2021

Development & Evolution of a Computer Science I Course

Chris Bourke

Development & Evolution of a Computer Science I Course

Dr. Chris Bourke
Associate Professor of Practice
School of Computing
University of Nebraska–Lincoln

December 2021

CSCE 155E is the flagship Computer Science I course in the School of Computing at the University of Nebraska–Lincoln. A high-enrollment course with multiple sections in multiple delivery modes, this course serves as an introduction to problem solving using computers. This course is the first computing course for Computer Science and Computer Engineering majors and is a service course for a variety of other (mostly engineering) majors. The course was redesigned in fall 2018 with the goal of making it more scalable and improving rigor, student outcomes, and the learning experience. Though it has been a slow process, the course has evolved over the last four years to meet these goals and has shown recent success in improving student outcomes (in particular in reducing DFW rates).

This document provides a comprehensive review of the course through the University of Nebraska–Lincoln's Faculty-led Inquiry into Reflective and Scholarly Teaching (FIRST) program (formerly known as the Peer Review of Teaching Project). We detail the course's structure, resources, assessment, redesign, and evolution over the last four years and provide a substantial analysis of its improved outcomes.

Contents

1. Introduction

CSCE 155E - Computer Science I is the flagship Computer Science I course in the School of Computing[1] at the University of Nebraska–Lincoln. The School has offered a variety of introductory computer science courses over many years, this particular version was initially developed in summer 2018 in an attempt to unify our Computer Science and Computer Engineering majors into a single introductory course.

This version of the course has been offered every fall semester for the last 4 years (fall 2018 – fall 2021) and has had a single instructor (the author). The course is typically a high-enrollment course with consistently more than 200 students across several sections (in-person, online, Nebraska Now[2]).

The re-development of this course unified several smaller sections into one single large section with several goals:

- to make it more cost effective in terms of instructors and instructional support
- to provide consistency in instruction and delivery to both computer science and computer engineering majors
- to make the course more scaleable
- to preserve the rigor of the content
- to improve the student learning experience

Over the last four years, the course has evolved with major and minor changes, but there is good evidence that these goals were achieved. Nevertheless, continuous improvement is always a goal. Though it has always been an aim, more recently improving *aggregate* student outcomes has become a focus. The most recent offerings of this course have seen changes intended to achieve a (modest) improvement in this area and data suggests they have been effective.

In the subsequent sections of this document, we provide some additional background and detail on the course; its history, relation to the School of Computing's curriculum, and student make-up. We then present the course structure, its learning objectives, teaching methods and content delivery, and how students are assessed in their learning. We then present and analyze data on student outcomes. Finally, we outline continuing challenges and present planned changes for future offerings to address them.

[1] Formerly, Department of Computer Science & Engineering, reorganized into the School of Computing in 2021

[2] A UNL program that offers greatly discounted online courses to high-school students for dual credit

2. Background

Prior to the establishment of the School of Computing, the Computer Science major was housed in the College of Arts & Sciences and the Computer Engineering major in the College of Engineering. Students in these two majors were separated into two different courses during their first semester: CSCE 155A for Computer Science majors and CSCE 155E for Computer Engineering majors. The content, topics, and learning outcomes of the two courses were essentially the same, but used different programming languages (Java and C). In fact, there were (and are) several other Computer Science I variants offered by the School. The idea being that students could come to the discipline from a variety of backgrounds and interests with courses tailored to match those interests. For our majors, however, several disadvantages became aparent.

- With a variety of courses, languages, instructors, etc. there was an inconsistency in outcomes that affected performance in future courses.
- CS majors were ill-prepared for courses that required C (generally, C is a more difficult language)
- The non-uniform background required coverage of Java in Computer Science II which the CS students were already familiar with. As a result, the first few weeks in CS2 covered both Java for those without the background and PHP for those with prior Java experience which was extremely challenging.

By 2017, facing reduced resources (TA funds) and fewer faculty, I proposed a revamp of CSCE 155E to serve as the flagship CS1 course for both Computer Science and Computer Engineering majors. Both majors (as well as some non-majors) would matriculate through the same course. C was chosen as a first language to ensure that both majors had it for future courses (in addition, it provides a more rigorous CS1 experience and this course was still necessary as service course to other engineering majors, primarily Electrical Engineering).

The revamp was presented to the faculty in early 2018 and met with (mostly) approval and had the support of the department chair. Funds were made available to develop the course material and during the summer 2018 I developed dozens of instructional and tutorial-style videos along with all new material (labs and hacks). This past fall was the fourth offering of this course and its most successful yet.

2.1. Relation to School of Computing Curriculum

CSCE 155E is essential for both Computer Science and Computer Engineering majors, being the first required computing course in the curriculum. This course is also a prerequisite for Computer Science II (CSCE 156), the second course in our first year sequence

for both majors. It provides a foundation in computing and programming necessary to continue in our program.[3]

For most students, this course is their first exposure to computing, programming, and what is actually involved in software development. It provides a vital "first impression" to the discipline. Many students fall in love with computing after taking this course, others find that the discipline is not for them. Regardless of the outcome, it is essential that students have an equitable, honest, and positive experience in this course. It needs to be equitable in that, though not all students will succeed, every student has the same opportunity and resources to do so. It needs to be honest in that it provides an accurate picture of what the day-to-day discipline of computing is like and what their future careers will involve. Many students come in with misimpressions of computing[4] or bad motivations (easily obtainable jobs with high salaries or that they'll make the next Facebook). It needs to be positive in that a student should not discontinue computing due to a bad experience such as a single negative interaction or lack of self-efficacy. If a student does not continue in the discipline it should be because they genuinely felt it wasn't for them or they found something they love even more.

2.2. Student Enrollment

The course is designed and intended primarily for Computer Science and Computer Engineering majors. Generally, every first year freshman or transfer student starting in either major is enrolled in this course as their first course. No prior computing or programming knowledge is required or expected of someone taking this course, though many students do have some programming background either formally from their high school curriculum or as a hobbyist.

The majority of students (60%) taking this course are Computer Science and Computer Engineering majors, but there is a substantial population of non-majors. Specific enrollment numbers for the last four years can be found in Table 1.

Table 1: Enrollment in CSCE 155E by major

Semester	Computer Science	Computer Engineering	Other	Total
Fall 2018	102	36	76	214
Fall 2019	87	31	99	217
Fall 2020	102	34	81	217
Fall 2021	95	32	83	210

Enrollment as well as the proportion of majors has remained stable over the last four years despite a substantial overall enrollment decline at UNL and in higher education in

[3]For both majors, a grade of C or better is required to continue in the program.
[4]They like playing games and *using* a computer, so it must be the same thing!

general.

Non-majors make up a substantial amount of the enrollment. This course does act as a service course to several engineering majors but only Electrical Engineering majors are *required* to take this version and most of them take it in spring semesters. In any given year, non-majors represent 30 different majors from every college at UNL. It remains unclear why so many non-majors take this particular course when the School of Computing does not actively recruit or promote it generally. It is also unclear why so many non-majors take this course when the School offers a variety of other introductory courses intended for non-majors. Unfortunately, non-majors have poorer outcomes (with respect to DFW rates, see Section 7).

3. Course Structure

This course is a traditional "Computer Science I" course in that it introduces students to problem solving using computers including general problem solving methods and strategies, software development principles and tools, and students learn a high-level programming language (specifically C). Basic programming concepts are covered from basic I/O to searching and sorting algorithms. Software development principles include best practices (writing readable and "clean" code with good style, documentation, and design) as well as tooling (IDE and command line usage, git, debugging with GDB, static analysis tools such as valgrind). Students also experience teamwork and collaboration.

3.1. Learning Objectives & Student Outcomes

The course has several well-defined learning objectives that are presented in terms of *skills objectives*. Programming and problem solving are skills that are learned through practice and iteration rather than simply acquiring knowledge. These are shared with students through the course syllabus.[5]

Upon the successful completion of this course students be able to approach a reasonably complex problem, design a top-down solution, and code a program in a high-level programming language that automates solutions. To that end, students should be able to exhibit the following skills.

- Have a mastery of the fundamentals of programming in a high-level language, including data types and rudimentary data structures, control flow, repetition, selection, input/output, and procedures and functions.

[5] see https://github.com/cbourke/ComputerScienceI/blob/master/documents/syllabus.md or https://github.com/cbourke/ComputerScienceI for all course content

- They should have a familiarity with problem solving methods, including problem analysis, requirements and specifications, design, decomposition and step-wise refinement, and algorithm development (including recursion).

- They should have a familiarity with software development principles and practices, including data and operation abstraction, encapsulation, modularity, code and artifact reuse, prototyping, iterative development, best practices in coding design, style, and documentation, a good understanding of proper testing and debugging techniques, and a familiarity with development tools.

- They should have exposure to algorithms for searching, sorting and other problems, graphical user interfaces, event-driven programming, and database access.

- They should have a foundation for further software development and exploration. They should have a deep enough understanding of at least one high-level programming language that they should be able to learn another programming language with relative ease in a relatively short amount of time.

3.2. Course Organization

Each of the main learning objectives is broken down and covered to various degrees in a series of 15 *modules* roughly corresponding to the standard 15-week academic semester. Each module focuses on one major programming and problem solving concept (basic I/O, loops, encapsulation, etc.) but at the same time builds on the knowledge and skills developed in a prior module *and* gives a preview/glimpse of future modules. For example, the file I/O module (the lab and hack) requires the knowledge built in the strings module in order to manipulate and process string-based data from a file while at the same time introducing the problem of sorting data (a future module). In this sense, each module is a "sliding window" of the major topics of the course.

Modules (in chronological order) include:

- Basics: variables, program structure, linear control flow, style and documentation

- Conditionals: if statements, logic and logical operators

- Loops: for and while loops

- Functions: modularity, code organization, unit testing

- Passing by reference, error handling

- Arrays: dynamic memory allocation and management, memory leaks and troubleshooting using dynamic analysis

- Debugging: use of static analysis tools and a debugger (GDB)
- Strings: data processing and manipulation
- File I/O: persistence of data using files
- Encapsulation: defining and using structures and modeling data
- Recursion: using recursion to solve a problem, drawbacks and alternatives to recursion
- Searching & Sorting: basic searching and sorting algorithms, practical algorithm and efficiency analysis
- Graphical User Interfaces: Event-based programming
- Databases: relational data and persistence

3.2.1. Typical Week

Each module is organized in a consistent weekly schedule.

- Monday: module starts; required graded reading for module is due prior to first lecture; students are expected (not assessed) to have watched required tutorial videos; students attend the module's first lecture
- Tuesday: lab; a practical introduction and application of the module's topic(s) through collaborative peer-led exercises
- Wednesday: students attend the module's second lecture
- Thursday: hack session; students attend a section to "get started" on a substantial programming assignment but are not expected to finish in the session. Students are encouraged to collaborate with peers and have Learning Assistants available to help them and answer questions. The hack is due the following Monday by midnight.

4. Teaching Methods

4.1. Course Delivery

This course is designed with the principle that *many small nets are better than one large one*. A one-size-fits-all delivery method is less likely to succeed than one enabling students

to learn course content using a variety of methods. Content is presented with redundancy built-in through several different mediums including readings (both required/graded and optional/supplemental), videos (both preprepared tutorial videos as well as traditional lectures), an online discussion forum, and weekly practicums (labs and hacks).

Students are not expected to utilize *every* resource we provide. A successful student will naturally utilize one main resource and treat others as supplementary or utilized as-needed.

- **Lecture** – material is presented in a traditional lecture format on a weekly, scheduled basis (typically MW5PM - 6:15PM). Lectures are delivered in a tutorial-style manner, demonstrating concepts through live coding snippets and exercises. I work off of prepared notes but do not use slides. I produce a note set using markdown, a lightweight markup language. Notes are typed (live), while I interleave code samples and demonstrate code usage and execution with a live environment. Notes are made available to students via the github repository for the course immediately following lecture.[6]

  Lectures are presented in a large capacity lecture hall but are also livestreamed via my YouTube channel (`https://www.youtube.com/c/ChrisBourkeUNL`). Online and remote students are encouraged to watch live and are able to ask questions through a live Q&A feature via Piazza (see below). Recordings are made available immediately following the lecture and linked to via canvas. I utilize https://obsproject.com/ in conjunction with several other tools to produce these livestream lectures.

- **Preprepared Videos** – When designing this course I produced more than 60 preprepared instructional and tutorial videos which students are directed to watch prior to lecture as preparation and exposure to the topics. The videos are highly produced and available on a YouTube playlist. The content is similar to lecture material but covers different examples and provides a briefer more digestible exposure to the topics.

- **Readings** – Weekly readings are assigned which present the same weekly material for each module. Prior to fall 2020, readings were assigned out of my own free (and open) CS1 textbook, Computer Science I. Students are expected to have finished the required reading prior to lectures for each module to ensure a minimal level of exposure to the topics.

  Though a substantial subset of students took advantage of my free textbook, the majority of students rarely read the required reading as there was no direct assessment of their reading and so no *direct* consequences for not reading. Students were often ill-prepared for lecture, lab and hack sessions (which negatively impacted their

[6]For a typical example, see `https://github.com/cbourke/ComputerScienceI/blob/master/notes/1218-Fall2021/4-loops.md`

assigned partner for labs).

In fall 2020, I adopted an online interactive textbook offered through zyBooks (https://www.zybooks.com/, student cost is about $60). Weekly readings were assigned and students are assessed on readings using simple exercises at the end of each section. Students may submit and resubmit answers (no limit, no penalty) until complete. For more on assessment of reading, see Section 5.2.

4.2. Resources

In addition to the primary delivery of course content, students have substantial resources and opportunities to get help.

- **Learning Assistant Program** This course was redeveloped in conjunction with the establishment of the School of Computing Learning Assistant Program. Through a rigorous application and interview process, undergraduate near-peer mentors/instructors are recruited (and paid) to manage lab and hack sessions, grade assignments, and hold regular weekly office hours both in-person and online using Zoom. The program was modeled after other highly successful programs that have shown to improve student outcomes [6, 13, 4]. The Learning Assistant Program has been transformational for student outcomes as well as improved student experience. It has also allowed the course to be more cost effective and scalable. More undergraduates can be hired at more hours than graduate teaching assistants and, as near-peer mentors, provide much better help and assistance to students. They have gone through the course, understand its content and expectations, know what it takes to be successful, and are committed to establishing a culture of success among students. The program has grown over the last four years and now includes several other introductory courses.

- **Piazza** – The primary communication tool for this course is Piazza, a free online forum that all students are enrolled in at the beginning of the semester. All questions/emails are (re)directed to Piazza for consistency as well as pedagogical reasons. Piazza allows students the option to ask questions anonymously (or to remain anonymous to their classmates or even to instructors using private posts). It also allows students to answer each others' questions and for instructors to endorse answers or to link to posts answering the same question. The forum is used for announcements, weekly reminders, and retrospectives, as well as coordinating communication between myself and Learning Assistants. Research suggests that online forums like these that enable anonymous discussions provide a substantial benefit to students [17] and may even provide a more equitable environment [16]. My own data analysis shows a relatively strong correlation between engagement on Piazza and outcomes.

- **Misc** Several other tools and resources have been adopted or used in this course as

well. I have developed a substantial set of (python) scripts to deal with the scale of the course and automate many administrative burdens. Primarily the scripts interface with the canvas API to (randomly) assign graders and push code artifacts to Codepost.io, a website/tool that allows graders to provide line-by-line feedback to students to help them improve their code. Students are also encouraged to establish a course discord channel that we monitor but do not directly interact with or use for official communication. This is intended to (and is successful at) building a community among the students.

5. Student Assessment

Student assessment for this course is directly related to each module (and thus topic) and is uniformly distributed over each module which consists of a graded reading, a lab, and a "hack". Each weekly assessment is formative in nature; students are allowed (and encouraged) to collaborate with each other, have ample time to complete the exercises, and are allowed any number of submit/grade/resubmits using our online auto-grader system.

In addition, there is one midterm exam, one final exam and an end-of-term project. The exam proctoring has changed over the offerings to accommodate online sections and in response to COVID. In early offerings, exams were administered in class with limited time. Exams consisted of live coding exercises and so required computers. As a consequence, students were allowed to use notes and other online resources. More recently, exams were made asynchronous. Students have been given a 24 hour window in which to work on the exam. In either case, multiple resubmits were allowed and our online auto-grader system was made available.

In total, the formative module assessment has accounted for between 60% and 80% of their final grade while the exams/project have accounted for the rest. The distribution has changed over the offerings in an attempt to deemphasize formative assessments. Early data analysis indicated there was a strong correlation between grades on the module and exam grades, giving me confidence that refocusing on the modules provided assurance of student outcomes while improving the student learning experience and reducing pressure students faced.

5.1. Assessment Design

The assessment in each module is designed to provide an increasing level of familiarity with the material.

- Readings provide initial *exposure* to a topic and provide immediate interactive feed-

back on comprehension.

- Labs provide *familiarity* with a topic through *practice* and hands-on guided exercises. There is a substantial support system with peer-instruction (lab partners) and near-peer mentors (Learning Assistants)

- Hacks provide *mastery* of the module's topics by requiring students to solve a more in-depth problem

Each one of the three main assessments levels are designed to map to Bloom's taxonomy [9]. Readings (as well as lecture/videos) provide a basic *understanding* of the topics. Labs require students to practice and *apply* those skills to low-stakes exercises. Hacks provide an opportunity to *create* a substantial original program to solve a problem.

Initial levels (exposure/familiarity in readings/labs respectively) are assessed based mostly on completion while mastery is assessed based on a full rubric that includes style, documentation, design, and correctness. The majority of points are weighted toward *correctness* of solutions. Correctness is typically based on whether or not the program passes a suite of automated tests.

All assessments are designed to utilize automated grading. Students submit their programs and other artifacts (test cases, input files, etc.) and may run a *webgrader* interface that reports either expected vs. actual output or an automated report on how many *test cases* pass. From an instructional perspective, this makes grading far more efficient, scalable, and less prone to human error. It is an extraordinarily difficult task for a human to discern whether or not a program is correct. Even when someone can determine correctness, it is extremely time consuming. Automated tests solve this problem.

Automated grading also has numerous pedagogical advantages. By reducing the time needed for grading, it frees up teaching support staff to focus their efforts on face-to-face interactions with students which is invaluable. From a student perspective, automated grading provides a sort-of 24 hour automated tutor. Students are able to submit, revise, and resubmit assignments as many times as they wish prior to the due date. Automated grading provides immediate *quantitative* feedback on how well they are doing. Students tend to put in a greater amount of effort revising their work to a much higher level than they would have otherwise.

5.2. Assessment Details

1. Assignments (removed, fall 2020) – Prior to the 2018 redesign, I would typically assign 6–7 substantial programming assignments on a biweekly basis. Unfortunately, students would typically procrastinate and underestimate how much work each one would require and ultimately fail to turn in anything substantial. This directly contributed to a higher probability of failure since even one failed assignment might

doom an entire grade. Thus, *in practice*, assignments were effectively summative.

With the 2018 course redesign, the bulk of these exercises were redistributed to the more frequent weekly "hack" assignments (see below) with less weight for each. Four assignments were retained but scaled down. During the first two years of the redesign, data indicated that this change had very little impact (see Section 7). Students who performed well did well regardless of the frequency or delivery of assignments. This was disappointing in that it failed to bring down the failure/no-submission rates, but encouraging in another: it gave me justification for eliminating the assignments in favor of assessed readings (see next item) without sacrificing rigor or student outcomes.

2. **Reading** – In 2020, after a meeting with Frank Vahid, founder of zyBooks, I adopted an online interactive textbook (zyBooks) to replace the aforementioned assignments. There is ample evidence [7, 12, 2, 8] demonstrating the efficacy of this type of assessment and its impact on improved student learning and outcomes.

   Weekly readings are assigned and due prior to the beginning of the relevant module (prior to the first lecture). After a short reading or interactive demonstration, zyBooks asks students basic comprehension questions and students are allowed multiple attempts. Assessment for each reading is based on a percentage of completion. Grades are automatically populated into canvas and some basic analytics are available to instructors (which I report to students).

   The adoption of zyBooks has been very successful. Over the last two years, there is a consistent 90% submission rate. On average, students complete over 94% of the reading exercises/questions with the good majority (70%) receiving full credit on a consistent basis. In addition, students only spend an average of 1 hour per week (12 total over the entire semester over 13 readings) to complete the reading. Students highly praise zyBooks and frequently cite it as one of the primary resources for learning the material in student evaluations.

   My own observations confirm zyBook's efficacy. Prior to its adoption I assigned weekly readings out of my own free and open source textbook, Computer Science I. The obvious drawback was that readings were not monitored nor assessed and students frequently did not do them. This made them ill-prepared for each module. My first impressions of zyBooks's content was that it was a bit shallow. (especially compared to my own textbook). However, this approach proved to be effective: it clearly makes students more likely to engage with the readings and materials. The readings provide an easy and gentle introduction to the material while the lectures, labs, and hacks then provide the depth of coverage and rigor necessary to learn the material.

   I still provide my textbook as a resource to students and assign optional *supplemental* reading. zyBooks had been an option for several years though I was reluctant to adopt it due to cost as well as a fundamental philosophical/practical reason.

I'd always believed (and still do) that college students should be treated as adults and that they need to take responsibility for their own learning. This is not high school; if they don't take advantage of the learning opportunities we've provided them then that's on them. The conflicting practical consideration is that they are, in fact, *not* adults. If left to their own devices then many of them make the wrong choices (this is even true of most adults). A balance is required between enabling responsibility and being paternalistic. By adopting zyBooks I've chosen to shift a bit more toward the paternalistic side and it has proven to be effective (see Section 7.4).

3. **Labs** – Each week students attend a 1.25 hour lab session in which they are randomly paired with a partner and expected to collaborate together to complete active learning programming exercises. A handout and starter code is provided and the exercises are straightforward and graded on completion. Prior to COVID, assessment was a manual process (labs were checked by the end of the lab session); post COVID, assessment is an automated process with students having until midnight the day of the lab to submit code to an online grading system.

   Collaboration is "required" in labs as a means of building a community and providing a support system to students. In addition, each lab session is staffed by undergraduate Learning Assistants (with a 10:1 target ratio). Labs provide a hands-on practical exposure to module topics in an open and collaborative environment.

4. **Hacks** – In a second weekly "hack" session (1.25 hours) students start on a more substantial weekly programming assignment. Collaboration is allowed and encouraged but not required Learning Assistants are present to help with questions and guide students but it is less structured than labs. Students are not expected to complete their hacks during the session. They have until the following Monday by midnight.

5. **Project** – A substantial project is due at the end of the semester that integrates most of the major topics the students have learned over the course. The project is roughly equivalent to 3 weekly hacks and students are expected to work individually as a test that they have not relied too heavily on other students to complete prior work. Students get started on their project during one of the hack sessions and use another hack session as a work/help session. The project involves substantial data processing and analysis to produce several reports.

6. **Exams** – Two substantial (midterm and final) exams have been administered. Prior to COVID, these were given in-class with a high-pressure time limitation. During COVID a full 24 hour period was granted to complete the exams. Students are required to complete live coding exercises and so exams are "open book, open note, required computer" exams (though closed communication and no collaboration is allowed).

6. Analysis & Reflection

In order to engage in a continuous cycle of self assessment and reflection, it is necessary to gather and analyze as much data as possible to inform and assess how changes are made to the course to improve student outcomes. Over the past four years, both qualitative feedback and quantitative data has been collected.

Qualitative feedback is collected in the following forms.

- Student Evaluations of Teaching (SETs)
- Faculty Peer Reviews
- Learning Assistant Retrospectives

All three feedback mechanisms have shown a consistently high rating for this course and characterized it as a highly rigorous course that provides sufficient resources and learning opportunities to all students. Feedback indicates that it is possible for everyone to succeed in this course and that it prepares them for the rest of our curriculum and beyond.

6.1. Student Evaluations of Teaching

Student evaluations have been consistently high with aggregate average ratings and response rates higher than other introductory courses (see Table 2).

Table 2: Student Evaluation Averages. 2018 results are under the old evaluation system. Starting in 2019, evaluation results are from EvaluationKit. As of this draft, fall 2021 results were not yet available.

Semester	Rating	Response Rate
Fall 2018	4.57	43.94%
Fall 2019	4.36	49.31%
Fall 2020	4.30	53.00%
Fall 2021	NA	51.90%

In addition to quantitative ratings, I closely read all written feedback and perform additional analytics on it. Using a sentiment analysis tool[7] written feedback is consistently generally positive (+2.1 to +6.6) or highly positive (+26.5 to +30.1) depending on the question.

In addition to sentiment analysis, I also do a frequency analysis on questions asking students to identify the best or most helpful aspects or resources of the course and the

[7]For example, https://www.danielsoper.com/sentimentanalysis/default.aspx

least helpful ones (or suggestions for changes). Students consistently identify the following items as the most helpful to their learning.

- Lecture/tutorial videos (both the preprepared videos as well as livestream/recorded lectures)
- Learning Assistants and Office Hours (especially the flexibility of having online office hours via zoom)
- The ability to collaborate with other students[8]
- Piazza (our online discussion forum)
- zyBooks (the online interactive textbook)

To a lesser degree, students have also consistently identified labs and hacks as well as "online resources" generally as useful and positive aspects of the course.

In contrast, when asked to identify negative aspects of the course or when asked what should be changed about the course, the most common answer is "nothing." However, among those that did identify negative aspects, the most common have been:

- The pace of the course
- The amount of work or general difficulty of the material

However, both of these items have been on a downward trend since fall 2019. In 2020, most students mostly identified COVID-related issues and problems (limited in-person interactions with the instructor, Learning Assistants, and other students).

6.2. Faculty Peer Observations

Two peer observations and evaluations have been performed by School of Computing faculty. Both have involved an analysis of course material, an observation of at least one lecture, and a survey of student attitudes (in contrast to SETs from Section 6.1, these surveys were administered in the middle of the semester). Faculty observations were then compiled into a letter included in my annual evaluation and for my 2020 reappointment packet.

Both observations were very positive. The 2019 reviewer stated that "I strongly feel that CSCE 155E, which is a critically important course in our curriculum, is being taught exceptionally well, thoroughly, and with passion." The 2020 reviewer concluded that "most students taking this course feel confident they can succeed in this and future CS courses-the hallmark of a good class and excellent teacher."

[8]Notably, students greatly lamented the limited opportunities to do this during COVID especially.

6.3. Learning Assistant Feedback

At the conclusion of each semester, I solicit Learning Assistants and Course Leaders for an end-of-term course retrospective. They are prompted to identify what went well, what didn't go so well, and make any suggestions on changes or improvements. Similar analysis and reflection is performed on this feedback. All feedback is summarized in end-of-term course reflections that I include in my annual evaluations and reappointment portfolios. Learning Assistant feedback has been consistently positive both in terms of student learning experience as well as the experience of being a Learning Assistant.

6.4. Student Success

Students who persist in this course generally do very well. The nature of the assessments (auto-graded, multiple submission opportunities and substantial opportunities for collaboration) leads the plurality to majority of students to earn very high grades (see Table 3).

Overall, this course generally has a bimodal distribution with many high A grades and many failing (DFW) grades. In Section 7 we give a full analysis on DFW outcomes, but here we highlight that a substantial plurality of students are able to succeed in this course. More granular data has demonstrated a high correlation between weekly programming exercises and exam grades. Though a final grade is not a definitive metric, along with the correlation it does provide a good and reliable measure for the success of student learning outcomes.

Table 3: Grade Outcomes. Overall mean score excludes Ws but includes failing grades. ABC outcomes include the percentage and count.

Semester	Overall Mean	A	B	C
Fall 2018	80.83%	21.96% (47)	15.42% (33)	18.22% (39)
Fall 2019	72.74%	29.95% (65)	12.90% (28)	10.60% (23)
Fall 2020	75.69%	25.81% (56)	22.12% (48)	9.68% (21)
Fall 2021	77.12%	34.93% (73)	20.57% (43)	13.88% (29)

7. Addressing DFW Rates

The original goal of this course redesign was to provide better scalability, make it more economical and to provide a more rigorous and consistent CS1 experience for CS/CE majors. For the most part, these goals were achieved.

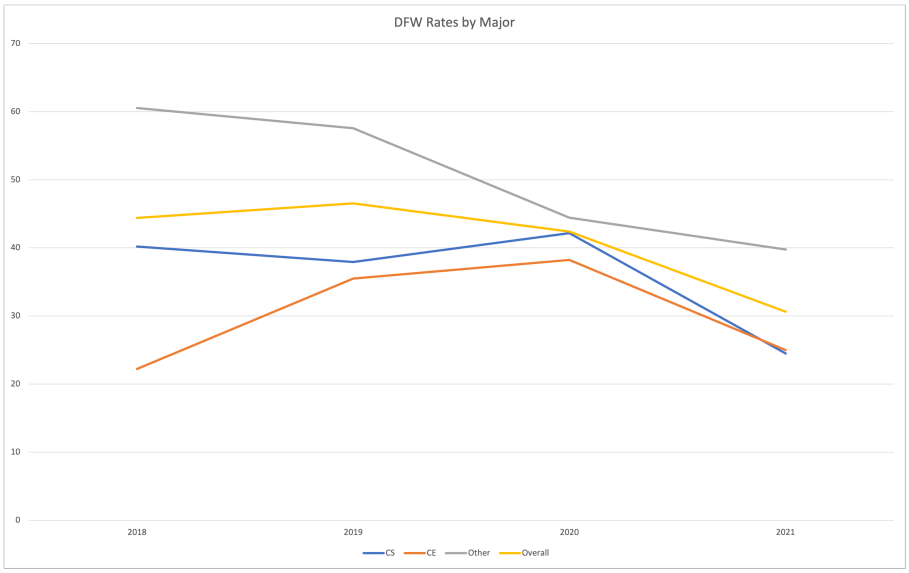

Figure 1: DFW Rates By Major

With the establishment of the School of Computing, we have developed a strategic plan with an ambitious goal of doubling undergraduate enrollment. To achieve this goal, retention and improved student outcomes are vital. One avenue for achieving this goal is to improve the DFW rates (students receiving a grade of D, F or withdrawing from the course) in our introductory course sequence. After the initial redesign of this course, this has become a primary focus for me and was main motivation for joining the FIRST program.

7.1. Background

Data from 2010 – 2018 indicates that our main Computer Science I variants have a historic DFW rate of 40–45%. For the first two years of this course redesign, the DFW rates were comparable to historic rates (see Figure 1). These "high" rates of failure are not at all uncommon. A series of large-scale studies have indicated that DFW rates of 33% are essentially invariant across time, instructors, institutions, programming language, etc.[3, 19]. Our historic DFW rates, as well as the first two years of the course redesign, are a bit higher *overall*, but when restricted to CS/CE majors are more consistent with this larger trend. Indeed, non-majors have much higher failure rates than majors.

7.2. How Students Fail

My first two years of this course focused on building the course infrastructure; making it cost effective and scalable while preserving its rigor. I was hopeful that the redesign

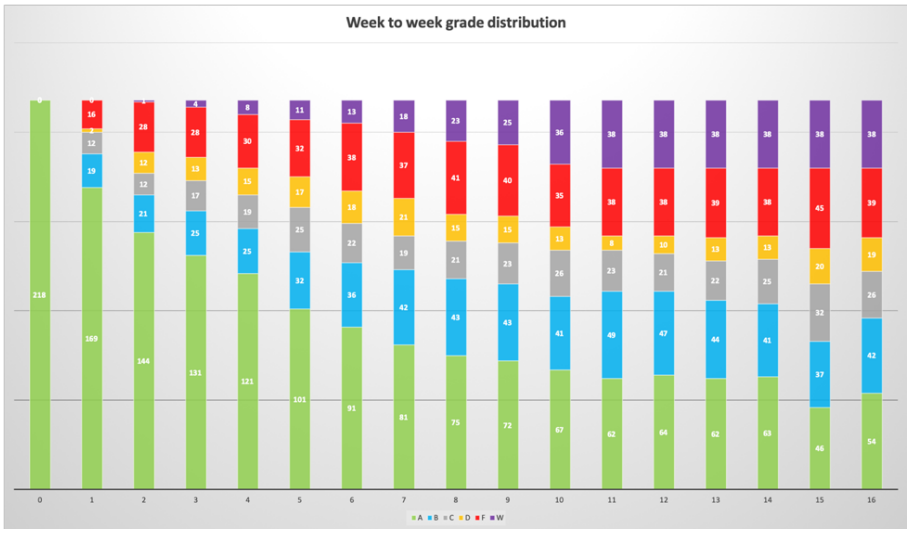

Figure 2: Week-to-Week Grade Distribution, Fall 2020.

would have improved the DFW rate. When the DFW rate remained unchanged in the first two years, I endeavored to collect more data and do a deeper analysis as to *how* students fail this course in order to find out *why* they fail and (perhaps) what could be done to change this.

Observation 1: Students who fail, fail-fast. For the last two years I collected fine-grained week-to-week data on grade distributions to find out how students failed or succeeded over time. The data is presented in Figures 2 and 3.

It is clear from this data that half of all DFW students have essentially failed by the second or third week of class. The DFW rate stabilizes by the 6th week of class (2 weeks prior to the midterm).

In 2020, among those that withdrew, only a minority (11 of 37) had a C or better when they withdrew suggesting that the primary reason for withdrawing is performance. Similarly, many low performing students likely do not withdraw in order to maintain their full time status for financial aid purposes but stop attending and engaging.

Observation 2: Students who fail may not necessarily fail due to this course. Full semester grade data was pulled for all students who took this course in the fall semesters, 2018–2020. I found that 72% of students who fail this course failed *at least* one other course in the same semester. A substantial number of them failed more than one other course. In contrast, only 22% of ABC students fail one or more other courses.

In addition, those that receive a DFW grade are 4 times more likely to drop out of UNL entirely than they are to persist (in the same major) or retain (change majors at UNL). Even for ABC students, there is a 15–30% UNL drop-out rate depending on their major. This suggests that a successful retention strategy aimed at increased enrollment

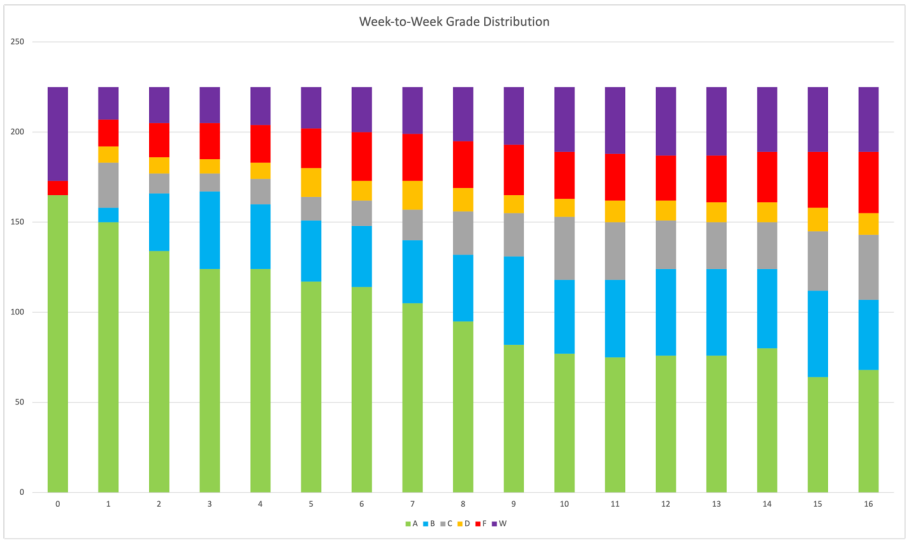

Figure 3: Week-to-Week Grade Distribution, Fall 2021.

will likely require a much larger, multifaceted effort than simply improving DFW rates in this course.

7.3. Why Students Fail

Various attempts have been made to collect data to address the question of *why* students fail. For the past 2 years I have attempted to reach out to DFW students at the end of the semester with a survey asking them for their input as to why they struggled in the course, the challenges they faced, and what could have been done that may have resulted in a different outcome. Unfortunately, the response rate in both years was less than 10% which makes it of little use as an instrument. Even among those that responded, no one identified any concrete reasons (or at least could not articulate them) other than general difficulty/workload/pace.

The nature of the material may provide a reason why a lot of students struggle. Students who fail to master one module's concepts are at a great disadvantage in subsequent modules. If a student doesn't master the concepts of loops, they will most likely not understand arrays for example. Students who fall behind cannot catch up. This compounds and leads to many of the early DFWs as supported by the week-to-week analysis above. This phenomenon is referred to in literature as *Learning Edge Momentum* [14]. Other disciplines may have introductory courses where topics are more independent so that failure to master one topic has minimal impact on a student's ability to learn other topics or to "recover" their grade.

7.4. Recent Success

The higher DFW rates have, until recently, remained frustrating. Recent literature reviews have identified successful best practices for CS1 courses that have demonstrated some level of success with respect to DFW rates [18, 11]. However, my course redesign already integrated a majority of these (8 of the 10 best practices have been integrated to some degree). Even with these best practices, the literature has found only a conservative improvement; [18] indicate that the best results achieve a modest 33% reduction in DFW grades affecting only 16.6% of the overall students.

However, things are starting to turnaround. As observed in Figure 1, DFW rates were down *substantially* for the most recent fall 2021 offering of this course. This came as even a surprise to me having observed the overall same week-to-week trend as fall 2020. After some reflection and analysis there are several possible explanations.

- It may be a fluke. Year to year data can often have substantial variations. Even the rates in Figure 1 have significant fluctuations (for example with respect to Computer Engineering majors). This could simply be a temporary improvement and rates may regress in subsequent years.

- Change takes time. Changing a culture and making fundamental and lasting improvements in a curriculum may take years to for the success to manifest. This was the fourth offering of this course. The Learning Assistant Program was much more developed and mature. The Senior Leaders in the program would have been the first students to go through the redesigned course. It may have simply taken years for the improved outcomes to manifest.

- The improvement could have been delayed due to COVID. Between the 2019 and 2020 offering several significant changes were made: assignments were replaced with zyBooks and grading was shifted to deemphasize summative assessments. This did not result in observed improvements in the 2020 offering, but at the same time no significant negative impacts were observed. This semester saw a major shift in education as COVID impacted every aspect of life. It may be that the changes did result in improvements that were simply offset by the negative impacts of COVID and remote learning.

Though COVID was still an issue, the 2021 offering saw a (somewhat) return to normal. It is my hope that these improvements will be sustained.

8. Future Changes

Whether or not the improved rates are temporary or permanent, the are very promising and provide motivation for continuing to improve and refine this course.

8.1. Attendance Policy

No attendance policy for lecture, lab, or hack has every been adopted or enforced for this course. Data collected in the first two years of the redesign, however, indicated that those who regularly attended lab and hack sessions performed one letter grade better than those who did not. In addition, qualitative feedback from both students and Learning Assistants indicated that some attendance policy should be adopted. Prior to the 2020 COVID year I had intended to require attendance at all lecture, lab, and hack sessions and to make it part of a student's assessment. Social distancing policies made it impossible to adopt this policy.

As we return to normal, I plan on experimenting with a *differential* attendance policy.[9] The aim is to give student the responsibility and agency to decide if attendance is necessary. Attendance will constitute 20% of the grade of each module. If a student attends and is *fully engaged* and actively working on the material (as judged by Learning Assistants), they receive full credit for attendance. If they cannot or choose not to attend, they can still earn the points back by earning a minimum threshold grade on the remaining assessment. If they earn 80% or better on the rest of the assessment, they will be awarded the attendance points. The idea being that they have proven their competency in the material and so requiring them to attend is unnecessary.

8.2. Limiting Collaboration

In general, plagiarism and academic integrity have not been an issue in this course. The vast majority of assessment (better than 75%) is based on exercises in which collaboration is allowed or highly encouraged. Under this design, the incentive and opportunity to plagiarize is minimized.

However, several incidents in the fall 2021 offering as well as both quantitative data and qualitative feedback have prompted me to refine our collaboration policy. In particular, we observed many more incidents of "bad" collaboration:

- The midterm saw several incidents of overt plagiarism when one student posted a solicitation for solutions to a known cheating website and a dozen students were caught using this solution.
- We observed several overly "large" clusters of teams We saw up to 6 students collaborating on certain hacks. It is highly doubtful that there was an equal effort in such a large group. In some instances we witnessed one person doing all the work and simply distributing the solution to the rest of the group.

[9] I will be experimenting with this in a spring 2022 offering of this course at the University of Omaha Nebraska.

Both of these types of collaboration are explicitly forbidden or discouraged in our code usage guidelines. Both are a result or symptom of abuses of our collaboration policy. Students who rely too much on a partner throughout the semester are ill prepared to complete assessments (exams, project) that do not allow collaboration and either fail them (honestly) or panic and resort to plagiarism.

Abuse of our open collaboration policy has always been a risk and we've observed similar incidents in all offerings but never to the extent that we saw them in fall 2021.[10] I still do not have indications that a majority of students abuse this policy, but they represent a significant enough population for me to refine the collaboration policy.

Two's company but three's a crowd. Going forward, hacks will be limited to at most two students with additional restricted collaboration outside these pairs. In larger groups the diffusion of responsibility as well as the risk of negative peer "network" pressure is too high. Students will still have the option of working alone. This will also have the added benefit of further streamlining grading (fewer submissions and more consistent grading between pairs).

I will also make a renewed effort to demonstrate (early and often) MOSS (Measure of Software Similarity, https://moss.stanford.edu/), the plagiarism detection software we use. In years past I have taken class time to demonstrate MOSS using old anonymized code samples to show how easy it is to detect and report plagiarism. Traditionally, this has been enough to dissuade attempts to violate our academic integrity policy. However, the limited number of incidents in recent years has made me complacent and I've neglected this practice (which I always detested as I always saw it as creating a negative, fearful environment). I will be making a conscious effort to renew this practice.

8.3. Placement Examination

Another way to reduce the DFW rates of this course is to improve the placement of students into a more appropriate course.

This course is designed for majors yet it still has a substantial population of non-majors who have a substantially higher DFW rate and who may be better served by taking a computing course intended and designed for non-majors.

For majors, there is a substantial population of students that are generally ill-prepared for college. Though they could still be successful in our majors, they may be better served by a less rigorous introduction to computing in order to build a better foundation and acclimatize to the college life.

[10] I do not believe that "bad collaboration" led to inflated grades explaining the improved DFW rates. Those involved in bad collaboration or plagiarism for the most part failed the course with poor exam and project grades.

The School of Computing offers an introductory course using Python, CSCE 101. In practice, this has been a terminal computing course, but recent redesigns and redevelopments have improved this course to the point that it could serve as a better entry point to computing for those that are not ready to take a full Computer Science I course. In 2020, I pushed for changes to our curriculum to allow majors to count this course toward their degree provided they took it before the regular sequence.

However, there is currently no instrument to determine if a student should take CSCE 101 or is ready to take CS1. One solution is to develop a Computing Placement Exam similar to the math placement exam to place students who are most at-risk for failing CS1 into CSCE 101 first.

There are several validated placement exams in the literature. Unfortunately, they usually focus on differentiating enrollment into either CS1 or CS2 rather than CS0 or CS1. These exams focus on prior computing/programming knowledge which is not assumed for someone entering CS1. A placement exam for CS0 vs CS1 should instead focus on general skills and self-efficacy to determine readiness.

During the fall 2021 semester, I've been working with UNL researchers to collect data in a "Computing Skills" survey with the hopes of developing a placement exam. It is my hope that a preliminary instrument can be in use for fall 2022.

9. Conclusion

We have presented the mechanics and logistics involved in the redevelopment of the School of Computing's flagship Computer Science I course, CSCE 155E. We have presented data and analysis indicating that, though this is a difficult and rigorous course, there are sufficient resources and opportunities for all students to be successful. Though it has experienced high historic failure rates, there have been substantial recent improvements. This course will continue to develop and evolve over the coming years and its importance will only increase as the School of Computing continues to grow.

A. Appendix

Supplemental documents are listed below but not included (for size) in this document.

Course Reflections

Every semester I write comprehensive course reflections. Anonymized Learning Assistant feedback is included in each. Reflections are included for years 2018 thru 2021.

Faculty Peer Evaluations

Two peer evaluations have been performed for this course over the last four years, Dr. Stephen Scott in Fall 2019 and Dr. Justin Bradley in Fall 2020.

Student Evaluations of Teaching

Prior to 2019, student evaluations were administered through the UNL Course Evaluation System. In 2019, the School of Computing adopted EvaluationKit. Evaluations are included for years 2018 thru 2020 (2021 is pending).

References

[1] Alireza Ahadi and Raymond Lister. Geek genes, prior knowledge, stumbling points and learning edge momentum: Parts of the one elephant? In *ICER 2013 - Proceedings of the 2013 ACM Conference on International Computing Education Research*, pages 123–128, August 2013.

[2] Joe Michael Allen, Frank Vahid, Kelly Downey, and Alex Daniel Edgcomb. Weekly programs in a cs1 class: Experiences with auto-graded many-small programs (msp). In *2018 ASEE Annual Conference & Exposition*, number 10.18260/1-2–31231 in ASEE '18, Salt Lake City, Utah, June 2018. ASEE Conferences. https://peer.asee.org/31231.

[3] Jens Bennedsen and Michael E. Caspersen. Failure rates in introductory programming. *SIGCSE Bulletin*, pages 32–36, 2007.

[4] Hannah C. Chapin, Benjamin L. Wiggins, and Linda E. Martin-Morris. Undergraduate science learners show comparable outcomes whether taught by undergraduate or graduate teaching assistants. *Journal of College Science Teaching*, 44(2):90–99, 2014.

[5] John Doucette. High failure rates in introductory computer science courses: Assessing the learning edge momentum hypothesis. Center for Teaching Excellence Blog, University of Waterloo, https://cte-blog.uwaterloo.ca/?p=5153, September 2015. Accessed: 2019-11-11.

[6] Denise Drane, Marina Micari, and Gregory Light. Students as teachers: effectiveness of a peer-led stem learning programme over 10 years. *Educational Research and Evaluation*, 20(3):210–230, 2014.

[7] A. Edgcomb, F. Vahid, R. Lysecky, A. Knoesen, R. Amirtharajah, and M.L. Dorf. Student performance improvement using interactive textbooks: A three-university cross-semester analysis. In *Proceedings of ASEE Annual Conference*, June 2015.

[8] Alex Edgcomb, Frank Vahid, Roman Lysecky, and Susan Lysecky. Getting students to earnestly do reading, studying, and homework in an introductory programming class. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '17, page 171–176, New York, NY, USA, 2017. Association for Computing Machinery.

[9] Ursula Fuller, Colin G. Johnson, Tuukka Ahoniemi, Diana Cukierman, Isidoro Hernán-losada, Jana Jackova, Essi Lahtinen, Tracy Lewis, Charles Riedesel, Errol Thompson, and et al. Developing a computer science-specific learning taxonomy. *SIGCSE BULLETIN*, 2007.

[10] Mark Guzdial. Teaching computer science better to get better results. Com-

puting Education Research Blog, https://computinged.wordpress.com/2014/10/15/we-need-to-fix-the-computer-science-teaching-problem/, October 2014. Accessed: 2019-11-11.

[11] Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A. Becker, Michail Giannakos, Amruth N. Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and et al. Introductory programming: A systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2018 Companion, page 55?106, New York, NY, USA, 2018. Association for Computing Machinery.

[12] Dawn McKinney, Alex Daniel Edgcomb, Roman Lysecky, and Frank Vahid. Improving pass rates by switching from a passive to an active learning textbook in cs0. In *2020 ASEE Virtual Annual Conference Content Access*, number 10.18260/1-2–34792 in ASEE '20, Virtual On line, June 2020. ASEE Conferences. https://peer.asee.org/34792.

[13] Stephanie B. Philipp, Thomas R. Tretter, and Christine V. Rich. Undergraduate teaching assistant impact on student academic achievement. *Electronic Journal of Science Education*, 20(2):1–13, 2016.

[14] Anthony Robins. Learning edge momentum: a new account of outcomes in cs1. *Computer Science Education*, 20(1):37–71, 2010.

[15] Mehran Sahami and Chris Piech. As cs enrollments grow, are we attracting weaker students? In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 54–59, New York, NY, USA, 2016. Association for Computing Machinery.

[16] Adrian Thinnyun, Ryan Lenfant, Raymond Pettit, and John R. Hott. Gender and engagement in cs courses on piazza. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, SIGCSE '21, page 438–444, New York, NY, USA, 2021. Association for Computing Machinery.

[17] Mickey Vellukunnel, Philip Buffum, Kristy Elizabeth Boyer, Jeffrey Forbes, Sarah Heckman, and Ketan Mayer-Patel. Deconstructing the discussion forum: Student questions and computer science learning. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '17, page 603–608, New York, NY, USA, 2017. Association for Computing Machinery.

[18] Arto Vihavainen, Jonne Airaksinen, and Christopher Watson. A systematic review of approaches for teaching introductory programming and their influence on success. *ICER 2014 - Proceedings of the 10th Annual International Conference on International Computing Education Research*, 07 2014.

[19] Christopher Watson and Frederick W.B. Li. Failure rates in introductory program-

ming revisited. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*, ITiCSE '14, pages 39–44, New York, NY, USA, 2014. Association for Computing Machinery.