Engineering a Complete Curriculum Overhaul

Luther Tychonievich
University of Virginia
Charlottesville, Virginia, USA
tychonievich@virginia.edu

Mark Sherriff*
University of Virginia
Charlottesville, Virginia, USA
sherriff@virginia.edu

ABSTRACT

We present an eight-year curriculum redesign effort impacting almost every course in our computer science department. Having not made a major update to our curriculum in two decades, complications began to arise from significant increases in enrollment and instituting multiple degrees in computing in the same department. Starting from a desire to adjust a few courses, we systematically collected a broad set of requirements and blue-sky ideas from many stakeholders, resulting in an unsatisfiable set of content, ordering, and course boundary constraints. After multiple rounds of conversation with our stakeholders in and out of the department, we evolved and relaxed several of our constraints, allowing us to develop a compromise plan for seven new courses and a new prerequisite system. We then piloted five of the new courses and collected feedback on results, iterating on these courses each semester for two years. We worked with registrars, advisors, and administrators to develop a transition plan from old to new courses. This paper presents highlights of each step of this process, a summary of the resulting curriculum design, and reflections and recommendations for other departments that may want to undertake a similar update.

CCS CONCEPTS

• **Social and professional topics → Computing education programs**.

KEYWORDS

curriculum; redesign; multiple majors; course specification

ACM Reference Format:
Luther Tychonievich and Mark Sherriff. 2022. Engineering a Complete Curriculum Overhaul. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2022), March 3–5, 2022, Providence, RI, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3478431.3499287

1 INTRODUCTION

When we describe a computer science curriculum – meaning the set of courses we teach in our department and the content we expect students to learn in each – we accept as self-evident that it is not a fixed, monolithic structure. Every instructor changes the classes that they teach for a myriad of reasons: advances in the field, updates in pedagogical practice, changes in student background, personal teaching styles, experiments to smooth over rough patches, and so on. Courses are not fixed points, meaning that the curricula composed of them are also not fixed.

As with software maintenance, we observed that this form of ad hoc curriculum maintenance contributed to a form of brittleness [13, 16, 19]. Efforts from different faculty working to optimize their own courses in isolation from the rest of the curriculum led to the courses fitting together less and less well over time. Further, as pedagogical advances are made along with advances in the field of computing, courses that that were effective in their methods and learning objectives twenty, ten, or even five years ago may no longer be as effective today.

Changing how content is distributed across a set of courses is much more complicated than changing each course in isolation. It impacts every degree program that uses any of the replaced courses. It is complicated by students who are in in the middle of their degree program when the change occurs. Because it is a large-scale change, it attracts more attention, causing stakeholders to notice when previous single-course changes were not aligned with their priorities. It also invites a wider range of innovative and mutually-contradictory designs as it is unconstrained by predefined course boundaries.

The decision that it was time to move forward with a major re-envisioning of our own curriculum was made by a group of our faculty over lunch at the 2014 SIGCSE Technical Symposium in Atlanta, GA, understanding that it was a significant undertaking and thinking it might be a 2–5 year project. Eight years later, we are finally beginning to offer the first full versions of our new courses and enacting the transition to the new curriculum – a process that will take at least another two years to complete.

In this paper, we describe and explore the process we went through to complete our curriculum redesign. We use the analogy of the software engineering life-cycle, from requirements elicitation to implementation and testing, to show our process and thinking at each stage, culminating with the final product that we are excited to begin using in our department. **Our goal with this report is to help other departments interested in redesigning their curriculum to understand our decisions and experiences at each phase in the process so that they can plan for these steps and move through them more smoothly.** We expect each institution would arrive at a different set of courses than we did, but are hopeful that the process we went through will be useful to others undertaking a similar change at their institution.

*Both authors contributed equally to this research.

This work is licensed under a Creative Commons Attribution International 4.0 License.

SIGCSE 2022, March 3–5, 2022, Providence, RI, USA.
© 2022 Copyright is held by the owner/author(s).
ACM ISBN 978-1-4503-9070-5/22/03.
https://doi.org/10.1145/3478431.3499287

2 RELATED WORK

There have been various reports providing curricular recommendations [2, 12, 25–27], implementing [17, 18] and analysing those recommendations [11, 21, 29], and surveying multiple curricula [3, 30]. While some of these have collected requirements and designed curricula, they have not addressed the process of implementing a curriculum change impacting multiple degrees and thousands of students.

Hoag described the creation and two revisions of a cybersecurity curriculum over the course of five years [15]. We describe a much slower process for changing a large, established CS curriculum with many stakeholders and constraints.

Others have reported efforts to design a curriculum to serve multiple stakeholders. Bills and Canosa analyzed the requirements of several computing disciplines and recommended first-year curriculum to serve them all [4]. Boisvert et al aligned the needs of high schools, colleges, and industry [6]. We describe a curriculum serving stakeholders across disciplines in our university; we also discuss the processes we used to implement the change after its design.

Many previous reports have discussed ways to infuse specific content into a curriculum [1, 5, 9, 14, 20, 28, 31, 34]. While our redesign does infuse ethical and social issues across the curriculum (see §9.2.3), our focus is on restructuring the curriculum as a whole.

3 CURRICULUM HISTORY

Our university is an R1 institution in the United States graduating around 4,000 undergraduates per year. When this revision project began in 2014, the Department of Computer Science had around 250 majors per class year across three degree programs – Bachelor of Science in Computer Science (BS CS), Bachelor of Arts in Computer Science (BA CS), and Bachelor of Science in Computer Engineering (BS CpE) [22–24]. At this time, our department consisted of 28 faculty, including both tenure-track and teaching-focused. As of 2021, we have grown to around 700 majors per class year with roughly 45 active faculty[1].

The last major update to our curriculum occurred in 1989 when the BS CS was our only degree. That update placed a brand-new course, *Program and Data Representation* (PDR), in a central role in the curriculum. PDR introduced the various abstraction layers of computing, from registers up through advanced data structures, and was the sole prerequisite for almost all CS electives. Object orientation and digital logic design were taught in pre-PDR courses while most systems content (including pipelined architectures and operating system design), algorithms, software engineering, and computational theory were in required post-PDR courses.

Overall, the change and resulting curriculum was well-reasoned and well-accepted by both faculty and students. Subsequently, the curriculum was accredited by ABET and has been successfully re-accredited each cycle. The change garnered external praise and was often cited by our accreditors and industrial partners as a key component in our curriculum.

[1]We have many faculty who split time between two departments with dual appointments, making an exact count of CS faculty difficult.

In subsequent years, we made many changes to the curriculum. Courses updated their content based on changes in the field; prerequisites changed to accommodate new degree programs; and many new courses were introduced. However, each change was localized to one or two courses and the overall structure of the BS CS curriculum remained unchanged.

Our BA CS degree is similar to the BS CS, but due to restrictions on the number of credits we can require, four of the required courses from the BS CS were omitted. BA students have the opportunity to take these courses, but they can graduate without them.

The BS CpE degree is jointly administered with the Department of Electrical and Computer Engineering, and contains a mix of Computer Science and Electrical Engineering courses as well as some CpE-specific courses. The CS portion of the BS CpE is similar to the BS CS, but they take networking instead of theory of computation and have a different computer architecture course sequence.

The CS courses in all three degrees are displayed in Figure 1.

4 WHY MAKE A CHANGE?

As we updated our curriculum in a series of local changes, each individually adequate, the cumulative effect slowly made the curriculum less cohesive. It gradually became evident that a full re-imagining of the curriculum was necessary. Several of the drivers of change are discussed below.

4.0.1 Challenges of Uniqueness. Our PDR course is not a model seen at many universities, and, while popular among our students, that difference proved an increasing problem over time. Having a unique course central to the degree program limited transfer students and study abroad opportunities. Its uniqueness made it hard for new faculty to teach the course and meant that faculty of subsequent courses often misunderstood prerequisite knowledge students gained in it. With growth and time, it became clear we needed to redistribute the material in this course.

4.0.2 Multiple Degree Programs. When our curriculum was last redesigned, it served only a single degree program, the BS CS.

Students in the BS CpE take a different course series to learn computer architecture, meaning the courses that depend on computer architecture have students enter with varying experience. Both the CS and CpE computer architecture courses have evolved since splitting, increasing that difference.

The BA CS was created to allow students in the university's College of Arts and Sciences to major in CS without transferring to the School of Engineering and Applied Sciences. The program was created as an Interdisciplinary Major, requiring fewer approvals but significantly limiting the number of credits it could require. Multiple prerequisites were adjusted to allow BA CS students to reach later courses without having taken courses in digital logic, computational theory, and advanced mathematics.

Because of the order in which degrees have been added, our curriculum is structured as a BS CS degree with accommodations for other programs. However, BA CS and BS CpE students represent well over half of our students. Non-major interest in CS classes has also grown, through both through the CS minor and through students in other majors being advised to take our classes. Our

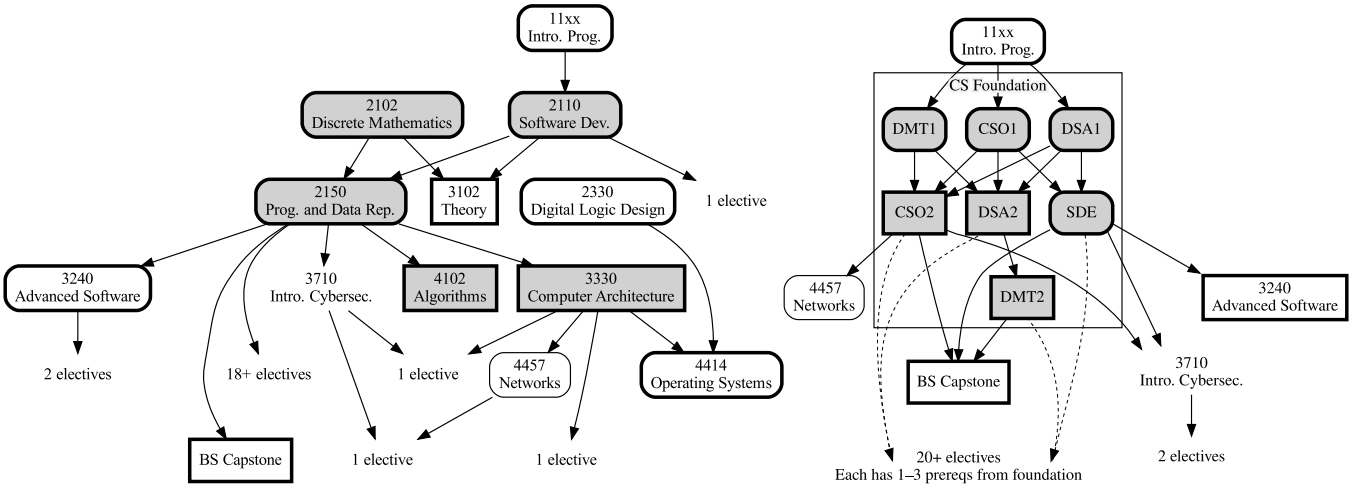

Figure 1: Prerequisites before (on left) and after (on right) change. Heavy borders indicate courses required by the BS CS degree, shaded nodes by the BA CS degree, and rounded edges by the BS CpE degree. The BS CpE degree also has many additional courses administered by Electrical Engineering department (not shown), some of which can satisfy prereqs for CS courses.

students and course sequences no longer look like they did at the last major curriculum redesign.

4.0.3 Updated Pedagogy and Course Material. Significant advances have been made by SIGCSE and related communities over the past 30 years in understanding what makes the most effective CS pedagogy. Implementing new pedagogy in our courses sometimes adjusted the order and level of detail of different topics, accidentally weakening the alignment between courses.

Many new topics have become important in our field since 1989; as one example, 30 years ago curricular recommendations for security focused on user accounts and type checking, not vulnerabilities and encryption [32]. As the computing field has evolved, so have our courses, but in an ad hoc way, adding new topics and removing or de-emphasizing other topics to make room. The resulting curriculum has some topics in many places while other topics are not covered to our desired level of detail.

4.0.4 Teaching More Students. Our student body has increased more than tenfold since our last curriculum redesign, and some previous curricular designs that had been designed to create or capitalize on a sense of community, cooperation, or friendly competition did not scale to a much larger student body. Our student/faculty ratio has also increased by a factor of five, meaning practices based on individual faculty-student contact have not aged well. Undergraduate teaching assistants have been key to our adapting to this rapid growth, and both instruction and assessment have been adjusted to make the use of TAs more effective.

4.0.5 Teaching a More Diverse Students Body. Some students enter the university with much more computing experience than they did 30 years ago, and simultaneously we have made intentional strides to be more welcoming to students with less experience and more tech hesitation, leading to a much wider spread of incoming experiences. We have become a major service-course provider for the university, with half of all university students taking our introduction to programming course. Part of our success in attracting more diverse students has come in recruiting to the major from that introduction to programming course [7, 8], resulting in an increased desire for it to be a good fit both for those who will never take another CS course and for those who intend to declare our majors[2]. To best serve this wide range of students we've changed the focus of early courses, with ripple-on effects to later courses.

5 REQUIREMENTS ELICITATION (2014–2017)

Before we began designing the new curriculum, we needed to understand the goals and desires of our stakeholders. Specifically, wished to balance the varying objectives of multiple degree programs, faculty both in and out of CS, transfer students, administrators, accreditors, and industry partners. We elicited requirements from each, a multi-year process that included the following:

5.0.1 Reviewing ACM/IEEE CS2013 Curricular Recommendations. When we began redesigning the curriculum in early 2014, the CS2013 Curricular Recommendations [2] had recently been released. Our undergraduate curriculum committee (UCC) began a two-part detailed review and curriculum audit based on those recommendations. Each UCC member was assigned a few Knowledge Areas from CS2013 and worked with instructors of relevant courses to evaluate how much of that material we currently taught. A report was generated [10] and used in committee and faculty meetings to discuss possible changes in coverage. These meetings often resulted in decisions to more closely match CS2013, but sometimes we agreed that we liked our different levels of focus and would keep those differences going forward.

5.0.2 TA "Blue Sky" Exercises. Our department depends heavily on undergraduate teaching assistants, who enroll in a 1-credit training course [33]. For three years, we included a session in that course

[2] At our university, students declare their major 1–2 years after matriculating.

where we explained the current curriculum, then had TAs design, describe, and defend changes to it. These sessions helped us discover material that was being repeated and prerequisite material that was not being taught; helped us understand our students' perspectives; and provided us with a wide variety of draft redesigns to consider.

5.0.3 Faculty Interviews. Not all faculty speak up in meetings or respond to open-ended emails, so we sent UCC representatives to visit with each faculty member and ask them about the curriculum and how it should change. This helped us better understand limitations of our current courses and directions for change.

A key insight from these interviews was that the set of topics our faculty considered to be "core" to CS was larger than we could hope to teach to all students. This insight, and a subsequent multi-month full-faculty discussion of what was "core" to CS, led us to change our plans and vocabulary from covering the "CS core" to defining the "CS foundation", the prerequisite knowledge required by multiple sub-fields of computing.

5.0.4 Degree Programs. Our courses have many clients, each of whom had some non-functional requirements. The BA CS degree needed to have prerequisites be no more than four-semesters deep. The BS CpE degree needed to offer alternate paths through parts of the curriculum to better integrate with electrical engineering courses. The computer science minor needed to let students reach a CS elective after only a few prerequisite courses. Our university has a matriculation agreement with our state's community college system, making alignment between their courses and ours desirable. Roughly a dozen other degree programs use one or two of our courses, each with different objectives from them.

Discovering these requirements involved conversations with deans, administrators, and faculty from many departments. The resulting set was not readily satisfiable, so additional meetings were needed to learn which were hard constraints we needed to meet and which were less rigid and could be changed with limited impact on the respective programs.

6 DESIGN (2017–2018)

As we began to understand our requirements, we started brainstorming ways of better meeting them. This proceeded in two phases.

Phase 1 explored high-level course content and prerequisite structures. We used a whiteboard with magnets for courses connected by lines for prerequisites and notes about content on the side. We invited faculty to view and modify this representation, which helped them express their ideas in their areas of interest without losing sight of the whole. Phase 1 ended with general consensus on the set of new courses.

Phase 2 fleshed out those courses in committees. Committees were formed for each group of related courses and were staffed by faculty who had taught that subject in the recent past. Committee processes varied, but generally operated with a committee chair proposing drafts of course content and using the committee as a brainstorming and advising group.

Throughout the design process we often faced the need to compromise between competing requirements. Some common competing interests include:

- Tightly integrated courses *vs.* flexibility to use a different subset of courses in each degree program.
- Innovative designs *vs.* compatibility with other programs.
- Deep understanding of basics *vs.* covering more topics.

We resolved each compromise based on our estimation of need and ability in our context. Some of these resolutions were very specific negotiations between the instructors of a course under design and a course or degree program that would depend on it, finding new homes for important material that would not fit in one of our courses. We expect each institution going through this process will resolve these conflicts in an individual way.

7 IMPLEMENTATION AND TESTING (2018–2020)

Changing our curriculum involved changes to courses and to university systems.

7.1 Piloting New Courses

To test changes to our courses, we ran a two-year pilot study of our new courses. A pilot was needed to refine the new courses in single-section offerings and to gain faculty buy-in prior to rolling out the new courses at the full 3–4-sections-per-semester scale. A full discussion of the pilot is beyond the scope of this report, but a few highlights of our pilot process may be useful to others considering doing their own curriculum redesign pilot.

Because content was moving between courses, it was important for each student to either take all the piloted courses or none of them. That requirement effectively created two concurrent kinds of CS students: those taking the standard courses and those in the pilot. Those in the pilot required separate advising by specific pilot-aware academic advisors; separate academic requirements, implemented as a multi-step exception process in the university record keeping designed in consultation with a registrar; and case-by-case academic planning for students who failed a pilot course or were otherwise unable to complete the pilot and needed to finish their requirements with non-pilot courses.

All students who entered the pilot needed to be in the same position in their academic journey in the same semester. Students on the usual path in our curriculum generally would be at that point in a Spring semester, while those either ahead or behind would be at that point in a Fall semester. We chose to start the pilot in a Fall semester to ensure we learned how well the courses worked for less-standard students.

We ran the pilot twice in two consecutive years, incorporating changes from the first year into the second year's offerings. The repeated offering helped us refine the courses and made us more comfortable with them as our new requirements. However, repeating the pilot created a significant PR problem: the first year pilot students were so vocal in their appreciation of the new courses that in the second year the student rumor mill labeled the students in the pilot as specially privileged. As that reputation grew, we found it important to change the conversation to emphasize the pilot as primarily an experimental reordering of material instead of emphasizing that it was an update and improvement. That said, this word-of-mouth positive reputation was helpful in convincing more stakeholders that the change was positive and worth the effort.

7.2 Updating Rules

Changing university systems was a multi-tiered process. As a state-sponsored school, there were as many as five different levels of approval to go through[3], a process that took more than a year after the design was finalized. More importantly, if less visibly, we spent many hours iterating with registrars and deans and directors of other degree programs to ensure that the new courses were properly coded, that orientation materials were available for them, that we had transition plans handling students who had some of the old courses and wanted to take some of the new courses, and so on.

Beginning conversations early, listening carefully, and making adjustments that seemed small to us but were important to others helped this process go forward without any significant obstacles.

8 DELIVERY (2021 ONWARD)

Final approval of the new courses was given during the 2020–2021 academic year, with the new curriculum rules for the BS CS going into effect for the 2021–2022 academic year. Due to approvals needed at the state level[4] and from multiple departments, we anticipate the new curriculum rules for the BA CS and BS CpE degrees to go into effect for the 2022–2023 academic year. The first non-pilot versions of the new courses will be offered in the fall of 2021.

In designing the transition path, we first identified which new courses were close enough to current courses that students could take either one and still have a reasonable expectation of having learned the prerequisite content needed for subsequent courses. That still left a set of six courses in the old curriculum whose content had been redistributed over five courses in the new curriculum in a way that made mixing the two inadvisable.

We then created a schedule of when each new course would first be offered and when each old course would last be offered and moved through it for each degree program, ensuring that regardless of which semester a student first took a course in either sequence, all of the remaining courses in that sequence would be available to them. After several iterations, we found a schedule that meets this goal with most courses being offered with only a single semester of overlap between old and new offerings, a level manageable within our current course staffing process.

9 NEW CURRICULUM OVERVIEW

9.1 Introduction to Programming

We changed our introduction to programming courses (CS1) early in the curriculum design process because clear requirements impacting only a few courses emerged early in the process. Prior to the change, CS1 taught basic object-oriented programming in Java, an approach that did not serve the needs of non-majors. After the change, CS1 teaches basic imperative programming with functional decomposition and basic debugging skills; it is formally language-agnostic, thought most sections teach Python. While designed to help non-majors, other departments, and transfer students, this change has also proven popular among CS faculty.

[3]The number of approval levels varies between the College of Arts and Sciences and the School of Engineering and Applied Science, and both were required to change the BA CS and BS CS.

[4]The BA CS changes required state approval due to transitioning from an Interdisciplinary Major to a stand-alone program.

9.2 The CS Foundation

We identified seven courses as the new CS Foundation. All of these depend on CS1, and collectively they provide the prerequisite knowledge for almost all[5] CS electives.

The foundation is divided into two levels: the first level depends only on Introduction to Programming, while the second level depends on the first level of foundation courses. The prerequisite chain of the courses (new and old) is presented in Figure 1.

9.2.1 First Level. The first level of the foundation intentionally consists of courses that are similar to courses taught in our local community colleges and in many other institutions. We want these courses to be easy to fill with transfer credit.

Data Structures and Algorithms 1 DSA1 is similar to other a data structures in Java courses, including searching and sorting algorithms and basic asymptotic analysis. DSA1 also introduces statically-typed languages and the use of an IDE. DSA1 includes a brief introduction to threads and concurrency to queue up the topic for later courses.

Discrete Math and Theory 1 DMT1 is similar to other discrete mathematics courses. It is more proof-oriented than some other offerings, emphasizing prose proof construction, mathematical induction, and conversion between English and Math, as well as introducing sets, functions, and logic. DMT1 includes a brief introduction to state machines, which are used to explain topics in later courses.

Computer Systems and Organization 1 CSO1 is an introductory systems course, covering binary, assembly, and C. CSO1 also teaches how to use command-line tools and manual pages. Not all schools offer a course like CSO1, but our faculty deemed it important for many fields, including security, cyberphysical systems, and emerging architectures.

9.2.2 Second Level. The second level foundation courses build off the first with the joint goal that all the major prerequisite knowledge needed by an introductory electives in any subfield of computer science is covered by some combination of these courses. Unlike the first level, alignment with other programs was not a design objective.

Data Structures and Algorithms 2 DSA2 is similar to many algorithms courses, covering the implementation and analysis of common greedy, dynamic programming, and graph algorithms. DSA2 also covers the basic concepts of machine learning: the structure of data-driven algorithms generally, the use of training and test data, and and introduction to neural networks.

Discrete Math and Theory 2 DMT2 is primarily a computational theory courses, covering languages and machines, computability, and complexity classes. Because it has DSA2 as a prerequisite, it includes analysis of specific nontrivial algorithms and algorithm families. Because it has CSO1 as a prerequisite, it discusses unclocked digital circuits as an example of a realistic non-Turing-complete computational model. DMT2 also serves as the course where students learn how to design their own proofs.

[5]The exception is tightly-coupled course sequences, such as Introduction to Cybersecurity being a prerequisite to two advanced security courses.

Computer Systems and Organization 2 CSO2 is a second systems course, covering concepts needed to understand hardware-based exploits like Meltdown, crpytographic protocols like HTTPS, and performance optimizations like lock-free concurrent and cache-aware data structures. Major topics come from architecture (caches, out-of-order processors, and virtual memory), operating systems (kernel vs user mode, atomic operations and synchronization, and permission lists), and networking (sockets, TCP/IP, and HTTP).

Software Development Essentials SDE serves as an introduction to software engineering and the development of software projects. Topics include object-oriented design, the development lifecycle, unit testing, data languages like XML and JSON, and the basics of databases and SQL. SDE also teaches tools including testing frameworks, source control systems, and automated build tools.

9.2.3 Crosscutting Concerns. During the design of the new curriculum, some topics were discussed and found most appropriate to distribute across all foundations courses instead of putting them in just one single course.

Ethical and social issues are essential for all CS students. To ensure all students see these topics, even those who take only a few CS courses, each foundation course has specific ethical and social issues it covers. These topics are selected to be motivated by and related to that course's content; for example, CSO1 teaches the command-line so one of its topics is the equity conflict inherent in "power-user interfaces" (like command-lines) that give more power to those who have the resources necessary to learn them.

We want students to gain proficiency in some programming language at a level greater than they'd gain in a single semester. Because of this, DSA1, DSA2, and SDE all have a non-functional requirement that all of their programming be done in the same language (currently Java).

9.3 Electives

Most computing elective courses in our old curriculum identified one course – Program and Data Representation – as the only prerequisite. With the new CS Foundation, we were able to create much more fine-grained prerequisites for upper-level electives, allowing students to potentially take them earlier in their program, while simultaneously taking the enrollment pressure and potential stress out of a perceived "weed-out course." We also anticipate the new structure will help faculty better understand what students come in to their course knowing.

10 LESSONS LEARNED

In our eight-year (and still ongoing) experience with evaluating and redesigning our curriculum, there are some key lessons that we learned that we believe may aid other departments who want to undertake a similar undertaking.

10.0.1 Listen to All of Your Stakeholders. As we began to engage with faculty, students, accreditors, administration, staff, and industry partners, it became increasingly clear that every group of stakeholders had their own needs and insights. If we had only worked with just the computer science faculty, we would have overlooked numerous potential opportunities and issues that could arise from the new curriculum.

10.0.2 Take Your Time to Get it Right. There is a definite desire when going through a curriculum change to just "get it done," particularly if there is momentum from the faculty and students. However, there are numerous concerns that have to be addressed at each phase of the process. Establish a timeline and goal for your effort, identifying key dates for when the different phases should be completed, and work toward those goals. Be realistic with yourself with how long it will take to complete each phase. It is better to take longer and ensure that all stakeholders are on board and invested in the changes rather than to try to push through quicker.

10.0.3 Avoid Confusing or Controversial Naming. We learned early on that small things, such as calling the set of required courses "core" classes, can quickly become rallying points for those that are resisting change. This problem also arose with the original names and numbers selected for the new Foundation courses. When these difficulties begin to appear, immediately work with the parties involved to determine what they believe a proper name should be. Not only will you hopefully avoid the problem, but the parties involved may feel even more invested in the process.

10.0.4 Pilot New Courses. Piloting new courses is a major undertaking, but one that is well worth it if you have the time and resources. Since we were creating new courses, we wanted to make sure that the structure and delivery of the material matched with the expected learning outcomes. Piloting the courses also led to positive word-of-mouth about the upcoming curriculum changes and students asking when they could start the new courses. While we encourage piloting new courses, departments need to consider potential difficulties that will arise:

- What happens if a student fails a pilot course?
- How will exceptions in student records be handled?
- Who will do advising for these students?
- How will you gather feedback and use that to improve the courses?

10.0.5 Plan for a Long Transition. Similar to taking your time while designing the new curriculum, know that there is often not an opportunity to just "switch over" all students into a new curriculum. Plan out how many semesters you will need to teach previous courses and how you will introduce the new courses. You will also need to create new advising material to guide new students into the new curriculum instead of starting to take older courses. Share these plans with administration and, more importantly, any student-facing staff members in the department and registrar's office. These individuals often take the brunt of student questions and will need up-to-date information to aid them.

ACKNOWLEDGMENTS

While the authors led this effort, it would not have happened without the support of more than 150 contributors, including faculty, students, and especially our long-suffering and infinitely-accommodating student-facing staff. Thanks also to our department leadership who remained committed even as our initial two-year brainstorm turned into a ten-year process.

REFERENCES

[1] Ken Abernethy and Kevin Treu. 2014. Integrating Sustainability across the Computer Science Curriculum. *J. Comput. Sci. Coll.* 30, 2 (Dec. 2014), 220–228.

[2] ACM/IEEE-CS Joint Task Force on Computing Curricula. 2013. *Computer Science Curricula 2013.* Technical Report. ACM Press and IEEE Computer Society Press. https://doi.org/10.1145/2534860

[3] Ismail Bile Hassan, Thanaa Ghanem, David Jacobson, Simon Jin, Katherine Johnson, Dalia Sulieman, and Wei Wei. 2021. *Data Science Curriculum Design: A Case Study.* Association for Computing Machinery, New York, NY, USA, 529–534. https://doi.org/10.1145/3408877.3432443

[4] Dianne P. Bills and Roxanne L. Canosa. 2007. Sharing Introductory Programming Curriculum across Disciplines. In *Proceedings of the 8th ACM SIGITE Conference on Information Technology Education* (Destin, Florida, USA) *(SIGITE '07)*. Association for Computing Machinery, New York, NY, USA, 99–106. https://doi.org/10.1145/1324302.1324324

[5] Jean R. S. Blair, Christa M. Chewar, Rajendra K. Raj, and Edward Sobiesk. 2020. Infusing Principles and Practices for Secure Computing Throughout an Undergraduate Computer Science Curriculum. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (Trondheim, Norway) *(ITiCSE '20)*. Association for Computing Machinery, New York, NY, USA, 82–88. https://doi.org/10.1145/3341525.3387426

[6] Deborah Boisvert, Robert Cohen, Oscar Gutierrez, and Joyce LaTulippe. 2006. Achieving a Regional Process for Curriculum Development. In *Proceedings of the 7th Conference on Information Technology Education* (Minneapolis, Minnesota, USA) *(SIGITE '06)*. Association for Computing Machinery, New York, NY, USA, 59–64. https://doi.org/10.1145/1168812.1168829

[7] James P. Cohoon, J. McGrath Cohoon, and Mary Lou Soffa. 2013. Educating Diverse Computing Students at the University of Virginia. *Computer* 46, 3 (March 2013), 52–55. https://doi.org/10.1109/MC.2013.39

[8] James P. Cohoon and Luther A. Tychonievich. 2011. Analysis of a CS1 Approach for Attracting Diverse and Inexperienced Students to Computing Majors. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (Dallas, TX, USA) *(SIGCSE '11)*. Association for Computing Machinery, New York, NY, USA, 165–170. https://doi.org/10.1145/1953163.1953217

[9] Ben Coleman and Matthew Lang. 2012. Collaboration across the Curriculum: A Disciplined Approach Todeveloping Team Skills. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (Raleigh, North Carolina, USA) *(SIGCSE '12)*. Association for Computing Machinery, New York, NY, USA, 277–282. https://doi.org/10.1145/2157136.2157220

[10] UVA CS Undergraduate Curriculum Committee. 2014. *Computer Science Curricula 2013 Evaluation Report.* The University of Virginia. https://www.cs.virginia.edu/~sherriff/papers/UVACS-CS2013Report.pdf

[11] Sebastian Dziallas and Sally Fincher. 2015. ACM Curriculum Reports: A Pedagogic Perspective. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (Omaha, Nebraska, USA) *(ICER '15)*. Association for Computing Machinery, New York, NY, USA, 81–89. https://doi.org/10.1145/2787622.2787714

[12] CC2020 Task Force. 2020. *Computing Curricula 2020: Paradigms for Global Computing Education.* Association for Computing Machinery, New York, NY, USA.

[13] Martin Fowler. 1999. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley Longman Publishing Co., Inc., USA.

[14] Mikey Goldweber, Joyce Currie Little, Gerry Cross, Renzo Davoli, Charles Riedesel, Brian R. von Konsky, and Henry Walker. 2010. Enhancing the Social Issues Components in Our Computing Curriculum: Computing for the Social Good. In *Proceedings of the 2010 ITiCSE Working Group Reports* (Ankara, Turkey) *(ITiCSE-WGR '10)*. Association for Computing Machinery, New York, NY, USA, 117–133. https://doi.org/10.1145/1971681.1988996

[15] Jim Hoag. 2013. Evolution of a Cybersecurity Curriculum. In *Proceedings of the 2013 on InfoSecCD '13: Information Security Curriculum Development Conference* (Kennesaw GA, USA) *(InfoSecCD '13)*. Association for Computing Machinery, New York, NY, USA, 94–99. https://doi.org/10.1145/2528908.2528925

[16] Ivar Jacobson, Magnus Christerson, Patrik Jonsson, and Gunnar Övergaard. 1992. *Object-Oriented Software Engineering: A Use Case Driven Approach.* ACM Press. Addison–Wesley, USA, 69–70.

[17] Reza Kamali, Samuel Liles, Charles Winer, Keyuan Jiang, and Barbara Nicolai. 2005. An Implementation of the SIGITE Model Curriculum. In *Proceedings of the 6th Conference on Information Technology Education* (Newark, NJ, USA) *(SIGITE '05)*. Association for Computing Machinery, New York, NY, USA, 15–17. https://doi.org/10.1145/1095714.1095720

[18] M. R. K. Krishna Rao, S. Junaidu, T. Maghrabi, M. Shafique, M. Ahmed, and K. Faisal. 2005. Principles of Curriculum Design and Revision: A Case Study in Implementing Computing Curricula CC2001. *SIGCSE Bull.* 37, 3 (June 2005), 256–260. https://doi.org/10.1145/1151954.1067515

[19] M. Manny Lehman and Les A. Belady. 1985. *Program evolution: processes of software change.* Academic Press, USA.

[20] Paul Leidig, Michael Goldweber, and Barbara Boucher Owens. 2012. Assessing the Benefits of Integrating Social Issues Components in the Computing Curriculum. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education* (Haifa, Israel) *(ITiCSE '12)*. Association for Computing Machinery, New York, NY, USA, 367–368. https://doi.org/10.1145/2325296.2325382

[21] Joyce Currie Little, Richard H. Austing, Harice Seeds, John Maniotes, and Gerald L. Engel. 1977. Curriculum Recommendations and Guidelines for the Community and Junior College Career Program in Computer Programming: A Working Paper of the Association for Computing Machinery Committee on Curriculum in Computer Sciences by the Sub Committee on Community and Junior College Curriculum. *SIGCSE Bull.* 9, 2 (June 1977), 17–36. https://doi.org/10.1145/988948.988951

[22] The University of Virginia Registrar. 2021. *Computer Engineering.* The University of Virginia. http://records.ureg.virginia.edu/preview_program.php?catoid=52&poid=6751

[23] The University of Virginia Registrar. 2021. *Computer Science (B.S.).* The University of Virginia. http://records.ureg.virginia.edu/preview_program.php?catoid=52&poid=6753

[24] The University of Virginia Registrar. 2021. *Interdisciplinary Major in Computer Science.* The University of Virginia. http://records.ureg.virginia.edu/preview_program.php?catoid=52&poid=6752

[25] ACM/IEEE-CS Joint Task Group on Computer Engineering Curricula. 2016. *Computer Engineering Curricula 2016: Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering.* ACM Press and IEEE Computer Society Press, New York, NY, USA.

[26] ACM/IEEE-CS Joint Task Force on Computing Curricula. 2015. *Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering.* ACM Press and IEEE Computer Society Press, New York, NY, USA.

[27] Joint Task Force on Cybersecurity Education. 2018. *Cybersecurity Curricula 2017: Curriculum Guidelines for Post-Secondary Degree Programs in Cybersecurity.* Association for Computing Machinery, New York, NY, USA.

[28] Krassie Petrova, Anne Philpott, Petteri Kaskenpalo, and Jim Buchan. 2004. Embedding Information Security Curricula in Existing Programmes. In *Proceedings of the 1st Annual Conference on Information Security Curriculum Development* (Kennesaw, Georgia) *(InfoSecCD '04)*. Association for Computing Machinery, New York, NY, USA, 20–29. https://doi.org/10.1145/1059524.1059529

[29] Charles W. Reynolds. 2006. Engineering the Information Technology Curriculum with Pervasive Themes. In *Proceedings of the 7th Conference on Information Technology Education* (Minneapolis, Minnesota, USA) *(SIGITE '06)*. Association for Computing Machinery, New York, NY, USA, 141–148. https://doi.org/10.1145/1168812.1168847

[30] Takayuki Sekiya, Yoshitatsu Matsuda, and Kazunori Yamaguchi. 2015. Curriculum Analysis of CS Departments Based on CS2013 by Simplified, Supervised LDA. In *Proceedings of the Fifth International Conference on Learning Analytics And Knowledge* (Poughkeepsie, New York) *(LAK '15)*. Association for Computing Machinery, New York, NY, USA, 330–339. https://doi.org/10.1145/2723576.2723594

[31] Blair Taylor and Shiva Azadegan. 2006. Threading Secure Coding Principles and Risk Analysis into the Undergraduate Computer Science and Information Systems Curriculum. In *Proceedings of the 3rd Annual Conference on Information Security Curriculum Development* (Kennesaw, Georgia) *(InfoSecCD '06)*. Association for Computing Machinery, New York, NY, USA, 24–29. https://doi.org/10.1145/1231047.1231053

[32] Allen B. Tucker. 1991. Computing Curricula 1991. *Commun. ACM* 34, 6 (June 1991), 68–84. https://doi.org/10.1145/103701.103710

[33] Luther A. Tychonievich. 2017. Training Course for Teaching Assistants in Computing. https://www.cs.virginia.edu/luther/ta-training. Accessed 2021-07-29.

[34] Jaideep Vaidya, Basit Shafiq, David Lorenzi, and Nazia Badar. 2013. Incorporating Privacy into the Undergraduate Curriculum. In *Proceedings of the 2013 on InfoSecCD '13: Information Security Curriculum Development Conference* (Kennesaw GA, USA) *(InfoSecCD '13)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/2528908.2528918