Does Reducing Curricular Complexity Impact Student Success in Computer Science?

Sumukhi Ganesan
Khoury College of Computer Sciences
Northeastern University
Boston, MA, USA
ganesan.su@northeastern.edu

Albert Lionelle
Khoury College of Computer Sciences
Northeastern University
Boston, MA, USA
a.lionelle@northeastern.edu

Catherine Gill
Center for Inclusive Computing
Northeastern University
Boston, MA, USA
c.gille@northeastern.edu

Carla Brodley
Center for Inclusive Computing
Northeastern University
Boston, MA, USA
c.brodley@northeastern.edu

Abstract

Computer science degree requirements often have a rigid pre- and corequisite structure, which can impede a student's progression through a degree and in particular can add one or more semesters to time to completion, particularly for those students who need to retake a course that serves as a prerequisite to other courses and for students who are not calculus-ready when they enter university. In this paper, we present the results of a comparative analysis of curricula before and after a major structural revision. The first curriculum adheres to the conventional, rigid prerequisite structure, while the second emphasizes student choice and multiple pathways through the degree. No changes were made to course content/outcomes between the two versions. Employing curricular metrics such as complexity and centrality, we examine the degree progress of 3010 students over a six-year period. Specifically, our investigation looks at the impact of reducing curricular complexity on student attrition from and attraction to the CS major. The new curriculum, with a 60% reduction in curricular structural complexity, showed both increased retention of students over the old curriculum (67% to 98%) and an increase in the number of students converting from undeclared to computer science (44% to 69%). Our findings demonstrate that reducing curricular complexity need not compromise program rigor and can benefit students by providing greater flexibility and ensuring earlier exposure to (and therefore retention in) CS.

CCS Concepts

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

Keywords

Computing education, Curriculum design, Retention, BPC

SIGCSE TS 2025, February 26-March 1, 2025, Pittsburgh, PA, USA

ACM ISBN 979-8-4007-0531-1/25/02
https://doi.org/10.1145/3641554.3701915

ACM Reference Format:
Sumukhi Ganesan, Albert Lionelle, Catherine Gill, and Carla Brodley. 2025. Does Reducing Curricular Complexity Impact Student Success in Computer Science?. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE TS 2025), February 26-March 1, 2025, Pittsburgh, PA, USA*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3641554.3701915

1 Introduction

Computer science degree requirements often have a rigid pre- and corequisite structure, designed with the goal of ensuring that students have a strong foundation in not only within-discipline introductory material but also in the mathematics topics that are required for many upper-level CS courses. However a rigid pre- and corequisite structure can impede a student's progression through a degree and in particular can add one or more semesters to time to completion, particularly for those students who need to retake a course that serves as a prerequisite to other courses and for students who are not calculus-ready when they enter university [2, 15]. Indeed, the ACM 2023 Curricular Guidelines [14] encourage schools to be more flexible in the curricular pathways they offer, including being more intentional about which pre- and corequisite classes are really needed.

In this paper, we study the impact on student attraction/retention of reducing the curricular complexity of the Bachelor of Science in CS degree (BSCS) at Colorado State. University.[1] We analyze the change in attraction and retention of students in the BSCS before and after the structural revision of the degree requirements. Note that the overall number of computing courses did not decrease (indeed the new curriculum added 2 additional required CS courses and one optional CS course) and there were no significant changes in the actual course content and course outcomes for the required and elective courses (other than the routine improvements that are part of the normal work of a professor in teaching a course over multiple semesters). Instead, what changed was the addition of multiple pathways through the degree, and a reduction in prerequisites to ensure each CS course required only those CS and math courses whose content was needed for *that* CS course. We detail these changes in Section 3.

[1] Albert Lionelle was a member of the faculty at CSU during the redesign.

To analyze the structural complexity of the "old" and "new" versions of the degree requirements, we use the metrics developed in the field of *Curricular Analytics*, which facilitates the study of degree structure and how it impacts student success [8–11, 16–21]. This paper adds to this growing body of research by asking: does curricular structure influence the attrition of current students and attraction of new students? Our findings show that the new, more flexible curricular design has both a lower attrition rate of declared CS majors and a higher attraction rate of undeclared students into the program. With the changes taking place only to the prerequisite chain and *not to any course content or learning outcomes*, the analysis indicates that it is possible to decrease curricular complexity (and improve retention and attraction) without decreasing the rigor and success of the program.

In the remainder of this paper, we first review existing research on curricular design in Section 2. We next provide the old and the new degree requirements with an analysis of their curricular complexity in Section 3. In Section 4 we present our research methodology and results including threats to validity. In Section 5 and 6, we discuss the takeaways for computing departments and our conclusions.

2 Related Work

There is a long history in the computer science education research community of studying and revising curricular guidelines as the field continues to evolve rapidly. In addition to CS education researchers, there are two professional organizations ABET-CAC [4] and the ACM [1] that provide curricular recommendations and guidelines for CS. Neither ABET-CAC nor ACM-2023 [14] talk in depth about the curricular structure of the degree requirements although in 2023, the ACM does suggest that calculus can be delayed.

In this section, we focus on research addressing the *curricular structure* of CS degree requirements, particularly the pre- and corequisite structure. Research has been done at the individual course level, and also at examining the entire structure of the degree requirements. Klingbeil and Bourne [13] presented results for a change to the prerequisite structure of an engineering program, which made calculus optional for first-year students. This adjustment yielded favorable outcomes, including increased graduation rates and higher GPAs, particularly among students from historically marginalized backgrounds. Similarly, a recent study by Das et al., [5], shows that removing such prerequisites had no impact on student performance in CS1 and additionally allowed more students to take CS classes earlier in their programs.

In the last decade, there has been a new focus on curricular structure of the entire degree program. Wigdhal et al., [21] introduced directed graphs for comparing degree programs, inspired by the curricular network analysis framework established by Slim et al., [19]. This collaborative effort explored metrics such as total credits, mandatory courses, delays, bottlenecks, and pass/fail rates. Heileman et al., [10] further expanded on this methodology, constructing degree maps using directed graphs to evaluate measures like delay factor, blocking factor, centrality, and program complexity, which we explain in detail next, as they will form the core measures of the CS curriculum before and after the structural revision.

A course's **delay factor** measures the longest prerequisite chain to which a course belongs. For example, if the course is the third course in a chain of six courses which is the longest chain of courses to which it belongs, its delay factor is six. The longest delay across a curriculum also represents the shortest time to graduate due to prerequisite blocks and limitations. A course's **blocking factor** measures the extent to which the course blocks a student's ability to take other courses in the curriculum and is defined by the number of courses reachable from the course node in the map. **Centrality** is the sum of the delay of all the paths containing that course. If a course has no pre or post-requisites, it's centrality is zero. Course **complexity** is a measure of the overall complexity rating for a course, which is calculated as an unweighted linear combination of its delay and blocking factors. **Program complexity** is the sum of all individual course complexities and thus acts as a single measure that can be used to compare different curricula. Heileman et al. argue that these metrics serve as valuable indicators for guiding program reform and simplification.

Slim et al., [17] investigated the relationship among course sequencing, semester complexity (the summation of course complexities taken in that semester), and student outcomes. Their findings indicated that students facing more complex semesters often experienced delays and lower GPAs, whereas those with work-load balanced course sequences tended to achieve higher GPAs. Subsequent studies by Slim et al., [16, 20] applied predictive modeling techniques to forecast graduation rates and student success based on curricular complexity. Heileman et al., [9–11] correlated program rankings, although problematic in their ways [3, 6, 7], with curricular complexity, revealing that higher-ranked electrical engineering and CS programs tend to have lower complexity. Conversely, lower-ranked programs appear to rely more heavily on high curricular complexity as a means of student preparation and program enforcement.Additionally, recent studies conducted by Lionelle et al., [15] and Jiang et al., [12], comparing CS programs using the above curricular metrics, have also shown that lower curricular complexity increases participation by women and transfer students in CS, respectively.

3 Curricula

To understand the changes made we describe both the old and the new curriculum (course names have been generalized). The old curriculum was built from the ACM guidelines and had been stable for twenty years. It emphasized a rigid course structure for two purposes: (1) to follow traditionally suggested guidelines on what is needed for courses especially in topics of mathematical maturity and to this end, required students to complete all "core" lower division courses before students could take any upper-division courses, and (2) to enforce *when* a course was taken in the program sequence. For example, because calculus II was a prerequisite for algorithms (traditionally the first upper-division course taken) the requirements specified that it must be taken before the third year.

The curricular maps for the old and new curriculum are shown in Figures 1 and 2, respectively. In Section 4 we discuss assumptions made in the construction of these maps. In each map, we have highlighted the longest pre-requisite chain through the major and highlighted the metrics for CS 2 (data structures).

The old curriculum's prerequisite structure created a web of prerequisites with multiple bottlenecks. It also assumed students came into the program calculus-ready. The core of the old curriculum is

Table 1: Curricular Requirements Old (2017) and New (2021).

	Old Curriculum	#/Credits	New Curriculum	#/Credits
Core Lower Division Courses	CS 1, CS 2, Discrete Math, Computer Organization Software Development	5 courses 20 credits	CS 1, CS 2, Discrete Math, Computer Organization Software Development Ethics (GE) Optional: CS 0 (GE)	6-7 courses 23-26 credits 3-6 credits overlap with general university requirements (GE)
Core Upper Division Courses	Algorithms Operating Systems Security Software Engineering 4 'pick list' CS courses	8 courses 28 credits	Algorithms Operating Systems Software Engineering 6 'pick list' CS courses	9 courses 31-32 credits
Technical Electives	Courses related to CS, but not CS (Math, Stats, Data Science, Business, Etc)	3 courses 9 credits	Added following to old list: Calc II, all upper-division CS, more psychology courses, design thinking course	2 courses 6 credits
Math Requirements	Calc I, Calc II, Statistics/Probability, Linear Algebra	4 courses 13-15 credits	Calc I, Statistics/Probabilty, Linear Algebra	3 courses 11 credits
Natural Sciences	12 credits, two lab courses 7 credits overlap with GE	3-4 courses 12 credits	7 credits, one lab course 7 credits overlap with GE	2 courses 7 credits
Totals		23-24 courses 82-85 credits		25-26 courses 78-82 credits

Note: GE courses and electives fill out the remainder of the credits for 120 credits total.

detailed in the second column of Table 1. Notable characteristics are that in order to take CS 1, students had to be calculus-ready, but in practice only 46% were calculus-ready, thus many students were delayed in taking CS 1 until their second semester or even their third semester. Indeed, calculus I was a prerequisite to both CS 2 and discrete structures. Further calculus II, linear algebra, and discrete structures were prerequisites to algorithms. Thus if a student was not calculus ready (or if they failed a math class), it would significantly delay their progression in the CS curriculum. Similarly, if they failed any of the core CS classes they were delayed by a semester before reaching the upper-division CS classes.

When building the new curriculum, the following priorities were set: create pathways throughout the program to reduce bottlenecks thus allowing more flexibility in when students took specific courses, *increase* the number of options especially with CS elective courses, and *increase* the number of CS courses students could complete. These changes significantly reduced the structural complexity of the degree without compromising its rigor. Note that the course content, outcomes, and exams remained the same, except to avoid cheating. CS 0 was added as an optional course for students who wish to start with a gentler introduction to CS and for students who did not meet the math pre-requisites for CS 1 which require most concepts from pre-calc. This means that students who start as CS majors and are calculus-ready can start with either CS 0 or CS 1 depending on their comfort level. And, students who are not calculus-ready can still take a programming course in their first semester. CS 0 was also coded to fulfill an arts and humanities requirement for the university. Calculus I was moved to be a pre-req to algorithms and calculus II was moved into the technical elective options. Prior to this change, a new linear algebra course (focusing on computational sciences) was developed by the math department that no longer required calculus II. The new curriculum also greatly simplified the prerequisites to CS 2 (data structures), discrete structures, and computer organization.

4 Results

This case study addresses the research question "Does curricular design influence student success in CS?" We break down this study into three parts. First, we analyze the old and new curriculum to assess the changes in curricular complexity as described in Section 3. Second, we analyze changes in student attrition rates from the old and the new curricula. Third, we examine the journeys of undeclared majors discovering CS and ultimately the changes in the attraction rates between the new and old curricula.

Building Curricular Maps: To build the curricular maps, we used the tools provided by CurricularAnalytics.org and the publicly available information presented on the university's website. For both the old and new curricula, we created an "optimized" four-year plan. This optimized plan includes all required courses for the major, general education requirements, and electives to achieve the university's 120-credit minimum for graduation. The university follows a semester-based program with fall, spring, and summer terms. For each required upper-division CS elective, we selected courses with a bias toward those courses that do not have additional prerequisites (e.g., we avoided courses like computer graphics which have additional math requirements because our analysis focuses on the "quickest path to graduation" plan of study). We further assumed students did not come in with any AP credits.

A complicating factor in our analysis was that we uncovered multiple "hidden" required courses due to the - largely false - assumption that students are calculus-ready in the first semester. We found that 46% of all students in the old curriculum were not calculus-ready and had to enroll in at least one prerequisite course before CS1. Thus, our version of the map for the old curriculum *explicitly* includes these hidden prerequisite courses.

To compare the curricular maps, we used the measures developed by Heileman et al., [8] In particular, we looked at the **structural complexity** of both courses and degree plans, and the **centrality** of courses (see Section 2 for definitions). Figure 1 shows the old curriculum and Figure 2 shows the new curriculum.

The structural complexity of the old curriculum with the hidden prerequisites was 415 (without it was 208). The complexity of the new curricular design is 164. We also observed a reduction in centrality (as prerequisite chains were decoupled), meaning many courses exist as part of a single chain instead of multiple chains. The increased flexibility reduced bottlenecks, and effectively allowed students to delay calculus I until the third semester, which gave time for students who needed to take the prerequisites to calculus.

Student Data: For the Fall 2017 - Spring 2024 period, we collected de-identified data for every undergraduate student taking math or computer science courses at the university. Data for each student included the term, courses taken, grades received, student's declared major in that term, and if they were a CS major when they completed their degree. Our data included 40,869 unique students over the six years. Note that we collected the data for students taking math because some of the students who will become CS majors are either majoring in another field or belong to a special subset of students who have an initial major designation named "Undeclared Seeking Computer Science (USCS)".

We divided students into two groups: those admitted prior to Fall 2021 (old curriculum) and those admitted from Fall 2021 onward (new curriculum). These groups consisted of 26,255 and 14,614 students respectively. Using the student's major for the term, we identified students who had declared CS as their major at least for a term and have either a) subsequently changed their major in a later term or b) stopped taking CS or math courses overall, potentially

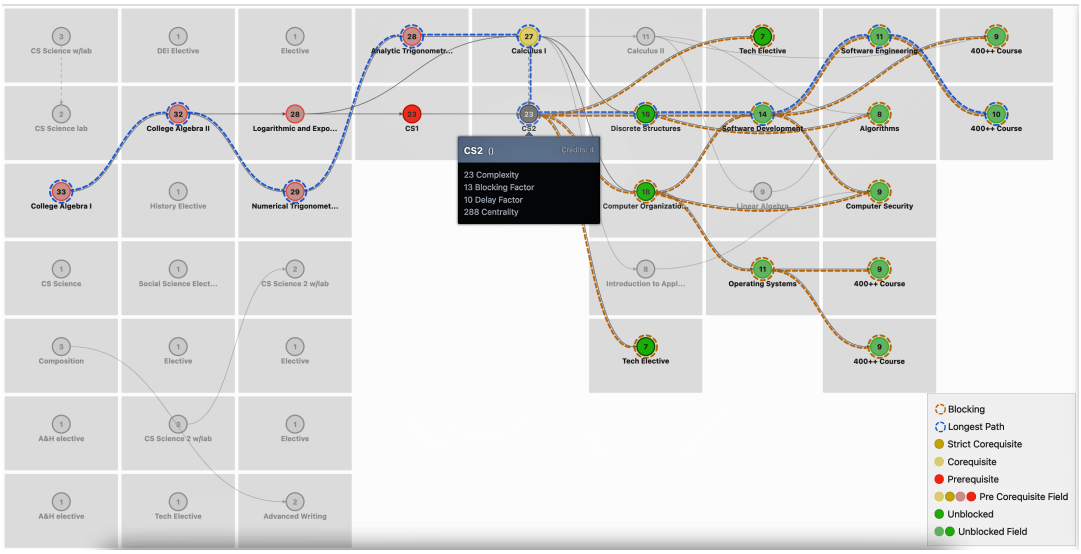

Figure 1: Degree Map of the Original Curriculum including Hidden Prerequisites (Complexity = 415).

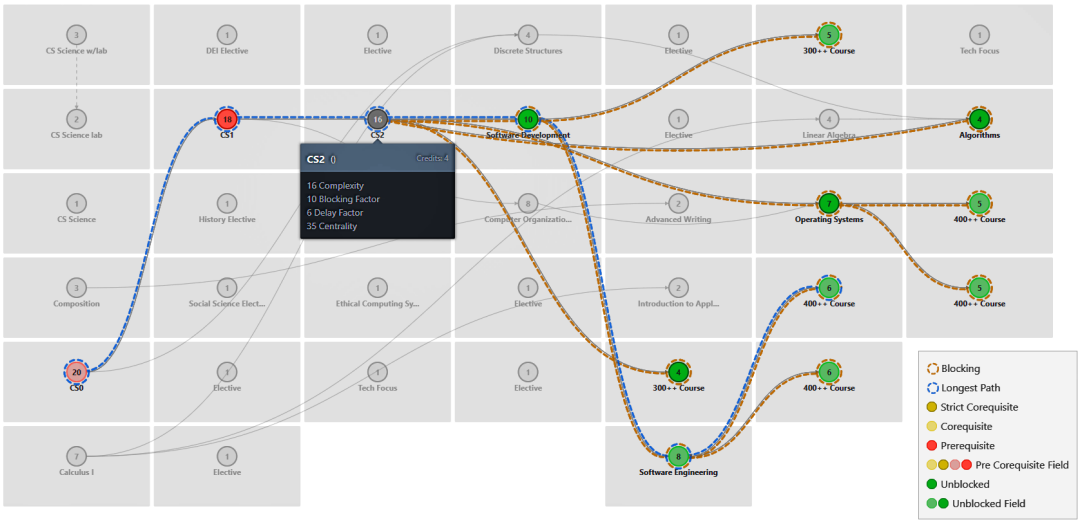

Figure 2: Degree Map of the New Curriculum (Complexity = 164).

indicating that the student has paused their studies or even left the university. We count the number of students identified using this criteria as attrition from the program.

Student Attrition and Terminal Courses: First examining the data for students who enrolled under the old curriculum, we identified those students who had declared as a CS major, but dropped out of the program before taking a single CS course. This analysis is what led us to "uncover" the hidden prerequisite structures not previously included in the published degree plan. We found 26 CS majors (4% of the total attrition) who left the major having never taken a computer science course. Furthermore, 118 CS majors (18%) withdrew with at least one unsatisfactory grade in a pre-calculus course during their last term as a declared CS major. If we look at the math courses in aggregate (pre-calculus, calculus I, calculus II, and linear algebra) we find that 282 (45%) of the CS majors who left CS took a math course in their last semester as a CS major.

To better understand the influence of course structure on student attrition, we track the students' journeys to see which courses they took before they left the program. We refer to the final course a student takes as their **"terminal course"**. Figure 3 provides the analysis of the top 10 "terminal courses" (the last math or CS class taken by a student who left the major) of the old curriculum, which includes the hidden courses. For some students, there were multiple courses that were terminal. In those cases, we counted each terminal course equally in our calculations, as there wasn't a way to determine if there was a particular course that led to the student's attrition. For example, if a student took CS 2, computer organization, and discrete structures in their final semester, all three of the courses counted toward the student's decision to leave. In Figure 3 we list each course's complexity and centrality inside its histogram

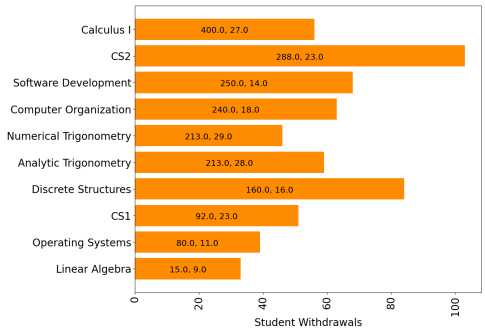

Figure 3: Terminal Courses in Original Curriculum.

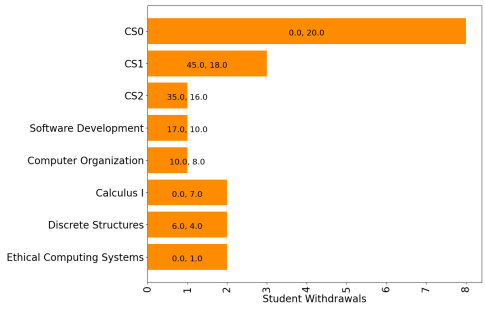

Figure 4: Terminal Courses in the New Curriculum.

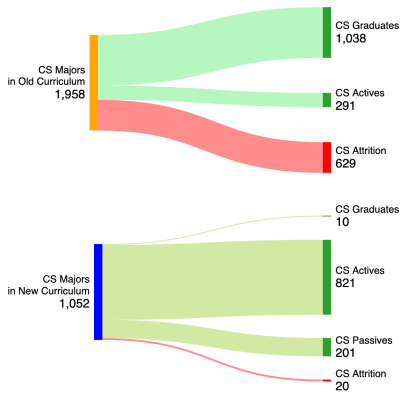

Figure 5: Student Journeys in the CS program

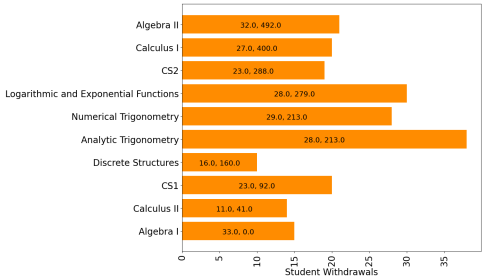

Figure 6: Terminal Courses for USCS in old curriculum.

bar, and the courses are ordered from highest to lowest centrality in the graph. We see that calculus, CS 2 and discrete structures are in the top ten terminal courses, with CS 2 having the highest number of withdrawals. Note that 33% of students repeat a terminal course one or more times before withdrawing. To confirm the relationship between centrality and retention, we ran a simple correlation test and showed a 0.806 correlation, indicating a strong relationship.

Figure 4 shows the top ten terminal courses of the new curriculum. Note that the scale has changed from 0-100 in Figure 3 to 0-8 in Figure 4. Fewer students leave the major once they have declared under the new curriculum compared to the old curriculum. Indeed the terminal course with the largest number of CS majors leaving is the new CS 0 which was explicitly designed to provide students an introduction to computer science, so they could make an early decision as to whether it was a good major for them. While there are still a few withdrawals with terminal math courses, the net impact of math on CS withdrawals has reduced significantly.

Figure 5 illustrates the different journeys of CS majors. Students who are still taking CS courses in Fall 2023 ("CS Actives") or are still enrolled in the CS program without taking courses ("CS Passives") are excluded from the attrition calculation as we assume that they are still engaged with the program. The limitations of this assumption are discussed below. In the period studied in which the old curriculum was in place, the overall program retention of CS majors (i.e., they declared and remained declared as computer science majors during the time window) was **67%**. In the redesigned curriculum, the retention across the period studied was **98%**.

Attraction: A group of particular importance outside of CS majors are students who are "Undeclared Seeking Computer Science (USCS)," which is a specific major at the university that comprises students who are either interested in CS but not ready to declare, or are interested but do not yet meet the major controls.[2] The question we pursue next is whether the curricular changes influence the attraction of USCS students to the major.

The data indicates that math plays an important role in the potential conversion of USCS students to CS majors. Figure 6 details the top 10 terminal courses that USCS students were taking in the old curriculum: 7 of the 10 are math courses. Indeed, 55% of USCS students were enrolled in one or more math courses in the semester when they made the decision *not* to pursue CS as a major.

A major benefit of the new curriculum is the emphasis it places on pathways to accommodate students with different levels of math preparation. In the old curriculum, the majority of USCS students chose another major outside of computer science or dropped out of the university completely with only 47% (135 out of 307) choosing to major in CS. In the new curriculum, 69% (149 out of 215) chose to major in CS or are still taking a CS course (meaning they could still be working toward the major requirements). A chi-square test of independence was performed to examine the relationship between the curricula and the decision to major in CS. The relationship between these variables was significant, $x^2(1, 522) = 32.7, p < 0.00001$. In sum, the new curriculum - with its reduction in curricular complexity - has not only higher retention of declared majors, but also attracts more students from the USCS population.

[2]The CS major controls require students to either declare the CS major when they arrive from high school or to pass CS 1 with a C or higher and calculus I with a C or higher while maintaining an overall GPA of at least 2.5.

Threats To Validity: There are three threats to the results presented. First, the Curricular Analytics tool examines the complexity of a single path through a degree, but, in truth, degrees will always have multiple paths and starting points. For example, students could come in with some college credit, transfer from a community college, or come from other majors. The curricular map and our comparison to the top terminal courses do not account for these differences. Regardless, the high centrality and complexity of introductory courses hold the strongest influence on student retention. The second threat stems from when the new curriculum was introduced; having only two years of student data means that the data for the new curriculum is smaller than that for the old curriculum and further that many students have not yet graduated. We mitigate this risk by looking at sliding windows for both the old and new curriculum and determined that the first two years in any program are most integral for student retention. Stated another way, students at our university rarely withdraw from a program after they complete the first two years. Indeed, for both the old and the new curriculum none of the top-ten terminal courses were upper-division (junior or senior level) courses. Third, we have no way to study students who were not already declared CS majors or declared USCS majors. Thus more students might have wanted to switch their major to CS and been discouraged/encouraged in either the old or the new curriculum and we have no good way to measure this attrition and attraction.

5 Discussion

The analysis of the differences in outcomes in the different curricular structures shows that a rigid curriculum is not required for students to meet the learning outcomes and be successful in the major. Through this analysis, we arrive at a set of recommendations that could be applied to any degree redesign. To form the new curriculum, faculty only specified a course as a prerequisite to another course X only of the content was essential to the *majority of course X*. In cases where a prerequisite course was only used for a small portion, the premise was that the topic could be covered in the class itself. An example was the algorithms, where the new curriculum no longer requires calculus II as a prerequisite (it had been placed as one to ensure that students took calculus II by the end of year 2). As another example, a number of courses required both algorithms and operating systems as prerequisites, not because the content itself was needed, but rather as a gatekeeping mechanism to force students to take the full set of core CS courses before advancing to upper-division electives. The evaluation of these situations led faculty to a more thoughtful either/or model where students have more than one prerequisite option for upper division course options.

Additionally, significant effort went into laying out the different pathways through the new curriculum. For example, most courses involving a higher level of mathematical competency such as algorithms and machine learning, were placed in one of the pathways. Other courses such as software engineering and computing systems were decoupled into their own pathways. While majors in computer science have to take courses from all three pathways as part of their core requirements, the pathway model allowed students to progress even if they were struggling with (or didn't enjoy) a certain course or area of the curriculum. This has the effect of reducing anxiety that is often caused by choke points in a curriculum. With the new design, the only two courses that remained "choke points" were CS 1 and CS 2, arguably essential prerequisites for nearly all CS courses. Even in those cases, a student could still take discrete structures and/or computer organization at the same time as CS 2.

A noticeable difference between the old and new curricula was the addition of CS 0, which provides a pathway into the major for students who need remedial math. The new curriculum gives students the choice to focus on programming and progress in their degree while they work on their math. Given that nearly half of all students coming to the university needed at least one of the remedial math courses, this broadened access to the major. It also allows students who did not enjoy computing an early opportunity to discover that and chart their next steps accordingly. The mathematics barrier cannot be overstated, and while math does have importance in computer science, it should not be a gateway into the program. In our design of the new currilum we focused on ensuring that students don't have to take calculus until it is needed. This change was one of the most powerful as it gives students time to work through the remedial math prerequisites without hindering progress in the degree.

Our third recommendation is to use the curricular complexity tools proactively. We employed them retroactively to analyze and quantify the level of change that was achieved in the curricular redesign, but suggest that departments that want to address systemic barriers that may be contributing to student retention (and lack of attraction) can use them diagnostically to determine where choke points and other challenges exist. Then, if a school decides to engage in a curricular redesign process, they can use the tools to test and scenario plan, mapping out different approaches and pressure testing those options for different student profiles (calc ready, not calc ready, external transfer, internal transfer, etc).

6 Conclusions and Future Work

In this case study, we examine the changes made to streamline the curricular structure of the BS in CS degree program. By purposefully considering the need to have clear pathways and options for all students, and by placing prerequisites only where absolutely necessary for student learning, the college achieved a reduction in structural complexity, with a resulting decrease in attrition and an increase in attraction to the CS major. An area for future work is to look at the data disaggregated by gender and race/ethnicity.

Importantly, this structural transformation did not compromise academic rigor or learning outcomes. The learning objectives and goals of the courses remained consistent throughout the transition. What the structural changes did achieve was an environment in which students with different levels of prior CS exposure and different levels of mathematics all have a clear starting point and pathways into the degree. Above all, the redesign no longer requires students to spend time building their math skills before starting their computer science courses.

Acknowledgments

This project was partially funded by the CIC at Northeastern University. We would like to thank Colorado State University faculty and staff who did the work on designing the new degree requirements and Heather Lionelle for editing and reviewing this paper.

References

[1] Association for Computing Machinery. 2024. Advancing Computing as a Science & Profession. http://www.acm.org/ Accessed: 2024-07-16.

[2] Carla E. Brodley, McKenna Quam, and Mark A. Weiss. 2024. An Analysis of the Math Requirements of 199 CS BS/BA Degrees at 158 U.S. Universities. arXiv:2404.15177 [cs.CY] https://arxiv.org/abs/2404.15177

[3] John Byrne. 2018. How much attention should you pay to U.S. News' College Rankings? *Forbes* (Sep 2018). https://www.forbes.com/sites/poetsandquants/2018/09/10/how-much-attention-should-you-pay-to-u-s-news-college-rankings/?sh=320d4c667daf

[4] Computing Accreditation Commission, ABET. 2024. Accreditation Criteria and Supporting Documents. https://www.abet.org/accreditation/accreditation-criteria/cac-criteria/ Accessed: 2024-07-16.

[5] Udayan Das and Chris Fulton. 2024. Reducing Barriers to Entry by Removing Prerequisites for a CS1 Introductory Programming Course. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 1616–1617. https://doi.org/10.1145/3626253.3635492

[6] Robert Farrington. 2022. How much should you trust college rankings? *Forbes* (Sep 2022). https://www.forbes.com/sites/robertfarrington/2022/09/29/how-much-should-you-trust-college-rankings/?sh=c6cd8917f01a

[7] Malcom Gladwell. 2011. The Order of Things What college rankings really tell us. *New Yorker* (Feb 2011). https://www.newyorker.com/magazine/2011/02/14/the-order-of-things

[8] Gregory L. Heileman, Chaouki T. Abdallah, Ahmad Slim, and Michael Hickman. 2018. Curricular analytics: A framework for quantifying the impact of curricular reforms and pedagogical innovations. (Nov 2018). arXiv:1811.09676 https://arxiv.org/abs/1811.09676v1

[9] Gregory L. Heileman, Hayden W. Free, Johnny Flynn, Camden Mackowiak, Jerzy W. Jaromczyk, and Chaouki T. Abdallah. 2020. Curricular complexity versus quality of computer science programs. (Jun 2020). arXiv:2006.06761 https://arxiv.org/abs/2006.06761v1

[10] Gregory L. Heileman, Ahmad Slim, Michael Hickman, and Chaouki T. Abdallah. 2017. Characterizing the complexity of curricular patterns in engineering programs. In *ASEE Annu. Conf. Expo. Conf. Proc.*, Vol. 2017-June. American Society for Engineering Education. https://doi.org/10.18260/1-2--28029

[11] Gregory L. Heileman, William G. Thompson-Arjona, Orhan Abar, and Hayden W. Free. 2019. Does curricular complexity imply program quality? *ASEE Annu. Conf. Expo. Conf. Proc.* (Jun 2019). https://doi.org/10.18260/1-2--32677

[12] Jinya Jiang, Richa Kafle, Christa Lehr, Simone Wright, Clarissa Guitierrez-Godoy, and Christine Alvarado. 2024. Understanding California's Computer Science Transfer Pathways. In *Proc. 55th ACM Tech. Symp. Comput. Sci. Educ. V. 1 (SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 604–610. https://doi.org/10.1145/3626252.3630956

[13] Nathan Klingbeil and Anthony Bourne. 2015. The Wright State model for engineering mathematics education: Longitudinal impact on initially underprepared students. (Jul 2015), 26.1580.1–26.1580.11. https://doi.org/10.18260/P.24917

[14] Amruth N. Kumar, Rajendra K. Raj, Sherif G. Aly, Monica D. Anderson, Brett A. Becker, Richard L. Blumenthal, Eric Eaton, Susan L. Epstein, Michael Goldweber, Pankaj Jalote, Douglas Lea, Michael Oudshoorn, Marcelo Pias, Susan Reiser, Christian Servin, Rahul Simha, Titus Winters, and Qiao Xiang. 2024. *Computer Science Curricula 2023*. Association for Computing Machinery, New York, NY, USA.

[15] Albert Lionelle, McKenna Quam, Carla Brodley, and Catherine Gill. 2024. Does Curricular Complexity in Computer Science Influence the Representation of Women CS Graduates?. In *Proc. 55th ACM Tech. Symp. Comput. Sci. Educ. V. 1 (SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3626252.3630835

[16] Ahmad Slim, Gregory L. Heileman, Chaouki T. Abdallah, Ameer Slim, and Najem Sirhan. 2021. *Restructuring curricular patterns using Bayesian networks*. Technical Report. Online.

[17] Ahmad Slim, Gregory L. Heileman, Wisam Al-Doroubi, and Chaouki T. Abdallah. 2016. The impact of course enrollment sequences on student success. *Proc. - Int. Conf. Adv. Inf. Netw. Appl. AINA* 2016-May (may 2016), 59–65. https://doi.org/10.1109/AINA.2016.140

[18] Ahmad Slim, Jarred Kozlick, Gregory L. Heileman, and Chaouki T. Abdallah. 2014. The complexity of university curricula according to course cruciality. *Proc. - 2014 8th Int. Conf. Complex, Intell. Softw. Intensive Syst. CISIS 2014* (Oct 2014), 242–248. https://doi.org/10.1109/CISIS.2014.34

[19] Ahmad Slim, Jarred Kozlick, Gregory L. Heileman, Jeff Wigdahl, and Chaouki T. Abdallah. 2014. Network analysis of university courses. *WWW 2014 Companion - Proc. 23rd Int. Conf. World Wide Web* (Apr 2014), 713–718. https://doi.org/10.1145/2567948.2579360

[20] Ahmad Slim, Husain Al Yusuf, Nadine Abbas, Chaouki T Abdallah, Gregory L Heileman, and Ameer Slim. 2021. A markov decision processes modeling for curricular analytics. In *2021 20th IEEE Int. Conf. Mach. Learn. Appl.* 415–421. https://doi.org/10.1109/ICMLA52953.2021.00071

[21] Jeffrey Wigdahl, Gregory L. Heileman, Ahmad Slim, and Chaouki T. Abdallah. 2014. Curricular efficiency: What role does it play in student success? *ASEE Annu. Conf. Expo. Conf. Proc.* (2014). https://doi.org/10.18260/1-2--20235