A Curricular Analysis of Computer Science at the University of Virginia

A Research Paper in STS 4600

Presented to the Faculty of the School of Engineering and Applied Science University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements of the Degree Bachelor of Science, School of Engineering

Michael Benos
Spring 2020

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

Advisor
Sean Ferguson, Department of Engineering and Society

Introduction

Computer science (CS) is a highly dynamic field, with new innovations regularly furthering it. One contributing factor to the field's volatility is the field's relative newness as an academic area of study, with Purdue being the first American university to develop a curriculum for it in 1962 (Rice et. al., 2017). Because the field is developing so rapidly, universities are struggling to keep pace in updating their CS curricula. Modifying a curriculum is an extremely lengthy process, often taking several years to get from a series of ideas that drive the reform to implemented change; overhauling a CS program is no different, and this means that by the time that proposed changes are enacted, the new degree is already likely antiquated. Despite this challenge, the University of Virginia (UVa) has recently test driven a new pilot program for the computer science degree with several key goals:

1. To restructure courses to better cover content.
2. To eliminate material duplicated across courses.
3. To better prepare students for internships earlier in their career.
4. To better unite the B.A. and B.S. CS programs.

(Sherriff, UVa CS 2020 Curriculum Pilot).

It is important to note that the UVa CS department claims that the standard curriculum and the pilot program cover the same material. New material is not introduced in the pilot program despite the field's rapid evolution. Additionally, the methods of delivery are remaining constant between the two curricula. This research paper aims to identify the areas where UVa CS is lacking, both materially and methodologically. To accomplish this, the STS theory of actor-network theory (ANT), a framework which aims to understand the relationships of both animate

and inanimate stakeholders in a sociotechnical system's network, will be mapped onto the sociotechnical system of the undergraduate CS program at UVa.

This paper is divided into the following sections: a literature review of ANT in education as well as of an ideal CS curriculum, evidence and analysis regarding how UVa's undergraduate CS program compares to the ideal curriculum explored in the literature review via ANT, and a discussion of the UVa curriculum's future outlook and opportunities for further research in this area. All in all, it is concluded that UVa's CS curriculum, as well as its recently explored changes, is stagnant and is due for more comprehensive reform than what the pilot program proposes.

Literature Review

Actor-Network Theory

To reiterate, actor-network theory is a framework which aims to understand the relationships of both animate and inanimate stakeholders in a sociotechnical system's network. When analyzing a sociotechnical system, ANT dictates that human and nonhuman stakeholders should be treated equally with respect to their potential roles within a system. This is because when analyzing a system without utilizing ANT, the analyzer may tend to focus too heavily on the human stakeholders. ANT attempts to mitigate this. With this in mind, ANT will be extremely useful in the analysis of the UVa CS curriculum because there are both human and nonhuman stakeholders of importance, and ANT will reveal the connections between these stakeholders. ANT also dictates that sociotechnical networks are volatile in nature and thus change over time. This is not as simple as the entry and exit of actors in the network, or certain actors becoming more or less prominent over time. While these changes occur, the more

interesting and nuanced changes pertain to volatility in the connections between stakeholders. Given that CS is a highly dynamic field, ANT is a prime framework for analyzing it.

There are several key components that comprise an undergraduate computer science curricular program. In this context, a program is defined as an educational pathway for earning a degree, such as a Bachelor of Science. As a whole, a computer science program is the "network" in question, and in relation to actor-network theory, the actors in this network are all entities that have a stake in said network. In accordance with ANT, actors are not necessarily human. For example, in the context of a CS program, actors can be human entities, such as the students enrolled in the program, as well as tangible non-human entities, such as the courses in the curriculum required for graduation. The point of a network is to draw connections between entities to illustrate some form of relationship – ANT describes this process as "translation." ANT prescribes that translation occurs in a material sense (more tangible; dealing with things) as well as in a "semiotic" sense (less tangible; dealing with ideas) (Fenwick and Edwards, 2010).

ACM/IEEE Joint Task Force

With an undergraduate CS program setting the stage for the network in question, identifying the key actors is the next logical step before attempting translation. The first key actor in the network is the curriculum material itself. The breadth and depth of topics covered at an undergraduate level is frequently debated by experts in the field. The Association for Computing Machinery (ACM) and The Institute of Electrical and Electronics Engineers (IEEE) work together every so often to assess the current state of the field of CS, and draft a set of guidelines for what the experts in the field deem to be a representative collection of recommendations for an undergraduate education in CS. The most recent summit convened in 2013, with curricular recommendations from this joint group dating back over fifty years ago to

1968. The process for formulating the 2013 report involved coordinating with global experts in CS academia, and then using a survey to gauge the current state of CS education at hundreds of higher education institutions. The survey results allowed the ACM/IEEE joint committee to propose improvements to the 2008 recommendations. The 2013 report highlighted 18 knowledge areas critical to a foundational CS education, several of which were entirely new from those outlined in 2008 (new areas indicated with an asterisk):

- AL - Algorithms and Complexity
- AR - Architecture and Organization
- CN - Computational Science
- DS - Discrete Structures
- GV - Graphics and Visualization
- HCI - Human-Computer Interaction
- IAS - Information Assurance and Security*
- IM - Information Management
- IS - Intelligent Systems
- NC - Networking and Communications
- OS - Operating Systems
- PBD - Platform-based Development*
- PD - Parallel and Distributed Computing*
- PL - Programming Languages
- SDF - Software Development Fundamentals
- SE - Software Engineering
- SF - Systems Fundamentals*

- SP - Social Issues and Professional Practice

As shown above, there were four entirely new knowledge areas added to the suggested curriculum, demonstrating volatility of CS as a field given that these areas were not present in the report released only five years prior. It is important to note that these "new" knowledge areas did not materialize out of thin air; rather, this material existed as a sub-area of another knowledge area. However, the new areas were deemed to have evolved and grown in size to a point where they were deserving of a dedicated knowledge area. Overall, given that this actor in the network is subject to change frequently, this will have implications for other actors in the network (Computer Science Curricula, 2013, p. 14).

Evolution of Knowledge Areas

To further illustrate the evolution of the knowledge areas, Figure 1 below demonstrates the fluctuation in number of credit hours between the 2001, 2008 and 2013 recommendations. Credit hours refer to the number of hours allocated for a student's time in lecture in one week, with one course typically having three credit hours, and thus a number greater than three in any given cell represents that the corresponding knowledge area spans multiple courses. Additionally, the 2013 curriculum introduced a multi-tiered approach – concepts falling under tier 1 of a knowledge area were deemed essential, while a vast majority concepts in tier 2 should still be covered, but some may be omitted. The report emphasized the necessity of 100% tier 1 coverage, with 80-90% tier 2 coverage being acceptable. Also, the curriculum advises additional elective material should supplement the "core" topics in order to be fully prepared to graduate, meaning that Figure 1 is not even comprehensive.

Knowledge Area	CS2013 Tier1	CS2013 Tier2	CS2008 Core	CC2001 Core
AL-Algorithms and Complexity	19	9	31	31
AR-Architecture and Organization	0	16	36	36
CN-Computational Science	1	0	0	0
DS-Discrete Structures	37	4	43	43
GV-Graphics and Visualization	2	1	3	3
HCI-Human-Computer Interaction	4	4	8	8
IAS-Information Assurance and Security	3	6	--	--
IM-Information Management	1	9	11	10
IS-Intelligent Systems	0	10	10	10
NC-Networking and Communication	3	7	15	15
OS-Operating Systems	4	11	18	18
PBD-Platform-based Development	0	0	--	--
PD-Parallel and Distributed Computing	5	10	--	--
PL-Programming Languages	8	20	21	21
SDF-Software Development Fundamentals	43	0	47	38
SE-Software Engineering	6	22	31	31
SF-Systems Fundamentals	18	9	--	--
SP-Social Issues and Professional Practice	11	5	16	16
Total Core Hours	**165**	**143**	**290**	**280**

All Tier1 + All Tier2 Total	**308**
All Tier1 + 90% of Tier2 Total	**293.7**
All Tier1 + 80% of Tier2 Total	**279.4**

Figure 1: ACM/IEEE curriculum recommendation – knowledge area versus number of suggested credit hours (Computer Science Curricula, 2013, p. 37).

Many of the knowledge areas retained similar credit hours, but a few changed drastically. The two areas to pay close attention to are *Architecture and Organization* (AR) and *System Fundamentals* (SF) - the 2013 curriculum placed a smaller emphasis on AR than in past years, while it places a higher dedicated focus on SF. The new SF area is demonstrative of the shifting

focus of the 2013 curriculum, which introduces the next actor in the network: teaching methodologies.

Teaching Methodologies

A closely related actor of the curriculum itself is the way that the curriculum is conveyed to students, or the teaching methodologies. The most general of the methodologies is constructivism, the idea that a student "constructs" a mental model of the material being taught, and if new knowledge relies on the foundation of a previously constructed mental model, then the foundational model must be cemented before new knowledge can be appended (Ben-Ari, 2001). Additionally, students must construct knowledge themselves by engaging in learning activities, such as homework assignments, to grapple with complex ideas, as opposed to simply memorizing lecture notes. In other words, although somewhat of a simplification, learn by doing.

A slightly less general methodology is breadth versus depth first teaching. In the case of breadth first, many topics are explored in rapid succession without delving deeply into any one topic. Depth first, on the other hand, deals with carefully examining a topic in extreme detail before moving onto the next topic. In other words, breadth first favors a high-level overview of a subject with exposure to many different topics, whereas depth first is detail-oriented. A curriculum does not have to use one or the other exclusively. The 2013 recommendation does not prescribe one over the other; rather, the curriculum emphasizes that it allows each institution the freedom to implement the curriculum as it sees fit. For example, Northeastern University utilizes a depth first approach for freshmen-level courses, but later expands to breadth first in later courses (Proulx et. al.).

The next methodology is practical project experience. The 2013 curriculum specifies that all students should complete at least one substantial software development project. The goal of

this is to practice skills that cannot be taught in lecture, such as "being integrative, … evaluation of potential solutions," teamwork, and interpersonal communication skills (Computer Science Curricula, 2013, p. 24).

Although numerous other methodologies exist, the final one that will be discussed is "gamification." This refers to "a new technology that use[s] elements such as badges, level, and point of game play … implemented in a non-game context" (Oktaviati et. al., 2018). Firstly, this methodology is not prescribed by the 2013 curriculum. Additionally, researchers disagree about the positive effects of this mechanism on student motivation. Several studies maintain that it is in fact a beneficial motivator (Khaleel et. al., Oktaviati et. al., Varannai et. al.), while others found it to have a negative impact on motivation over time (Hanus et. al.). Solely based on these studies, it is not possible to say that it is beneficial, but this methodology is in use in the CS curriculum at UVa, so its validity within the UVa coursework is evaluated in the *Evidence and Analysis* section below.

Thoughts on Translation

Explicitly enumerating the actors that have been identified up until this point, both the curriculum itself, as well as teaching methodologies have been discussed. Other important actors include the obvious ones: the students enrolled in the CS program, the professors teaching the material, the faculty involved with curricular modifications, accreditation boards such as The Accreditation Board for Engineering and Technology, Inc. (ABET), employers of newly graduated students, and graduate schools, among others. Fully constructing a network and all its connections out of all these actors is beyond the scope of this research. The important translation to be noted is that between the curriculum and the teaching methodologies, because understanding this connection brings to light new insight as to why CS curricula require changes

aside from the fact that new technologies are emerging. Another key actor in the network that is driving this translation is the underlying principles driving curricular reform. The important takeaway from the 2013 recommendations is that the ideal CS curriculum is interdisciplinary in nature. This aims to ensure that students are prepared for a variety of professions upon graduation. Not only will this make them more marketable, but also more prepared to adapt with the field as it changes. In the next section, this actor-network will be mapped onto the UVa CS department in order to explore its strengths and weaknesses.

Evidence and Analysis

The spotlight is now on UVa's CS program. Additional actors are present in the UVa CS program's actor-network. To begin, there are two CS degrees possible: a Bachelor of Science (B.S.), which resides within the School of Engineering and Applied Science, and a Bachelor of Arts (B.A.), which can be earned through the College of Arts and Sciences. Each will be discussed in greater detail below. An additional actor in the network includes the recently developed CS pilot program, a new test curriculum only available to students between 2018 and 2020 that served as a replacement to the traditional curriculum, which could be used to satisfy either the B.S. or B.A., depending on the school of enrollment of the student.

Bachelor of Science

The B.S. CS degree requires 127 credit hours in order to graduate. The curriculum includes about thirteen of the eighteen knowledge areas in its core curriculum, falling short of the 2013 recommendation. This curriculum only includes space for four CS elective courses, which ideally allow for the other knowledge areas to be covered in some way (such as NC, IS, GV, or HCI; see Figure 1 for the mapping of abbreviation to knowledge area title). Although certain courses in the B.S. curriculum further the goal of interdisciplinary learning, such as by requiring

PHYS 2415: General Physics II, this course does not provide direct value to the CS curriculum other than by furthering one's problem solving ability. Although the 2013 recommendation indicated that problem solving ability is a critical skill that is difficult to teach, this ability could most certainly be fostered in a computing-related course instead that also involves a knowledge area that is underrepresented in the curriculum.

Bachelor of Arts

Beginning in 2006, students who were enrolled in the College of Arts and Sciences had the option of completing the B.A. CS degree. The focus of the program was, and still is, to be as interdisciplinary as possible. Because students still have to complete liberal arts requirements in the College, many of the requirements for the B.S. degree do not exist in the B.A. degree. Certain hallmarks of the B.S. degree are not required for the B.A., such as Advanced Software Development (discussed below in *Gamification at UVa*) or Operating Systems. Just like the B.S. degree, however, students still have to complete four CS electives. Additionally, B.A. students have to complete four "integration electives," courses that lack the technical rigor of courses found within the Engineering School in favor of offering a more interdisciplinary paradigm. In theory, the integration electives seem to align perfectly with the interdisciplinary principle from the 2013 recommended curriculum. Although this is true, the integration electives come with a cost. These courses foster a diverse interdisciplinary mindset by directly translating CS to another department, such as BIOL 4230: Bioinformatics and Functional Genomics or CHEM 3240: Coding in Matlab/Mathematica with Applications. However, mandating these electives and other liberal arts courses requires that some CS courses be removed in order to make the degree attainable in four years. Thus, a weak foundation in the main concepts of CS results because of the trimmed down core CS set of courses. One such example of this is the absence of

operating systems (CS 4414) from the B.A. program. Although regarded as one of the most challenging courses within the UVa CS Department, it covers several critical knowledge areas like AR and PD to supplement the obvious OS requirement. Although just one example, it exposes the main weakness of the B.A. program: only requiring courses in roughly seven of the eighteen knowledge areas discussed by the 2013 recommendation.

Pilot Program

The pilot program was instituted as a trial curriculum in 2018 and will no longer accept new students beginning in the Fall of 2020 (Sherriff, UVa CS 2020 Curriculum Pilot). To reiterate, this curriculum consists of fewer core courses than required by the traditional B.S. degree, but the CS department claims that it manages to cover the same material that the traditional curriculum does. The pilot aims to restructure courses to better cover content, to eliminate material duplicated across courses, to better prepare students for internships earlier in their career, and to better unite the B.A. and B.S. CS programs. The pilot curriculum restructures the traditional curriculum's approach to CS: rather than having every relatively distinct knowledge area be its own course, as is the case with the traditional curriculum, the pilot groups content into larger buckets and offering a two-course sequence for each bucket (buckets include *data structures and algorithms*, *discrete mathematics and theory*, *computer organization and architecture*, and *software development essentials*) (Sherriff, UVa CS 2020 Curriculum Pilot). Although this appears to accomplish its goals, this new proposal does not introduce any new material, lacking some of the 2013 recommendation knowledge areas entirely, just as the B.S. and B.A. do.

Practical Project Experience

All students aiming for the B.S. degree must complete CS 3240: Advanced Software Development. This course satisfies the 2013 recommendation for a sizable software development project; students must work in teams to develop a web application using the Python Django framework, and also are exposed to the latest in software engineering practice, such as agile and scrum. B.S. students must also complete a capstone project, where they either perform independent research or work on a software development project requested by a real-world client. Overall, the B.A. program does not have a mandatory course that involves a practical project.

Gamification at UVa

As previously stated, all B.S. students are required to complete CS 3240: Advanced Software Development. This course as of the Fall 2019 semester has been "gamified" to an extent by having an "XP" system for grading, as opposed to a traditional points-based system. The system follows a positive reinforcement system by rewarding students for success by rewarding XP points based on what has been done correctly, rather than penalizing students for incorrectness. At the end of the semester, the amount of XP a student has is correlated to a letter grade, so although students still have a traditional evaluation metric, they focus less on letter grades and more on achieving more XP. One such example of this is that the semester long project does not have an XP cap, meaning that students who far exceed expectations can earn more than the predetermined maximum XP for the project. This positively incentivizes students to go above and beyond and lessens the focus on scoring. This interdisciplinary course spans several knowledge areas including HCI, IM, PBD, SDF, SE, and SF, all while fostering students' teamwork abilities on a large project, thus making it a course that checks a lot of the 2013

recommendation's requirements. However, the B.A. program does not require this course in its curriculum.

Discussion

It is apparent that the B.A. degree falls short of the ACM/IEEE recommendations for a CS curriculum. Although the B.A. embraces the recommendation's goal of being interdisciplinary, it is lacking significant core CS material that should be required. The B.S. degree more closely aligns with the thoroughness of the 2013 recommendation, but still lacks several of the knowledge areas that have been established as necessary. Many of the key knowledge areas fall into elective coursework. Thus, the current UVa CS curriculum makes it very difficult to take a course in every knowledge area. Additionally, many of these electives require prerequisite courses that students do not complete until late in the curriculum, so options are limited.

A key element of the 2013 report was that a CS education must be flexible in nature – although there are many composite subfields that make up CS in its entirety, and this list continues to grow, the length of an undergraduate education is not proportionally increasing in duration, so it will thus be impossible to cover every topic in depth. It would be logistically impossible for a four-year undergraduate degree to meet the minimal 2013 recommendation of 279.4 credit hours solely on the basis of time alone. With this being said, since UVa offers courses that touch on each of these subjects, its CS offerings are seemingly sufficient. However, the B.A. program does not cover enough core material, and even though the coverage provided by the B.S. is more thorough, it still is insufficient by the 2013 report's guidelines. Finally, the pilot program's goal of bringing the B.S. and B.A. degrees closer together is much needed, but needs to be expanded to best cover all critical knowledge areas.

One such solution would be to require all students seeking a CS degree to complete CS 3240, and revert CS 3240 to a previous version of the course since there are other CS courses that are available for learning web application programming and best practices, such as the electives Programming Languages for Web Applications (CS 4640) or Internet Scale Applications (CS 4260). Historically, CS 3240 has been a testing ground for new curricular additions. In Spring 2009, the course required students to utilize Lego robots and replace the default firmware with one called LeJOS, so that the students could then interface with the robots using the programming language Java. The project taught concepts such as "threading, network communications, and the implementation of command protocols" (Lew et. al., 2010). If all students were to take this version of the course, the number of knowledge areas explored would increase, including GV, IS, NC, OS, PD, SDF, SE and SF. Courses that do not exclusively deal with one topic force students to have the interdisciplinary focus, while exposing them to knowledge areas that all computing students should explore before completing an undergraduate degree. Even though the current version of this course spans multiple knowledge areas, this previous version spans several additional knowledge areas that are entirely absent from required courses, such as GV, and still would fulfill the requirement of a semester-long project.

Conclusion

Redesigning a curriculum from the ground up is no easy feat. The introduction of the pilot program is a notable achievement, and its goals of restructuring courses to better cover content, eliminating duplicated material, better preparing students for internships earlier in their career, and better unite the B.A. and B.S. CS programs are clearly valid and necessary as the traditional curriculum is becoming antiquated. Via ANT analysis, it is evident that there are many subtle stakeholders to pay attention to as curricular improvements are developed, and the

best path forward for UVa CS is one that delivers technical excellence in an interdisciplinary way.

References

ACM Curriculum Committee on Computer Science. 1968. Curriculum 68: Recommendations for Academic Programs in Computer Science. Comm. ACM 11, 3 (Mar. 1968), 151-197.

ACM/IEEE-CS Joint Task Force on Computing Curricula. 2001. ACM/IEEE Computing Curricula 2001 Final Report. http://www.acm.org/sigcse/cc2001.

ACM/IEEE-CS Joint Task Force for Computer Curricula 2005. Computing Curricula 2005: An Overview Report. http://www.acm.org/education/curric_vols/CC2005- March06Final.pdf

ACM/IEEE-CS Joint Interim Review Task Force. 2008. Computer Science Curriculum 2008: An Interim Revision of CS 2001, Report from the Interim Review Task Force. http://www.acm.org/education/curricula/ComputerScience2008.pdf

Basawapatna, A. R., Koh, K. H., & Repenning, A. (2010). Using scalable game design to teach computer science from middle school to graduate school. Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education - ITiCSE '10, 224–228. https://doi.org/10.1145/1822090.1822154

Ben-Ari, M. (2001). Constructivism in Computer Science Education. Journal of Computers in Mathematics and Science Teaching, 20(1), 45-73.

Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science. (2013). doi: 10.1145/2534860

Hanus, M.D., & Fox, J. (2015). Assessing the effects of gamification in the classroom: A longitudinal study on intrinsic motivation, social comparison, satisfaction, effort, and academic performance. Comput. Educ., 80, 152-161.

Khaleel, F.L., Ashaari, N.S., Wook, T.S., & Ismail, A. (2017). Gamification-based learning framework for a programming course. 2017 6th International Conference on Electrical Engineering and Informatics (ICEEI), 1-6.

Kirschner, P. A., Sweller, J., & Clark, R. E. (2006). Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching. Educational Psychologist, 41(2), 75–86.

Oktaviati, R., & Jaharadak, A.A. (2018). The Impact of Using Gamification in Learning Computer Science for Students in University.

Proulx, V. K., Rasala, R., & Fell, H. (n.d.). Foundations of Computer Science: What are they and how do we teach them? [Scholarly Publication, Northeastern University]. Retrieved from http://www.ccs.neu.edu/home/vkp/Papers/Foundations-iticse96.pdf

Rice, J. R., & Rosen, S. (2017, March 15). History of the Department. Retrieved from https://www.cs.purdue.edu/history/index.html

S. Heckman, T. B. Horton and M. Sherriff, "Teaching second-level Java and software engineering with Android," 2011 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T), Honolulu, HI, 2011, pp. 540-542.

Sherriff, M. (n.d.). UVa CS 2020 Curriculum Pilot. Retrieved from http://pilot.cs.virginia.edu/

Varannai, I., Sasvari, P., & Urbanovics, A. (2017). The Use of Gamification in Higher Education: An Empirical Study.