2023 Annual Conference & Exposition
Baltimore Convention Center, MD | June 25 - 28, 2023
The Harbor of Engineering Education for 130 Years
ASEE
Paper ID #39422

Work in Progress: An Interview-Based Retrospective on the Redesign of an Introductory Computing Course

Stephanos Matsumoto, Olin College of Engineering

Stephanos (Steve) Matsumoto is an Assistant Professor of Computer Science and Engineering at the Olin College of Engineering. His research interests are in computing education, particularly in how to incorporate better software engineering practices when teaching computing in undergraduate STEM courses.

Work-in-Progress: An Interview-Based Retrospective on the Redesign of an Introductory Computing Course

Stephanos Matsumoto
smatsumoto@olin.edu
Olin College of Engineering

Abstract
In this paper, we conduct a qualitative study to describe how focusing more on software engineering skills, code quality, and reflection on programming practices in an introductory computing course has led to improvements in students' experience and learning outcomes. Our work took place during the summer and fall of 2020 at Olin College of Engineering, a small, undergraduate-only engineering college in Massachusetts. We describe how, motivated by difficulties in developing and assessing code quality in students work, we significantly redesigned course materials, assessment rubrics, and tooling. While we and our colleagues see informal evidence that the overall quality of student work and coding habits have improved, we conducted semi-structured interviews with current students and alumni, representing the perspectives of those who took the old version of the course and those who took the redesigned version. We present early results from these interviews, identifying themes that represent student perceptions on how computing courses are useful in their education and in their careers. These themes will help us to develop further metrics that will allow us to assess the usefulness of the course redesign in a more detailed way. In turn, these metrics will help us analyze future redesigns, both to this course and to other courses in our computing curriculum.

Introduction

Much work in computing education research has studied introductory programming or computer science courses in undergraduate education, with the literature being extensive enough that reviews consider hundreds or even thousands of published papers [1, 2]. Though programming is taught in many STEM disciplines, these introductory courses are often taught within computer science and thus are typically referred to as CS1 [3]. Topics covered in a typical CS 1 course include types, control flow, basic data structures (e.g., arrays), foundational problems and algorithms (e.g., sorting), and simple recursion [4].

CS1 courses are often seen as the entry point to computing-related fields, and further courses in all areas of STEM build on foundational programming knowledge taught in CS1. Accordingly, previous research has largely focused on assessing CS1 courses using relatively traditional approaches such as concept inventories [5], retention rates [6], or student perceptions of the course [7]. In addition to quantitative evaluations, there have been efforts to qualitatively evaluate student experiences in CS1-style courses, such as their self-efficacy [8] or struggles [9].

Unfortunately, there is considerably less research on the development of knowledge and skills relevant to professional programmers, such as code quality, and the existing research seems to indicate that these skills are not priorities in CS1 courses [10, 11]. A significant fraction of a software project's budget can go towards code quality [12], and the need for better software engineering practices is becoming important in STEM at large [13]. As introductory computing courses serve an increasingly broad range of students, disciplines, and applications, and these traditional metrics may not provide sufficient insight into what is best for students' learning.

Teaching at a small engineering college with no formal computer science program and a heavily project-based curriculum, we are especially aware of the limitations of traditional metrics often used to evaluate first courses in programming and computer science. As we explain later in this paper, our first course in computing, which we dub Software Design (SoftDes), is not a traditional one: many topics typical of a CS1 course are left out to make room for project work and skills likely to be used in practice, such as version control and data science tools. However, retention rates and student perceptions of SoftDes say little about how students view the course in the context of their overalll education, or how these skills are useful in their future studies, internships, or jobs. Despite this, SoftDes is designed to better equip students with computing skills, whatever they may end up doing, and traditional metrics do not capture the success of the course in that goal.

In the summer of 2020, we undertook a major redesign of SoftDes in preparation for the Fall 2020 semester. Our changes were motivated by major hurdles we had encountered in teaching the course: (1) limited opportunity to build (rather than practice) technical skills, (2) lackluster code quality in student submissions, and (3) difficulties in providing thoughtful feedback and assessment to students. Among other changes, we added much more detailed assessment rubrics and adopted modern tools such as autoformatters, linters, and unit testing frameworks. In order to serve students interested in applying their software design skills to an ever-increasing range of disciplines, we also added problems in topics such as voting, game design, and scientific communication, as well as modules on testing, debugging, refactoring, and documentation. Our anecdotal experience is that the quality of our students' technical knowledge and code have increased significantly.

Ultimately, our goal is to build a *software engineering mindset* [14] in students: the idea that software engineers need to be able to build software that may outlive their individual contributions, be worked on and maintained by others, and used by potentially massive numbers of users. As we iterate towards making SoftDes a first step towards this goal, we acknowledge that despite improvements in assessing students technical knowledge, we do not have corresponding metrics to measure the usefulness of this course in the context of how they might use this knowledge in the future.

As a first step in addressing this problem and developing metrics that will help us better evaluate the contributions of nontraditional, innovative computing courses such as SoftDes towards a variety of learning and career goals, we describe a study design and preliminary results in qualitatively assessing the changes that we have made in SoftDes. Specifically, we are conducting semi-structured interviews with previous students of the course, some of whom are current students and some of whom are alumni. Our goal is to evaluate how the course has served the learning and career goals of students in ways not captured by traditional metrics for introductory

computing courses. Additionally, we aim to survey the perspectives of students who took the old version of SoftDes as well as those who took the redesigned version to assess the effects of the changes we made to the course. In doing so, we hope to gain valuable perspectives that will help us develop better metrics for courses and programs like ours.

In the remainder of this paper, we describe the institutional context in which we carried out our work, as well as the details of the changes we made. We then describe our methodology and preliminary results. We end with a brief discussion of related work and next steps.

Redesigning an Introductory Computing Course

In this section, we describe the institutional context of Olin College of Engineering, at which this redesign was done. We then describe details of the improvements we made to the course, along with informal responses to this redesign.

Institutional Context

Our redesign work took place at Olin College of Engineering (hereafter Olin College), a small engineering college in Needham, Massachusetts. The college has around 350 students, and only offers engineering majors (Engineering, Mechanical Engineering, and Electrical and Computer Engineering). Students in the Engineering major choose a concentration, such as Design, Computing, or Sustainability. Our introductory course SoftDes is open to all students of Olin College, but is only required for those majoring or concentrating in a computing-related field, such as Electrical and Computer Engineering or Robotics. Most students who take SoftDes are required to do so for their major or concentration, but the course also routinely has non-majors, such as Mechanical Engineering majors, as well as non-STEM majors from neighboring colleges. The course is taught almost entirely in Python, though students also spend a bit of time learning basic UNIX command-line tools and Git. Typical annual enrollment in the course is between 70 and 100 students. There are no examinations in the course, but students demonstrate their learning through two major projects done in small teams (typically 2–3 students).

The Need for Redesign

Even before the redesign, SoftDes was not a traditional CS1 course. The course covered concepts traditionally considered important among CS1 instructors, such as control flow, methods, and data types, but also left out some of these concepts, such as graphs and trees [4]. The course also spent comparatively less time on topics such as recursion, sorting, and arrays. Students learned a majority of their core programming concepts (such as control flow, functions, types, and classes) through readings and checked their understanding through Jupyter notebooks with short written or programming questions. Much of the coding in the course was done through projects of varying scope, in which students could apply their programming knowledge to problems in various fields. Examples of these projects include (1) implementing a suite of functions that used string, list, and dictionary operations to find likely protein-coding sequences within an organism's DNA, (2) writing recursive functions to generate images of random art [15], and (3) designing and implementing an object-oriented interactive program, such as a game, Web service, or command-line tool. The last of these was typically done as the final project in the course.

The project-based nature of the course and the emphasis on real-world applications of computing helped students see how computing could benefit practitioners of a range of disciplines that

extended far beyond the traditional realm of computing. At the same time, we identified several clear areas for improvement:

- Much of students' time in the course was spent on projects, which allowed students to practice programming in a real-world context but only provided limited practice of the technical material.
- As is relatively common in code submitted in introductory courses [16], code quality was generally lackluster, with non-idiomatic code and inconsistent formatting making submissions hard to read and understand.
- We lacked the assessment infrastructure necessary to provide detailed feedback to students beyond correctness, making it difficult to help students improve their skills beyond the readings and projects.

Redesigning the Course

To address these areas of improvement, we spent the summer of 2020 undertaking a major redesign of the course. In particular:

- We created more traditional, scaffolded worksheets and assignments designed to help students practice core programming and software engineering skills. Worksheets are done in class and consist of short programming or discussion problems. Assignments contain larger or more difficult problems, some which are adapted from the smaller projects previously done in the course.
- To model and encourage good practices such as testing, documentation, and idiomatic, well-styled code, we explicitly cover these concepts during the semester, both in readings and assignments, and demonstrated these practices in sample solutions given to students.
- We created detailed assessment rubrics to outline our expectations of correctness, style, and documentation to students. We also added a self-assessment component to assignments, in which students would reflect on their code, the output from running unit tests and style checks on their code, and the sample solutions we provided. These changes aimed to follow principles of good feedback, allowing students engage in a dialogue with the teaching staff about their learning [17].

Because of the way that we integrated these changes throughout the course, students were able to engage with topics such as testing and documentation for a much greater portion of the course than before. Since the course requires no previous programming experience, we presented these topics in a scaffolded way to help students practice the concepts before diving into the code. Table 1 gives an example of such scaffolding for a topic. Other topics that we focus more on in the course redesign include debugging and error handling, refactoring, using and managing external libraries, scientific communication (particularly in the realm of data science), object-oriented design, and the model-view-controller architecture.

In this redesign, we focused on adding topics in software engineering that would primarily help students in their programming workflows, with a particular focus on code quality. As such, there were software engineering topics that we could only cover in a limited capacity or as part of other topics. For example, requirements elicitation was somewhat subsumed into unit testing, with

Table 1: A selected example of a new course topic (testing and documentation) presented in a scaffolded way throughout the course. This topic is presented over two to three weeks.

Experience	Activity
Reading 1	Learn about basic data types (integers, floating-point numbers, booleans, and strings) and what operations can be done with them.
Worksheet 1	Experiment with functions and operators with different data types to get hands-on knowledge of parameter and return types (e.g., int division produces a float, the `print` function returns nothing).
Assignment 1	Write docstrings demonstrating understanding of parameter and return types as well as the course docstring format.
Reading 2	Learn what unit testing is, what it does and does not guarantee, and principles for designing unit tests (e.g., test at input boundaries, test exercise different code paths).
Worksheet 2	Write several unit tests in Pytest and discuss how to handle cases without a clear answer (e.g., what to do when finding the minimum of an empty list).
Assignment 2	Write a suite of unit tests and run them against a set of implementations without seeing the source code to determine which implementations are likely correct.

students asking more detailed requirements of specific functions or classes in the process of coming up with test cases for the function/class in question. While we hope to make this a core part of our course in the future, this work is still in progress.

In addition the above changes, we also changed the in-class experience. While class time before the redesign was largely spent tackling projects or conducting design/code reviews, we reduced much of this "open work time" in favor of synchronous, deliberate practice through worksheets. During class meetings, students alternated between working on exercises in the worksheet and discussing solutions. Anecdotally, this change spurred a lively dialogue among students and the teaching staff, and provided a natural opportunity to discuss problem-solving approaches, code quality, and the thought process of experienced programmers. The addition of worksheets thus helped students, providing not only more opportunities to practice technical programming concepts, but also more opportunities for the teaching team to practice good feedback approaches [18].

We also made more minor improvements to the course, including changing the way that course assistants (CAs) accessed and graded assignments, making the computational setup process less onerous for students, and choosing tools to use and teach in the course (e.g., IDEs and autoformatters) that better suited our needs.

Reactions to the Redesign

Though we more formally assess our redesign in the following section, we received much informal feedback on our changes throughout the process. Generally, students perceived the course as being more difficult and having a heavier workload, largely due to the effort of ensuring code quality in their submissions. Despite this, there was a marked increase in students' technical skills and code quality that has been visible throughout the institution. A faculty member in a

Table 2: The survey question designed to elicit how SoftDes has been useful in participants' learning and career goals and select interview participants.

Have you used knowledge, skills, or abilities you learned in SoftDes for any of the following? Check all that apply.
• Other courses at this institution in computing • Other courses at this institution not in computing • Bachelor's-level courses not at this institution in computing • Bachelor's-level courses not at this institution not in computing • Master's degree programs in computing • Master's degree programs not in computing • Professional degree programs (for example: MD, DDS, DVM, LLB, JD) • Doctorate degree programs (for example; PhD, EdD) in computing • Doctorate degree programs not in computing • Internships in computing • Internships not in computing • Full-time employment in computing • Full-time employment not in computing

computing-adjacent field (e.g., computational physics) noted that it was clear which students had taken SoftDes, because the quality of their code was better than that of their peers' code. A student who did an industry internship after taking SoftDes noted that their manager saw the style and level of detail in their documentation and got the rest of their team to adopt the same approach to documentation. Overall, despite some mixed responses over the increased workload, community reaction to these changes seem to be positive.

Study Design and Preliminary Results

In order to more formally gauge perspectives on SoftDes and our redesign of the course, we designed a qualitative study based largely around semi-structured interviews with past students of SoftDes. The students are a mix of current students (upperclassmen) and alumni.

We recruited participants through an email sent to mailing lists for current students and for alumni. The email invited them to fill out a survey to indicate their interest in participating in an interview. The survey asked students for their graduation year and major, as well as the ways in which they have used knowledge, skills, or abilities learned in SoftDes (full question in Table 2). We use the responses to interview a broad range of participants, capturing the ways in which SoftDes has or has not served various majors, applications, and career trajectories.

The survey is still active, but we have begun the process of selecting and interviewing participants. We are selecting interview participants to be balanced across graduation year, major, and their responses on how they have used SoftDes since taking the course. In total, we have 11 initial participants signed up, and are continuing to recruit additional participants. We chose to carry out interviews over Zoom (which all current students have access to through the college) to

minimize the cost of participation, and paid participants USD 15 (the average hourly wage of a student worker at our institution) to spend 30–60 minutes interviewing with us.

While we are still in the process of carrying out our interviews, here are some preliminary themes that have emerged in our initial conversations:

- Particularly before the redesign, SoftDes heavily utilized "open class" time, in which students could work on worksheets, take-home assignments, or projects, and get help and feedback on any of those tasks. Students found this time helpful in building collaboration skills, but not necessarily in building technical skills, with some students feeling that the tasks were more about getting code to work than understanding the underlying concepts.
- Our institution faces systemic issues that make running a successful SoftDes challenging. In particular, the students who enroll in the class, like in many CS1 courses, can bring a wide range of previous experience. Because of our small size and relatively small computing curriculum, students with a great deal of experience can still end up in SoftDes. Here, they learn a great deal about code quality but are generally bored when covering programming topics.
- Attitudes towards the emphasis on code quality are indeed mixed. Readability is generally easily solved with the use of an autoformatter, but many do not practice good code quality as rigorously outside of the class. Nevertheless, one student felt that modeling what good code should look like was helpful.
- Students at our institution see a introductory computing course primarily as a place to build foundational skills for software engineering. These skills include familiarity with common data types such as integers, strings, and lists, critical thinking and problem-solving skills, and how to find helpful information in documentation and forums when stuck or using a new library.
- Those who have done internships often find themselves in the position where they are dropped into a massive codebase and need to understand the components of it that they will be working with. They note that our program (like many) does not make sufficient space to practice this skill common in industry jobs.

Discussion

In this section, we discuss work related to our work on SoftDes, and briefly describe our next steps in this work.

Related Work

SoftDes teaches a variety of topics, some of which overlap with topics traditionally taught in CS1 courses. The topic list of SoftDes covers much of those described in literature reviews by Medeiros et al. [2] and Luxton-Reilly et al. [1], as well as that of Hertz's study on topics covered in CS1 vs CS2 [4]. We are by no means the first course to cover software engineering topics in an introductory course: Chen and Hall propose a software engineering project for CS1 whose process resembles that of our projects [19], and our coverage of software testing has some overlap with many courses reviewed by Scatalon et al. [20]. As we continue to iterate on the list of topics

covered in SoftDes, we rely on these works to characterize the space that our course occupies in the literature.

Souza et al. shows that project-based learning in the context of software engineering education can be valuable [21], which is somewhat supported by our preliminary interview results. Perretta and DeOrio show that providing feedback on student-written testing (particularly the presence of false positives) can improve the quality of student tests [22], and we have several small exercises to do this in SoftDes. Effenberger and Pelánek describe a variety of code quality defects that are common in student code in introductory courses and suggest targeted feedback to help curb these practices [16]. We found that many of these could be caught by linters such as Pylint, and we consider linter output in our assessment rubrics. These works provide support that our practices in how we teach software engineering in the context of SoftDes are effective, and provide us with ideas for further improvement.

Next Steps

We are continuing to interview students. Perhaps unsurprisingly, the vast majority of our survey respondents are majors in a computing-related field, and most of them have used skills from SoftDes in further courses at our institution and in industry rather than in graduate studies or academia. We hope to recruit and interview more students in these areas to get a more diverse range of perspectives on the course.

Once we have a complete dataset and have identified themes in participant responses, we plan to use the results of our analysis to propose and validate nontraditional metrics for introductory computing courses.

Conclusions

In this paper, we described our work in redesigning an introductory computing course, SoftDes, at a small engineering college. In response to existing difficulties with running the course, we more heavily emphasized technical practice, improved our assessment infrastructure and development tooling, and focused strongly on good code quality. Overall, responses to these changes seem to have been positive, and interviews indicate that while some room for improvement remains, the knowledge, skills, and abilities learned in SoftDes have been useful. There is still work to be done in gaining further perspectives from students and identifying themes that will help us develop better metrics for assessing nontradtional introductory computing courses. Nevertheless, we are encouraged by our progress so far and hope that our work will help us iterate towards a SoftDes that helps build a software engineering mindset in an ever-increasing number of its students.

References

[1] A. Luxton-Reilly, Simon, I. Albluwi, B. A. Becker, M. Giannakos, A. N. Kumar, L. Ott, J. Paterson, M. J. Scott, J. Sheard, and C. Szabo, "Introductory programming: a systematic literature review," in *ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, Jul. 2018, pp. 55–106.

[2] R. P. Medeiros, G. L. Ramalho, and T. P. Falcao, "A systematic literature review on teaching and learning introductory programming in higher education," *IEEE Transactions on Education*, vol. 62, no. 2, pp. 77–90, May 2019.

[3] K. Quille and S. Bergin, "CS1: How will they do? How can we help? A decade of research and practice," *Computer Science Education*, vol. 29, no. 2-3, pp. 254–282, May 2019.

[4] M. Hertz, "What do "CS1" and "CS2" mean? investigating differences in the early courses," in *ACM Technical Symposium on Computer Science Education (SIGCSE)*, Mar. 2010.

[5] M. C. Parker, M. Guzdial, and S. Engleman, "Replication, validation, and use of a language independent CS1 knowledge assessment," in *ACM Conference on International Computing Education Research (ICER)*, Aug. 2016, pp. 93–101.

[6] M. S. Kirkpatrick and C. Mayfield, "Evaluating an alternative CS1 for students with prior programming experience," in *ACM Technical Symposium on Computer Science Education (SIGCSE)*, Mar. 2017, pp. 333–338.

[7] J. Campbell, D. Horton, M. Craig, and P. Gries, "Evaluating an inverted CS1," in *ACM Technical Symposium on Computer Science Education (SIGCSE)*, Mar. 2014, pp. 307–312.

[8] P. Kinnunen and B. Simon, "CS majors' self-efficacy perceptions in CS1: Results in light of social cognitive theory," in *International Workshop on Computing Education Research (ICER)*, Aug. 2011, pp. 19–26.

[9] J. L. Huff and H. R. Clements, "The hidden person within the frustrated student: An interpretative phenomenological analysis of a student's experience in a programming course," in *ASEE Annual Conference and Exposition*, Jun. 2017.

[10] D. Kirk, T. Crow, A. Luxton-Reilly, and E. Tempero, "On assuring learning about code quality," in *ACM Australasian Computing Education Conference (ACE)*, Feb. 2020.

[11] L. Östlund, N. Wicklund, and R. Glassey, "It's never too early to learn about code quality: A longitudinal study of code quality in first-year computer science students," in *ACM Technical Symposium on Computer Science Education (SIGCSE)*, Mar. 2023, pp. 792–798.

[12] Y. Hilburn and M. Towhidnejad, "Software quality across the curriculum," in *Frontiers in Education Conference (FIE)*. IEEE.

[13] J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, and G. Wilson, "How do scientists develop and use scientific software?" in *ICSE Workshop on Software Engineering for Computational Science and Engineering*. IEEE, May 2009.

[14] M. J. Lutz, J. R. Vallino, K. Martinez, and D. E. Krutz, "Instilling a software engineering mindset through freshman seminar," in *Frontiers in Education Conference (FIE)*, Oct. 2012.

[15] C. A. Stone, "Random art," 2009. [Online]. Available: http://nifty.stanford.edu/2009/stone-random-art/

[16] T. Effenberger and R. Pelánek, "Code quality defects across introductory programming topics," in *ACM Technical Symposium on Computer Science Education (SIGCSE)*, Feb. 2022, pp. 941–947.

[17] D. J. Nicol and D. Macfarlane-Dick, "Formative assessment and self-regulated learning: a model and seven principles of good feedback practice," *Studies in Higher Education*, vol. 31, no. 2, pp. 199–218, Apr. 2006.

[18] C. Ott, A. Robins, and K. Shephard, "Translating principles of effective feedback for students into the CS1 context," *ACM Transactions on Computing Education (TOCE)*, vol. 16, no. 1, pp. 1–27, Jan. 2016.

[19] W. K. Chen and B. R. Hall, "Applying software engineering in CS1," in *ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, Jul. 2013, pp. 297–302.

[20] L. P. Scatalon, J. C. Carver, R. E. Garcia, and E. F. Barbosa, "Software testing in introductory programming courses," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, feb 2019.

[21] M. Souza, R. Moreira, and E. Figueiredo, "Students perception on the use of project-based learning in software engineering education," in *Brazilian Symposium on Software Engineering (SBES)*. ACM, Sep. 2019.

[22] J. Perretta and A. DeOrio, "Teaching software testing with automated feedback," in *ASEE Annual Conference and Exposition*, Jul. 2018.