10+ Years of Teaching Software Engineering with iTrust: the Good, the Bad, and the Ugly

Sarah Heckman
North Carolina State University
sarah_heckman@ncsu.edu

Kathryn T. Stolee
North Carolina State University
ktstolee@ncsu.edu

Christopher Parnin
North Carolina State University
cjparnin@ncsu.edu

Abstract

This paper presents an experience report with a junior-level software engineering course at North Carolina State University. We provide an overview of the course structure and the course project, iTrust, that has been developed by students over 25 semesters. We summarize reflections from faculty, teaching assistants, and students (through course evaluations). From our lessons learned, we present our course improvements as we prepare for the next ten years of software engineering courses. Our main lessons learned are 1) course technologies have a lifespan and require periodic updating to balance student learning and working with a legacy system; 2) teaching assistant longevity and support is critical to course success; and 3) the value of working with a large, legacy system in a semester long course is supported by faculty, teaching assistants, and eventually students.

CCS Concepts • **Applied computing → Education**;

Keywords software engineering education, iTrust

ACM Reference format:
Sarah Heckman, Kathryn T. Stolee, and Christopher Parnin. 2018. 10+ Years of Teaching Software Engineering with iTrust: the Good, the Bad, and the Ugly. In *Proceedings of 40th International Conference on Software Engineering: Software Engineering Education and Training Track, Gothenburg, Sweden, May 27-June 3, 2018 (ICSE-SEET'18),* 4 pages.
https://doi.org/10.1145/3183377.3183393

1 Introduction

As computer science educators, we try to transform our students from novices to professionals ready to participate in teams working on existing code bases. The learning theory of *legitimate peripheral participation* supports realism in the classroom by treating students as novice members of the profession [4]. Unfortunately, as students progress through most of the computer science curriculum, they rarely encounter settings that emulate "real world" projects, despite best efforts by educators incorporating realistic elements.

Toward this goal of realism, faculty have evolved a junior-level software engineering course for computer science majors that places working on a legacy software system, called *iTrust*, at the center of the course design. iTrust[1] is an electronic health records (EHR) system that has been developed over 25 semesters by student teams. At its largest point, iTrust contained over 50,000 lines of source code and 1,000 test cases which took over 45 minutes to run.

This course experience report draws on the experiences of six faculty members who have taught the course over the past decade, several current and former teaching assistants (TAs), and student evaluations. We briefly describe the undergraduate software engineering course and the iTrust project that has sustained the course for 10+ years. We present what worked (the good), what did not work (the bad), and what we are doing to revitalize the course for the next decade (in response to the ugly). Upon reflection, we have learned:

- The technologies chosen have a lifespan. Periodically rewriting the whole system may be less work for the teaching staff than doing smaller upgrades annually.
- Hiring TAs who have experience with the semester project is critical to student and faculty success.
- While students struggle to work with legacy code, faculty, TAs, and alumni see value in a team project that more closely models industry.

2 Software Engineering Course Structure

Software Engineering (SE) is a required course for computer science majors. The course is taught every semester and regularly has over 100 students. Typically taken their junior year, software engineering is the last core course before their capstone course, in which students work with industry partners on projects. The SE course learning outcomes cover software testing; software inspections; software design and evaluation of design; application of design patterns; writing software requirements; software development lifecycle; team-work; project management and risk management; and software maintenance. Using the knowledge from their earlier coursework, students are expected to pick up new technologies without formal instructional support.

[1] https://github.com/ncsu-csc326/iTrust

ICSE-SEET'18, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5660-2/18/05…$15.00
https://doi.org/10.1145/3183377.3183393

Prior to Fall 2017, the SE course is a three credit course with two 50-minute lectures and a 110-minute open-lab[2] offered over a 15-week semester. Lectures cover software engineering topics that students apply in the lab.

The first half of the course focuses on software engineering lifecycle phases (testing, design, and requirements) and the second half covers process and cross-cutting topics. Students complete three, multi-part homework assignments and a six-week long project. Most homework assignments and the project are completed in teams to allow students to put their education into practice [1, 3].

Lecture content varies by instructor, but involves active learning activities like think-pair-share or workshops. The labs involve activities that support collaboration. All major assignments in SE are associated with iTrust (Section 3).

2.1 Labs

Students participate in a weekly lab run by the TAs. Lab time is dedicated to group work, clarifying questions on assignments and the project, and code/design inspections.

2.2 Homework Assignments

The homework assignments introduce students to each other, the course technologies, and iTrust in a collaborative environment. Pairs or groups of three for the homework assignments are constrained to students within the same lab to allow for collaboration time during labs.

2.3 Team Project

Students complete a six-week team project during the second half of the semester in groups of four to five. The TAs take on the role of team managers for teams in their lab section(s). Within each team, each student takes on at least one leadership role from the following: Team Leader, Planning Lead, Development Lead, and Quality Assurance Lead. This provides exposure for students to the various roles a software engineer can hold in practice [5]. There is an expectation that everyone contribute to the team's development efforts. We assess this by looking at the project version control history.

Scope of the work: The project starts with the instructor providing a list of enhancements and maintenance requests to iTrust; some requests are required, others are optional. The scope of a typical enhancement is to add a new form or UI element and connect it with the existing infrastructure. Students must complete some of the optional pieces to receive full credit for the assignment, allowing some choice in what the students implement.

Development Process: Each student team has a GitHub repository to store their source code and wiki.[3] Using an Agile methodology for development, the work is organized into iterations, each lasting one week. This includes a requirements and planning iteration (iteration 0) and four development iterations (iterations 1 – 4). For the planning iteration students complete a *Software Project Management Plan (SPMP)*. In the development iterations, students complete enhancement and maintenance tasks. Students report their weekly progress to their lab TA and in a report on the team wiki. Additionally, the team provides a plan for the next week. Each task is expected to have an owner and the work to be distributed fairly to all team members.

The students complete development on one or more development branches, and completed code is merged with master. A completed use case requires that teams present a sequence diagram for one scenario through the use case, a class diagram showing the design for the use case, passing automated system test cases, passing unit test cases with greater than 80% coverage on back-end code, passing acceptance tests demoed to the TA, and a merge to master with a clean build on the continuous integration system. For deliverables related to non-functional requirements or constraints, the students must demonstrate completion as defined in the requirements. Each iteration is evaluated to show that students are making steady progress toward delivery [2].

The end of Iteration 4 is a 15-minute team demo on completed functionality to the lab. All team members are expected to present and teams are evaluated using a common rubric by their peers, the TA(s), and sometimes the instructor.

Transfer At the end of each semester, the current instructor works with the TAs to identify which project should become the starting project for the next semester. A member of the teaching staff removes identifying details about the student team and any failing tests cases are fixed. Refactoring, technology updates, and performance enhancements are completed as time allows.

3 iTrust

iTrust is an electronic health records (EHR) system created as a course project for SE. An EHR system was selected to 1) support security and privacy requirements; 2) be a large system that can be developed over many semesters, and 3) promote inclusiveness and diversity by providing a project domain that all students can relate to. iTrust was introduced in Fall 2004 and has been used as the course project every semester since, except Spring 2005. The basic functionality includes role based authentication (e.g., health care provider (HCP), patient, administrator), logging, and office visits (e.g., scheduling, recording patient information). The system consists of a MySQL and Java backend with a JSP frontend. The backend is tested with JUnit. Automated system level testing started with HttpUnit and has since moved to Selenium and Cucumber. We moved from an Ant to Maven build in Spring 2016.

[2] In open labs, students will not finish the lab work during the lab period.
[3] This describes the most recent set of software development tools. Various version control and collaboration support tools have been used in the past based on availability and support.

iTrust was restructured in Summer of 2016 due to a degrading architecture and aging technology stack. As part of the restructuring, a significant portion of functionality was pulled out of the system and the office visit functionality was siloed from the remaining system. A basic office visit was created using an updated technology stack that included Java Server Faces in the frontend.

Continuous Integration: To support automated grading and software engineering best practices, continuous integration was introduced in Spring 2011 and fully incorporated into the class in Fall 2014. Student teams use GitHub for version control and the continuous integration system, Jenkins, builds and tests their application using the master and development branches. Students get feedback about their work on a common system and TAs use the results for grading.

Enhancements: Each semester, student teams implement new functionality in iTrust that involves modifications to the database and frontend. For example, an enhancement in the Spring 2016 semester was to add a physical therapy module to iTrust that would allow the HCP to upload images of patient scans, schedule office visits for patients, and assign exercises. In Spring 2017, students added an obstetric and gynecology module that permits HCPs to schedule office visits and hospital visits for check-ups and delivery, respectively. Optional pieces typically include performance enhancements and other maintenance tasks.

4 iTrust: The Good, The Bad, and The Ugly

Six faculty members, including the three authors, and dozens of undergraduate and graduate TAs have taught the SE class and its labs since the introduction of the iTrust course project. The main structure of the course has remained relatively consistent between semesters with some variations and innovations over the years; however, with so many people evolving the iTrust system and the SE course over the past 10 years, many shared challenges have emerged for students, instructors, and TAs. In this section, we draw on the experiences of each of these groups to distill the lessons learned and reflect on the goal of using iTrust in support of realism.

Student reflections are from course evaluations for the question, *Comment on strengths and weaknesses of the course.*

4.1 The Good

Reflections in this category describe things that are going as intended, with regards to iTrust and the SE course.

Delayed appreciation for working with legacy systems. By the end of the semester, many students reach a level of comfort with iTrust and its technologies. While working with iTrust is frequently complained about in student evaluations, over the years, instructors have received a steady stream of thanks from alumni who later realize the value of the experience. Further, experience with iTrust is frequently mentioned as a primary reason by local corporations for recruiting students from the university.

Teamwork is critical for success in software engineering. The project is sized for completion by a team. While students may have had team experiences in earlier coursework, they may not have training to efficiently work on a team. In the SE course, team training is provided to help students adjust to working on projects with a larger scope. Teamwork with non-traditional or working students adds complications; faculty cannot assume that students have time to perform collocated work outside of class and lab times. This adds a focus on communication and effective use of tools, such as pull requests and wikis, for teams that cannot meet as frequently.

4.2 The Bad

Reflections in this category can often be resolved with small tweaks to the course structure or feedback cycles.

Grading is overwhelming for the TAs, sometimes leading to delayed feedback to students. Grading in the SE course has gone through significant changes over the last 10+ years. Before 2011, grading was manual and required a significant time investment. By 2011 iTrust and the test suite had grown to the point where running the unit and system tests took 30-45 minutes per project. Given the large class sizes, providing timely feedback for students was difficult. In 2011, a TA experimented with using a continuous integration system to offload the testing time for both the students and TAs. The use of continuous integration for automating student feedback and automating grading was fully adopted in Fall 2014. Additional benefits are that students are exposed to a tool they would likely see in industry, lending to the realism of the project. TAs have created other tooling to support common tasks like generating feedback files. Giving TAs independence to innovate benefits the course for the current and following semesters.

Large class sizes and group work completed in lab means the instructor is not always immediately privy to issues with group dynamics. As TAs work with the students in labs, they observe student difficulties more directly than instructors. Students who lack sufficient prerequisite knowledge have difficulty succeeding in the SE course. TAs also run into difficulty evaluating individuals during team projects. Instructors need to develop better rubrics to give TAs the opportunity to penalize non-performing team members.

4.3 The Ugly

Reflections in this category require deeper changes to the course for full resolution.

The workload is too high for a three credit hour course. The most common complaint is that students feel that the amount of work that they complete in the course is more

than should be associated with a three credit hour course. Faculty try to identify appropriately sized assignments for completion in a team, but there are often unforeseen difficulties (e.g., technology compatibility issues with personal machines) that may lead to additional time for completion.

Working with legacy technologies is overwhelming. Because iTrust is a complex application, students are overwhelmed by the complexity of working on an existing code base. The learning curve is steep. Additionally, students are more likely to be subjected to poor design choices due to the longevity of iTrust and the density of student code therein.

The schedule is aggressive. The first half of the course involves weekly programming assignments. The team project also involves weekly iterations that require programming. The teaching staff, TAs and faculty, find that supporting the SE course workload is overwhelming. This is especially true if the TAs do not have experience with the course, technologies (especially Jenkins), or professional experience.

A further issue is the students view disconnects between homework assignments and the project, and between the projects and lecture. The intention is for the homework assignments to introduce students to the technologies and teamwork, but this can gets lost in the aggressive deadlines.

5 iTrust2 - The Next 10 Years

After working with the same software for 10+ years, we are addressing *The Ugly* in the SE course and iTrust.

The workload is too high. Starting in Fall 2017, the SE course moved to a four credit hour course with two 75-minute lectures and one 110-minute lab each week. The extended time in lecture improved team communication by providing additional time for group work and scrum meetings during the team project portion of the course.

Working with legacy technologies is overwhelming. A major change is the creation of iTrust2[4]. The design of iTrust had significantly degraded and the technologies had aged to a point where students were fighting the technology and not practicing good software engineering. To keep the educational experience current [5], for Fall 2017 we rewrote iTrust with an updated technology stack: MySQL, Hibernate, and Java backend tested with JUnit; a REST API; and an Angular frontend tested with Selenium and Cucumber. These technologies more closely match those seen in the senior capstone course and used by industrial partners who hire many of our graduates. We continue using GitHub and Jenkins to support collaboration, rapid feedback, and grading.

The schedule is too aggressive. The assignment structure changed to provide more time for students to learn technologies and processes. Students start the course with a two and a half week on-boarding activity consisting of team development activities in lecture and lab that introduce students to course technology with maximal instructional support. Next, students work through a four-week guided project implementing a use case in iTrust2. Finally, students complete a six-week team project. Team training, code review, and process are more emphasized. The faculty have developed new rubrics that support evaluating process in addition to product and recognition of individual contributions to the assignments.

[4]https://github.com/ncsu-csc326/iTrust2

6 Conclusion

Instructional staff and students ultimately value the exposure to a large existing system. However, systems like iTrust require routine maintenance between semesters to minimize pain points for the teaching staff. The best practice is to have several faculty or staff involved with a continuing TA to maintain iTrust. Additionally, legacy systems become a liability to student learning when the design degrades and the technologies are outdated. Identifying the appropriate point to start over to maintain student learning outcomes is critical. Restarting every five to seven years may be more appropriate than waiting over 10 years.

As instructors, we feel that the SE course moves students into the software engineering profession, preparing them for the senior capstone and following professional experiences. While during the semester, students may find the course time consuming and difficult, they ultimately value working in teams with new technologies using professional tools and best practices on a large, complex software system.

Acknowledgments

The authors thank Emerson Murphy-Hill, Laurie Williams, Jason King, Meiyappan Nagappan, Kai Presler-Marshall, Andy Meneely, Lauren Schaefer, Ben Smith, and Michael Winters. iTrust2 would not be possible without the work of Kai Presler-Marshall and Elizabeth Gilbert. We especially want to thank our students who learned, survived, and (in most cases) thrived during the SE course experience. This work is supported in part by an NCSU DELTA grant to the authors and NSF SHF-1645136.

References

[1] Jürgen Börstler and Thomas B. Hilburn. 2016. Team Projects in Computing Education. *Trans. Comput. Educ.* 16, 2, Article 4 (March 2016), 4 pages. https://doi.org/10.1145/2808192

[2] Heidi JC Ellis. 2008. *Software Engineering: Effective Teaching and Learning Approaches and Practices: Effective Teaching and Learning Approaches and Practices.* IGI Global.

[3] Howard Hills. 2001. *Team-based learning.* Gower Publishing, Ltd.

[4] J. Lave and E. Wenger. 1991. *Situated Learning: Legitimate Peripheral Participation.* Cambridge University Press.

[5] Mary Shaw. 2000. Software Engineering Education: A Roadmap. In *Proceedings of the Conference on The Future of Software Engineering (ICSE '00).* ACM, New York, NY, USA, 371–380. https://doi.org/10.1145/336512.336592