at Florida International University. He is a member of the ACM (SIGSOFT, SIGCSE, and SIGAPP); IEEE Computer Society; and a member of the Association for Software Testing (AST).

©American Society for Engineering Education, 2021

Experiences of Teaching Software Testing in an Undergraduate Class using Different Approaches to the Group Project

Abstract

Software testing continues to be one of the main approaches used to validate software even as software systems become more complex. Although there is increased software testing throughout the software development life cycle in the industry, academic institutions continue to lag in offering software testing courses that adequately prepare students for jobs in the industry. In addition, academic institutions continue to grapple with how best to integrate software testing into their curricula.

This paper presents the experiences after teaching an undergraduate software testing course over four non-consecutive years, focusing mainly on the class project. During this period, different approaches were used to implement the project activities. These approaches are (a) testing different capstone projects that are sponsored by local companies, (b) all student teams testing a single project that is being developed at the same time, and (c) testing a single project that has been previously developed. Each approach presented several advantages and challenges for both the students and the instructor, as described in the paper. Based on the experience after teaching the course with different types of group projects, we present lessons learned and recommendations for future editions of the course. These recommendations include offering the software testing course earlier in the curriculum and implementing a peer-evaluation process for the group projects. These recommendations are consistent with other experience reports described in the literature.

1 Introduction

The ubiquity of software continues to grow as more systems are becoming software-dependent. These systems include small devices for IoT components to large systems that run video streaming applications [1]. The increased ubiquity of software has resulted in the need for more software engineers to service all aspects of the software development life cycle, including maintaining software quality. It is estimated that the poor quality of software for 2018 has resulted in a loss of more than $2.26 trillion to the US economy [2]. Software testing is one of the main approaches used to ensure software quality, even as software systems become more complex. As a result, the skill set for software testers has expanded over the past decade [3, 4]. Yet, academic institutions continue to lag in offering the number of software testing courses needed to prepare students for jobs in industry [5, 6, 7, 8].

Although many researchers and practitioners use innovative approaches to expose students to software testing techniques and tools [9, 10], academic institutions can do a better job integrating software testing into their curricula. Many of these innovative approaches involve: using active learning in the classroom [11, 12, 13]; using gamification in software testing classes [14, 15]; providing supplemental learning materials online that support the use of testing tools [16]; improving testing courses by identifying the difficulties and challenges students encountered when performing testing [17]; and, testing real-world projects as a significant part of the software testing course [18].

This paper presents the experiences of teaching an undergraduate software testing course over four semesters using three different approaches for the project component of the course. A major component of the course is a semester-long group project with several deliverables. The nature of the group projects includes (a) testing different capstone projects that are sponsored by local companies, (b) all student teams testing a single project that is being developed during the semester, and (c) testing a single project that has been previously developed. Several educators recommend testing projects that local companies sponsor. Such projects expose students to real-world experiences and are supported by the principles of problem-based learning [19]. However, such an approach must be adequately managed since industry practices, and goals can differ from those of teaching a course in academia. The contributions of the paper include:

1. A description of the software testing course and how it interfaces with other courses in the undergraduate course sequence is provided.
2. Presenting data on the three different types of projects used in the testing course.
3. Identifying the advantages and challenges of using each type of project.
4. Presenting the lessons learned and recommendations going forward.

The paper is organized as follows. Section 2 presents background on testing and related work. Section 3 describes the course structure, including the group project component, and Section 4 describes the different group projects used in the course. Section 5 presents the data collected for the group projects for each edition of the course over a four-year period. Section 6 describes the experiences, lessons learned and recommendations going forward, and we conclude in Section 7.

2 Literature Review

This section presents a short overview of software testing and the work most closely related to the experiences gained from teaching undergraduate software testing.

2.1 Software Testing

Software testing is defined as the *dynamic verification of the behavior of a program on a finite set of test cases suitably selected from the usually infinite execution domain, against the expected behavior* [20]. The most challenging part of testing is to find the smallest subset of the input domain that represents the behavior of the usually infinite domain. This objective has resulted in a variety of testing techniques, testing levels, testing-related measures, and testing processes [20]. To adequately support the automation of software testing activities, a wide variety of testing tools have been developed [21].

The testing techniques we focus on are *black-box* - focuses on the input/output behavior of the entity being tested, and *white-box* - focuses on the internal properties of the entity being tested, e.g., statement coverage and branch coverage [22]. The testing levels include *unit* - where individual classes are tested, *integration* - occurs after unit testing and where multiple classes are tested together to form a subsystem, and *system* - where the entire system is validated against the user requirements. One other form of testing mentioned in the paper is *regression* testing, where the software is tested after updating an already tested artifact due to requirement changes and maintenance activities.

2.2 Related Work

Aniche et al. [17] propose a curriculum that uses a pragmatic approach to testing based on students' common mistakes, their perception of topics that are difficult to learn, the learning activities they enjoyed the most, and challenges they faced when studying software testing. They analyzed the feedback reports from 230 students in the first year software quality and testing course and surveyed results from 84 students and seven teaching assistants. The analysis of the reports based on the eight themes identified is as follows. The top three common mistakes included not effectively using test coverage to write more effective test cases, the maintainability of the test code that was written, and understanding testing concepts. The survey data showed that students found the use of JUnit [23], the Act-Arrange-Assert pattern for unit tests, and choosing testing levels the easiest to learn. In contrast, the most difficult topics included identifying the minimum set of tests, avoiding flaky tests, and exploratory testing. Unlike the work presented by Aniche et al. [17], the experiences presented in this paper are based on different editions of a final year software testing course, with a focus on how the class project was implemented. That is, we present details on the different approaches used to incorporate testing in the project while at the same time introducing students to testing concepts and the tools to support testing.

Krutz et al. [18] describe an approach to teaching software testing where the project is the main component of the course. The course outcomes include stressing the importance of testing, exposing students to testing tools and techniques, and providing students with the opportunity to make informed testing decisions. A student survey was conducted at the end of the term, and a majority of students enjoyed the project component and would recommend the course to their classmates. Students also reported that taking the course enhanced their chances of getting a co-op opportunity or a full-time job. The software testing course described by Krutz et al. [18] focuses on the project component, which is similar to our approach, except they describe the implementation of the most recent version of the project. In our work, we describe three versions of the class project identifying the challenges and benefits of each version. We also provide a list of the testing tools used with the different projects and the challenges encountered with each project approach.

de Andrade et al. [9] report on their experiences teaching two undergraduate software testing courses at two different institutions, one for Computer Science students (full-time students) and the other for Information Systems students (part-time students). The authors used a Problem-Based Learning (PBL) approach with real open-source projects to get students motivated and more engaged in learning software testing. Students provided feedback on their perceptions of the courses using anonymous questionnaires. The results in both courses showed a partial improvement of the content acquired by the students, and most students viewed the active learning approach as positive. Due to using the PBL approach, the instructors provided extra classes. However, this was difficult for the part-time students. Although the focus of the work by de Andrade et al. [9] focuses on using a different pedagogical technique to motivate students' interest in software testing, they also investigated the problems of teaching testing with a real-world problem. Several challenges were identified using projects from the industry that are similar to one of the approaches we presented in this paper, e.g., the confidentiality of the project.

Heckman et al. [24] describe their experiences with integrating software testing into the undergraduate curriculum at their institution for the past ten years. The authors focus on CS1 - Introductory Programming, CS2 - Software Development Fundamentals, and other courses beyond CS2. In the

CS1 course, testing is formally introduced near the middle of the semester and covers black-box testing, white-box testing, and using the JUnit [23] testing tool. In the CS2 course, the testing topics covered were unit, integration, system, acceptance, and regression testing, and test coverage. Additional tools used were JaCoCo [25] and FindBugs [26]. Recommendations by the authors include (a) integrate testing throughout the curriculum, (b) introduce students to testing tools early, (c) use common tools and language across the curriculum, (d) provide appropriate examples to show the need for testing and debugging, and (e) use automation for rapid feedback. Unlike Heckman et al. [24] we only compared the implementation of the class project in the testing course for three editions of the course, using different approaches for the project. In addition, there is little or no applied software testing in courses taken by students before the testing course. Based on the experiences of teaching the testing course reported in this paper, several of our recommendations are similar to the ones presented by Heckman et al. [24].

3 Software Testing Course

Florida Gulf Coast University offers two sections of an undergraduate software testing course, CEN4072, each spring semester. CEN4072 is a three-credit required hybrid course (50% face-to-face and 50% online) and is offered to seniors in the last semester of their degree program. One section of CEN4072 is offered on Tuesday and the other on Thursday. The course is structured as a lecture with no lab component, students complete assignments online, and a group project is worked on mainly outside of lecture time. The prerequisites for this course are Software Specifications, and Data Structures. The course learning objectives for CEN4072 are as follows. Students should be able to:

1. Perform unit-testing and identify possible faults in code
2. Perform integration-testing and continuous integration testing on computer programs
3. Apply test coverage criteria, such as code coverage, path coverage, and input partition coverage to code
4. Apply test-driven development and different levels of software testing, such as unit, integration, functional, acceptance testing, and regression testing during software development
5. Describe the ethical issues related to software testing

Students are assessed using various assignments, exams, quizzes, activities, and a semester-long group project. The grading scheme for the course (100%) is estimated to be: quizzes/activities/discussions (10%-30%), the group project (10%-40%), and exams (30%-60%). Note that in some years, the grading scheme would change slightly depending on the approach used for the project. The textbook currently used in the course is "A Practitioners Guide to Software Test Design" by Lee [22]. Other reading materials include class notes and tutorials on testing tools.

The instructor used various approaches to teach the software testing class over a four-year period (2016, 2017, 2019, and 2020). The instructor did not teach software testing CEN4072 in 2018. The instructor illustrates how to apply different software testing principles and strategies to solve common real-world problems during lectures. Emphasis is placed on using various software testing techniques and strategies to develop test cases, create a test design plan, evaluate the effectiveness of test cases, and use automated software testing tools to perform different levels of testing. The main difference between these years was the complexity and scope of the group project.

During 2016-2019 the only prerequisite for the CEN4072 was Software Specifications; however,

a recommendation was made in 2020 to add Data Structures as a second prerequisite. This change was necessary because students could register for the CEN4072 class but were not equipped with the essential programming skill-sets and knowledge to handle the course material. Additionally, it is challenging to find Teaching Assistants (TA) for CEN4072 because it is offered in the final semester of the degree program. As a result, there were no available TAs in 2019 and 2020. Over the four years, two TAs were hired to assist with grading assignments, one in 2016 and the other in 2017, respectively. These TAs were available because they did not graduate after completing CEN4072 in Spring 2015 and 2016. Currently, there is no established graduate program in the department or graduate students from other departments who would have the skills or knowledge to assist the CEN4072 instructor.

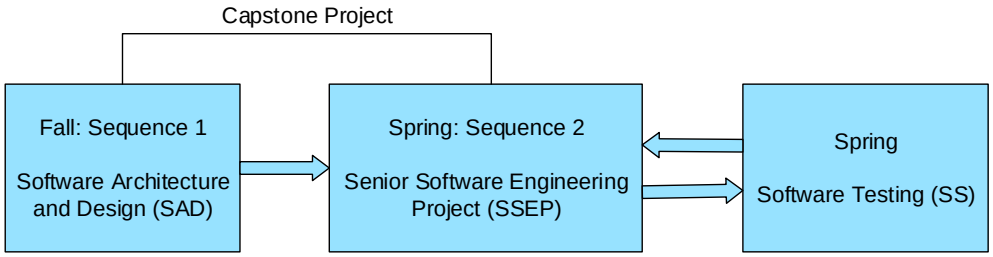

Figure 1: Course sequence including software testing.

In 2017 and 2019, the group project component was based on the senior capstone project. Seniors completed a capstone project, which is offered in the fall and spring semesters in a two-course sequence, (a) Software Architecture and Design (SAD) and (b) Course Senior Software Engineering Project (SSEP), respectively see Figure 1. Students complete the software requirements, design, and architecture work in the fall in the SAD course and continue with the system implementation in the SSEP course in the spring. The projects from the SSEP course are tested in CEN4072 as depicted by the double arrows on the right of Figure 1. Due to the COVID-19 pandemic, the face-to-face component of the testing course was changed to a remote synchronous online (virtual) modality in Spring 2020.

4 Class Projects

The CEN4072 Software Testing course is primarily a project-based course. Over the four years that the instructor taught the course, the group project's percentage weight and nature were different. In 2016, the group project was selected by the instructor, the students were allowed to create their teams, and the group project was worth 20% of the overall course grade. The group project in 2016 involved both development and testing. In 2020, the group project was also a standalone project that the instructor selected with a percentage weight of 40%. In 2020 the group project involved testing only.

In 2017 and 2019, the senior capstone project was used as the group project in CEN4072. The capstone projects were diverse in both scope and complexity, and the teams worked on different types of projects using a cross-section of languages and tools. The team size varied because the teams were formed in a previous semester, and the capstone project is completed over two semesters (fall and spring). It should be noted that the group project implementation was interleaved with testing in CEN4072 as shown in Figure 1. The group project percentage weight in the overall grade was 10% and 40% in 2017 and 2019, respectively. In 2017, the relatively small weighting was used

since this was the first year the capstone projects were being integrated into the software testing course. There were many unknowns about how successful this integration would be.

Every project consisted of four general deliverables that each team was expected to submit at different points during the spring semester. Each deliverable represented a milestone in the testing process. Throughout the semester, each group presented the work they completed for each of the deliverables and obtained feedback from the instructor about improving or correcting mistakes. These deliverable are described in more detail in Sections 4.1 and 4.2. Samples of the class projects are presented in Section 4.3.

4.1 Different Capstone Projects

Seniors have multiple project-based courses in their final semester of the Software Engineering degree program. Seniors in previous years shared their recommendations to improve the program in the student exit surveys. One of the main recommendations was to use the same group project in multiple courses to reduce the workload of graduating seniors. As a result, the capstone project was integrated into the CEN4072 Software Testing starting in 2017. The capstone projects varied in terms of programming language, tools, scope, and complexity. The projects spanned different domains, including but not limited to mobile, web, Cloud-services, data mining, aerospace, gaming/virtual reality, e-commerce, and medical applications. The capstone projects were also done in collaboration with industry partners. Testing the capstone projects in the software testing course presented several challenges as described below.

Student registration. Each spring, two sections of CEN4072 are offered; each section is capped between 35 to 42 students. Because students register online, it wasn't easy to maintain the pre-established teams in the same class section. Team assignment was a problem since team presentations and demonstrations were done during the class lectures. As a result, it was the instructor's responsibility to place students into the same course section with their team members. This placement became a major scheduling and time-intensive task because students have different work schedules, class schedules, and other personal commitments.

Varying scope and complexity of each capstone project. For some teams, the project was very straightforward to test because the required tools and other resources were readily available. While other teams struggled to find tools to conduct automated unit testing and obtain statement and branch coverage results. Additionally, because the capstone projects are being developed and tested in parallel with the stakeholders, requirements and circumstances often changed, delaying the testing efforts for some teams.

Non-Disclosure Agreements (NDAs). Due to the nature of the collaborative efforts with the industry partners, students were required to sign NDAs with local companies who partnered on the senior capstone projects. Using NDAs meant that the instructor could not have access to the software or data being tested. Students had to demonstrate the software, tools, and development environment on their respective computer systems. The project demonstration would include executing the test cases and showing the coverage results (statement and branch) obtained for each test set.

Number of capstone projects. Overall, there were 30 capstone group projects, 14 in 2017 and 16 in 2019 (see Table 1). The teams consisted of between two to five students; as a result, the workload for each student varied from team to team. In order to perform unit and integration testing and to

capture coverage results, teams used over 40 different testing tools in 2017 and 2019 combined (see Table 2).

The contents of the deliverables for testing *different* capstone group projects with the *development and testing phases interleaved* are as follows:

Deliverable 1 Define testable requirements, install software testing tools, and demonstrate the current version of the software product.

Deliverable 2 Continue development, create a test design plan, and perform continuous unit testing and regression testing.

Deliverable 3 Continue development, update test design plan, perform continuous integration testing and regression testing

Deliverable 4 Complete development, complete test design plan, and perform system testing

Since the capstone projects involved industry stakeholders, the teams dealt with changing requirements and programming languages late into the spring semester. These changes impacted the quality of the testing that was performed.

4.2 Similar Projects

In 2016 and 2020, students worked on the same type of group project. There were 14 and 21 teams in 2016 and 2020, respectively (see Table 1). In 2016 students developed and tested an *Addressbook* Java application that allowed a user to create new contacts via user input or by uploading a file of contacts, search, update and delete contacts in the address book. Students formed teams with no more than four students in each team. In 2016 some teams found it challenging to develop and test the application in one semester despite not having issues with tools and technical issues.

In 2020 changes were made where students were randomly placed in teams not exceeding four students, and they were given a previously developed AddressBook Java application to test. Students were required to figure out how to configure and set up the environment to execute the application and then perform unit and integration testing. This approach in 2020 proved to be the best thus far for both students and the instructor. Since the projects required no development, students could concentrate on finding defects in the code and not the lack of tool support or time devoted to development. It also allowed students to gain more experience working in a team, getting hands-on experience testing a medium-sized software product, and using testing tools effectively to test software. Using the 2020 project approach in the testing class allowed the instructor to provide more effective help with the tools used while testing the software.

The contents of the deliverables for all teams working on the *same project* with the *development and testing phases interleaved* are as follows:

Deliverable 1 Define testable requirements, install software testing tools, and demonstrate the current version of the software product.

Deliverable 2 Continue development, create a test design plan, and perform continuous unit testing and regression testing

Deliverable 3 Continue development, update test design plan, and perform continuous integration testing and regression testing

Deliverable 4 Complete development, complete test design plan, and perform system testing

The contents of the deliverables for all teams working on the *same project* with *only testing phases* are as follows:

Deliverable 1 Define testable requirements, install software testing tools, and demonstrate the execution of the existing software product.
Deliverable 2 Create a test design plan and perform continuous unit testing and regression testing
Deliverable 3 Update test design plan, and perform continuous integration testing and regression testing
Deliverable 4 Complete the test design plan and perform system testing

Note that developing and testing the same project sometimes involved changing requirements; on the other hand, testing the same project did not involve changing requirements, thereby making the testing process a lot easier.

4.3 Sample Projects

This section summarizes some of the group projects used for testing in CEN4072 in 2016, 2017, 2019, and 2020. The summary includes the semester (year) the project was done, the project name, a short description of the project, the language(s) used, and development and testing tools used. In 2016 and 2020, the group project was the AddressBook application. In 2017 and 2019, students developed and tested a variety of applications.

Semester: Spring 2016
Project Name: AddressBook
Description: An Address Book application that allows users to keep all their contacts in one place and offer diverse functionality to create, modify, sort or delete and upload contacts from a file.
Languages: Java
Development Tools: IntelliJ IDEA [27], Eclipse IDE [28]
Testing Tools: JUnit [29], TestFX [30], EasyMock [31], PowerMock [32], IntelliJ Code Coverage [33], EclEmma [34]

Semester: Spring 2017
Project Name: The Frozen Fortune
Description: A 2D video game.
Languages: C#
Development Tools: Unity Game Engine [35], Microsoft Visual Studio [36]
Testing Tools: NUnit [37], Ncover [38], Nsubstitute [39]

Semester: Spring 2019
Project Name: Rewards & Loyalty Program
Description: A mobile application that allow users to sign up to a rewards program and keep track of the user's loyalty point.
Languages: JavaScript
Development Tools: Microsoft Visual Studio [36]
Testing Tools: Jest [40], Enzyme [41]

Semester: Spring 2020
Project Name: AddressBook
Description: See Semester 2016.

Languages: Java
Development Tools: IntelliJ IDEA [27], NetBeans IDE [42], Microsoft Visual Studio [36], Scene Builder [43], SQLite [44], and Travis CI [45]
Testing Tools: JUnit [23], TestFX [30], IntelliJ Code Coverage [33], Mockito [46], Cucumber [47], and AssertJ [48], JaCoCo [25], CodeCov [49]

In Spring 2020, the development tools were only used to get the application running so that it could be tested. In addition, students were not restricted in terms of the tools they wanted to use. As a result, different teams used a variety of tools to test the same project.

5 Data Collected

This section presents data related to the CEN4072 Software Testing class projects for each of the four editions of the course between 2016 and 2020.

5.1 Data on Testing Projects

Over the four-year period, 186 students registered for CEN4072, and 65 teams worked on group projects. The teams used various programming languages, tools, and technologies to implement and test each project depending on the nature of the group project. Table 1 shows the data for each year, including the total student enrollment, number of teams, the type of project, and the complexity of the project. In 2016 all student teams developed and tested the AddressBook application, and in 2020, students only tested the AdressBook application. In 2017 and 2019, students tested the project they were currently working on in the senior capstone course.

The project grade data in Table 1 shows that the grades for the project progressively improved each year the course was taught. In addition, the spread of the project grades was the smallest in 2020, unlike the other years when the spread of grades was at least three times more than in 2020. This slight grade deviation may suggest that more teams were meeting the objectives when testing the project.

Table 1: Summary of project data and student performance.

Year	Enroll.	No. Teams	Project Type	Complexity	Project Grade (%) M (SD)
2016	40	14	AddressBook Application (Java)	Development and Testing	65.2 (18.5)
2017	35	14	Capstone Project (different projects)	Development and Testing	82.8 (20.3)
2019	50	16	Capstone Project (different projects)	Development and Testing	85.0 (18.3)
2020	61	21	AddressBook Application (Java)	Testing Only	89.9 (5.1)
Total	186	65			

Table 2 shows the number of different programming languages and tools (development and testing) used by the teams during the project. Students used significantly more and a wider variety of other software testing tools when working on the capstone projects than the students who worked

on the same group project. In 2016, students struggled with implementing the AddressBook application, which reduced the focus on software testing. As previously mentioned in Section 4.1, the challenges in 2017 and 2019 include the teams struggling because of the diverse nature of the capstone projects, implementing the project with changing software requirements, difficulty finding adequate open-source software testing tools, and the lack of tool support and resources.

Table 2: Number of different programming languages and tools used by the teams during the project.

Year	No. Prog. Langs Used	No. Different Tools Used			
		Dev.	Unit Testing	Integration Testing	Coverage
2016	1	2	2	3	2
2017	5	10	10	7	4
2019	6	13	13	7	9
2020	1	6	5	3	4
Total	6	22	22	18	13

After removing the implementation aspect of the group project in 2020 and all teams working on the same group project, the outcomes of the project improved. Students were able to focus primarily on software testing, installing tools, and setting up the testing environments; this resulted in improved code coverage (branch and statement coverage). Another factor that improved the quality of the project was the use of a single implementation language. Using a popular common programming language, e.g., Java, provided students access to many software testing tools and resources that are widely accessible. Furthermore, because students worked on the same group project, they had more support from the instructor and their peers. In contrast, students who had to test the diverse capstone projects were limited to the help they received from their team members and stakeholders.

The data in Table 3 shows the number of teams exceeding the statement and branch coverage requirements stipulated by the instructor, this further supports the improved outcomes of the project. The criteria for code coverage were greater than or equal to 85% statement coverage and greater than or equal to 80% branch coverage. In 2020, 100% of the teams were able to meet the stipulated code coverage requirements. In the other years, less than 70% of the teams achieved the specified statement coverage criterion, and less than 57% met the specified branch coverage criterion. It is worth noting that initially, students complained about setting up the testing tools and learning how to use them. However, once they became accustomed to the environment and tools, they became more proficient in developing the test cases and test suites.

6 Experiences and Lessons Learned

This section presents the experiences gained, lessons learned, and recommendations for teaching the CEN4072 Software Testing course in the future, focusing on the project component of the course.

6.1 Observations and Lessons Learned

The three different approaches used for the group project in CEN4072 provided insights into what worked and did not work well when teaching software testing. Our observations and lessons

Table 3: Number of teams exceeding the statement coverage and branch coverage requirements.

Year	No. Teams	No. of Teams with	
		Stmnt Coverage (>=85%)	Branch Coverage (>=80%)
2016	14	9 (64%)	4 (29%)
2017	14	8 (57%)	6 (43%)
2019	16	11 (69%)	9 (56%)
2020	21	21 (100%)	21 (100%)

learned are presented as a list of advantages and challenges for each approach used for testing group projects from the instructor's point of view. Due to the paper's length restriction, we limit the advantages and challenges to the top three for each approach.

Approach 1: Same Group Project - with the development and testing phases interleaved.
Advantages:

1. Students had more peer support and could problem solve technical issues with other teams.
2. Students had access to software testing tools and resources.
3. Students received an in-depth understanding of the software product because they were implementing it.

Challenges:

1. Students struggled with programming-related skills. As a result, too much time was spent on implementing the application.
2. Some team members preferred to avoid the technical work and only contributed to the writing of the reports.
3. Students struggled to meet the statement and branch coverage requirements for testing.

Approach 2: Capstone Group Project - with the development and testing phases interleaved.
Advantages:

1. Students were motivated about working on the capstone project because they worked with local companies that could potentially employ them at the end of the semester.
2. Students were exposed to a variety of technologies, domains, tools, and business processes.
3. Students were forced to learn how to develop different types of applications without support from their peers.

Challenges:

1. Due to the diverse nature of capstone projects, students had limited access to software testing tool support and resources.
2. Students had to sign Non-Disclosure Agreements (NDAs); this meant that the software code and data associated with the group project were not accessible by the instructor in CEN4072.
3. Students struggled to meet the statement and branch coverage requirements.

Approach 3: Same Group Project- with testing only.

Advantages:

1. Same as Approach 1 bullets (1) and (2)
2. Students had more time to focus on testing only since the group project was already implemented.
3. Students were able to meet the statement and branch coverage requirements for the group project.

Challenges:

1. Some team members preferred to avoid participating in technical work and only contributed to writing the reports.
2. Students got less exposure to different technologies, domains, and software development and testing tools.
3. Students got less implementation practice.

6.2 Recommendations

After teaching the CEN4072 course using the three different approaches for the group project, the instructor proposed the following recommendations in the future. Some of these recommendations have already been implemented in the 2020 version of the CEN4072 Software Testing course.

1. *Students should take the CEN4072 course earlier in the Software Engineering Degree program*. CEN4072 is offered in the last semester of the Software Engineering degree program; thus, this time frame is very stressful for graduating seniors. Typically, students prioritize job hunting and interviews and seem less motivated to use their time to obtain a high grade in the software testing course.

2. *Students should be exposed to using testing tools earlier in their program of study*. Students find it overwhelming to be learning all the testing tools required for unit testing, integration testing, system testing, and code coverage at the same time.

3. *Students need to have more advanced problem solving and algorithmic skills before taking CEN4072 (implemented)*. Even though students do not have to implement a software product, they still need strong programming and problem-solving skills to write effective test cases and test suites. In addition, these skills will help students to understand better many of the concepts based on graphs, e.g., control flow graphs and cyclomatic complexity. As a result, the data structures course was added as a prerequisite to CEN4072.

4. *Use a more rigorous peer-evaluation process for the group projects (implemented)*. To accomplish this, all teams complete a peer- and self-evaluation for each significant project deliverable, which includes: assembling the software product, tool installation and environment set-up, unit testing, integration testing, and system testing. This type of evaluation allows students to obtain a more equitable grade on the group project based on their contribution to each project deliverable.

5. *Assigning the same group project to all teams in the class (implemented)*. Although students could gain valuable industry experience by working with sponsors/local companies, their goals do not necessarily align well with the software testing course's traditional require-

ments and objectives. The non-disclosure agreements that the project sponsors established introduced additional restrictions and limitations on students and the instructor. To better meet the course objectives and create a stable learning environment for students, it proved better for the instructor to assign the same group project to teach software testing.

6. *Demonstration of using tools to test the software project during the instructor evaluation*. To ensure all students are involved in the technical work when testing the software, individual students should demonstrate using the testing tools and running test cases on the software. This demonstration could be part of the final instructor evaluation for the project. Hopefully, this will encourage all students to work more on the technical aspects of the project, such as installing the tools and performing unit, integration, and system testing.

7. *Limit the number of students in each section of the course*. Software testing is a complex technical hands-on course that is challenging to teach. As a result, the instructor recommends that the class size be restricted to 30 students in each course section. In general, smaller class sizes provide many benefits to students. This change would allow the instructor to allocate more time to help individual students or teams with problem-solving and technical issues and accommodate students experiencing extenuating circumstances.

Several of our recommendations are similar to the ones presented by Heckman et al. [24]. Software testing should not be a single course provided to students when they are about to graduate; testing should be integrated and applied throughout the curriculum, in addition to having a separate testing course. Students being exposed to software testing tools earlier would provide an easier transition, better tool proficiency, more comprehensive and in-depth coverage of testing topics, rather than exposing all of these aspects in one course, in the final semester of the undergraduate program.

7 Conclusion

To effectively teach undergraduates how to apply software testing principles and techniques and use various tools to test a software application adequately, the course should have a project component. In addition, the student assessments need to be more comprehensive. Assessments that are smaller in scope, such as assignments and labs, do not provide the appropriate rigor or opportunity to perform the testing level required in the industry. This paper presented the experiences of teaching a software testing course that involved three different approaches to the project component over a four-year period.

Based on the instructor's experiences teaching the course over a four-year period, observations and lessons learned are presented, and recommendations are given for future editions of the course. These recommendations include: (a) the software testing course should be offered earlier in the program of study, not in the final semester; (b) students should be exposed to testing and testing tools earlier in the curriculum, e.g., CS1 and CS2; (c) students should be exposed to advanced data structures before taking the software testing course; (d) student teams should be assigned the same group project; and (e) students should be encouraged to use a core set of software testing tools. These recommendations will improve student learning and reduce the overall workload for the students and the instructor in the software testing course. Several of these recommendations are consistent with those identified in other experience reports found in the literature.

Acknowledgment

Partial support for this work was provided by the National Science Foundation's Improving Undergraduate STEM Education (IUSE) program under Award Numbers DUE-1562773 and DUE-1525112. Any opinions, findings, and conclusions expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The authors would like to thank the reviewers for their thoughtful and encouraging feedback on improving the paper.

References

[1] C. Ebert and S. Counsell, "Toward software technology 2050," *IEEE Software*, vol. 34, no. 4, pp. 82–88, 2017.

[2] H. Krasner, "The cost of poor quality software in the us: A 2018 report," Consortium for IT Software Quality (CISQ), September 2018, https://cra.org/data/Generation-CS/ (retrieved August, 2020).

[3] R. Florea and V. Stray, "A global view on the hard skills and testing tools in software testing," in *Proceedings of the 14th International Conference on Global Software Engineering*, ser. ICGSE '19. IEEE Press, 2019, p. 133–141. [Online]. Available: https://doi-org.ezproxy.fiu.edu/10.1109/ICGSE.2019.00036

[4] M. Kassab, J. F. DeFranco, and P. A. Laplante, "Software testing: The state of the practice," *IEEE Software*, vol. 34, no. 5, pp. 46–52, 2017.

[5] A. Arcuri, "An experience report on applying software testing academic results in industry: we need usable automated test generation," *Empirical Software Engineering*, vol. 23, no. 4, pp. 1959–1981, 2018.

[6] M. Craig, P. Conrad, D. Lynch, N. Lee, and L. Anthony, "Listening to early career software developers," *J. Comput. Sci. Coll.*, vol. 33, no. 4, p. 138–149, Apr. 2018.

[7] M. Felderer and R. Ramler, "Special issue on collaboration in software testing between industry and academia," *Software Quality Journal*, vol. 25, no. 4, pp. 1087–1089, 2017.

[8] R. Pham, S. Kiesling, L. Singer, and K. Schneider, "Onboarding inexperienced developers: struggles and perceptions regarding automated testing," *Software Quality Journal*, vol. 25, no. 4, pp. 1239–1268, 2017.

[9] S. a. A. de Andrade, V. de Oliveira Neves, and M. E. Delamaro, "Software testing education: Dreams and challenges when bringing academia and industry closer together," in *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, ser. SBES 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 47–56. [Online]. Available: https://doi-org.ezproxy.fiu.edu/10.1145/3350768.3353903

[10] T. Hynninen, J. Kasurinen, A. Knutas, and O. Taipale, "Guidelines for software testing education objectives from industry practices with a constructive alignment approach," in *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 278–283. [Online]. Available: https://doi-org.ezproxy.fiu.edu/10.1145/3197091.3197108

[11] P. J. Clarke, D. L. Davis, I. A. Buckley, G. Potvin, M. Thirunarayanan, and E. L. Jones, "An approach to integrating learning and engagement strategies (LESs) into CS class activities," in *126th American Society for Engineering Education (ASEE)*. Washington DC: ASEE, 2019, paper No. 26116.

[12] R. Smith, T. Tang, J. Warren, and S. Rixner, "An automated system for interactively learning software testing," in *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 98–103. [Online]. Available: https://doi-org.ezproxy.fiu.edu/10.1145/3059009.3059022

[13] S. Tiwari, V. Saini, P. Singh, and A. Sureka, "A case study on the application of case-based learning in software testing," in *Proceedings of the 11th Innovations in Software Engineering Conference*, ser. ISEC '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi-org.ezproxy.fiu.edu/10.1145/3172871.3172881

[14] G. M. de Jesus, L. N. Paschoal, F. C. Ferrari, and S. R. S. Souza, "Is it worth using gamification on software testing education? an experience report," in *Proceedings of the XVIII Brazilian Symposium on Software Quality*, ser. SBQS'19. New York, NY, USA: Association for Computing Machinery, 2019, p. 178–187. [Online]. Available: https://doi-org.ezproxy.fiu.edu/10.1145/3364641.3364661

[15] G. Fraser, A. Gambi, M. Kreis, and J. M. Rojas, "Gamifying a software testing course with code defenders," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 571–577. [Online]. Available: https://doi-org.ezproxy.fiu.edu/10.1145/3287324.3287471

[16] P. J. Clarke, D. L. Davis, R. Chang-Lau, and T. M. King, "Impact of using tools in an undergraduate software testing course supported by WReSTT," *ACM Trans. Comput. Educ.*, vol. 17, no. 4, Aug. 2017. [Online]. Available: https://doi-org.ezproxy.fiu.edu/10.1145/3068324

[17] M. Aniche, F. Hermans, and A. van Deursen, "Pragmatic software testing education," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 414–420. [Online]. Available: https://doi-org.ezproxy.fiu.edu/10.1145/3287324.3287461

[18] D. E. Krutz, S. A. Malachowsky, and T. Reichlmayr, "Using a real world project in a software testing course," in *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 49–54. [Online]. Available: https://doi-org.ezproxy.fiu.edu/10.1145/2538862.2538955

[19] A. Walker, H. Leary, C. Hmelo-Silver, and P. Ertmer, Eds., *Essential Readings in Problem-Based Learning: Exploring and Extending the Legacy of Howard S. Barrows*, ser. EBL-Schweitzer. Purdue University Press, 2015. [Online]. Available: https://books.google.com/books?id=8zWyswEACAAJ

[20] P. Bourque and R. Fairley, Eds., *Guide to the Software Engineering Body of Knowledge, Version 3.0*. USA: IEEE Computer Society, 2014, www.swebok.org (Accesses August 2020).

[21] Wikipedia, "Category:software testing tools," 2020, https://en.wikipedia.org/wiki/Category:Software_testing_tools (Accessed August 2020).

[22] L. Copeland, *A Practitioner's Guide to Software Test Design*. USA: Artech House, Inc., 2003.

[23] The JUnit Team, "JUnit 5," 2020, http://www.junit.org/ (Accessed August 2020).

[24] S. Heckman, J. Y. Schmidt, and J. King, "Integrating testing throughout the cs curriculum," in *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2020, pp. 441–444.

[25] EclEmma Development Team, "JaCoCo," 2020, https://www.eclemma.org/jacoco/ (Accessed August, 2020).

[26] FindBugs Development Team, "FindBugs," 2020, http://findbugs.sourceforge.net/ (accessed August 2020).

[27] JetBrains s.r.o., "Intellij idea - community edition," 2020, https://www.jetbrains.com/idea/ (Accessed August, 2020).

[28] Eclipse Foundation, "Desktop ides," 2020, https://www.eclipse.org/ide/ (Accessed August, 2020).

[29] The JUnit Team, "JUnit 4," 2020, https://junit.org/junit4/ (Accessed August 2020).

[30] The TestFX Team, "TestFX," 2020, https://github.com/TestFX/TestFX (Accessed August 2020).

[31] EasyMock Development Team, "Easymock," 2020, https://easymock.org/ (Accessed August, 2020).

[32] PowerMock Development Team, "Powermock," 2016, https://powermock.github.io/ (Accessed August, 2020).

[33] JetBrains s.r.o., "Intellij idea - code coverage," 2020, https://www.jetbrains.com/help/idea/code-coverage.html (Accessed August, 2020).

[34] M. R. Hoffmann, B. Janiczak, and E. Mandrikov, "EclEmma," 2018, http://www.eclemma.org/ (Accessed August, 2020).

[35] Unity Technologies, “Unity for Games,” 2020, https://unity.com/solutions/game (Accessed August 2020).

[36] Microsoft, “Visual Studio,” 2020, https://visualstudio.microsoft.com/ (Accessed August 2020).

[37] .NET Foundation, “NUnit,” 2019, https://nunit.org/ (Accessed August 2020).

[38] Gnoso, “NCover,” 2020, https://www.ncover.com/ (Accessed August 2020).

[39] NSubstitute Development Team, “NSubstitute,” 2020, https://nsubstitute.github.io/ (Accessed August 2020).

[40] JEST Development Team, “JEST,” 2020, https://jestjs.io/ (Accessed August 2020).

[41] Enzyme Development Foundation, “Enzyme,” 2020, https://enzymejs.github.io/enzyme/ (Accessed August, 2020).

[42] The Apache Software Foundation, “Apache NetBeans,” 2020, https://netbeans.org/ (Accessed August 2020).

[43] Scene Builder Development Team, “Scene builder,” 2016, https://gluonhq.com/products/scene-builder/ (Accessed August, 2020).

[44] SQLite Consortium, “Sqlite,” 2016, https://www.sqlite.org/index.html (Accessed August, 2020).

[45] Travis CI, “Travis CI,” 2020, https://travis-ci.org/ (Accessed August 2020).

[46] Szczepan Faber and friends., “Mockito,” 2020, https://site.mockito.org/ (Accessed August 2020).

[47] SmartBear Software, “Cucumber,” 2020, https://cucumber.io/ (Accessed August, 2020).

[48] AssertJ Development Team, “Assertj - fluent assertions java library,” 2020, https://assertj.github.io/doc/ (Accessed August, 2020).

[49] Codecov Development Team, “Codecov,” 2020, https://codecov.io/ (Accessed August, 2020).