Clifford B. Anderson and Gayathri Narasimham

An Introduction to the Beauty and Joy of Computing for Theological Librarians

The dizzying growth of the digital humanities during the past decade has kindled interest among students and faculty at divinity schools and theological seminaries in technologies such as markup languages, geospatial information systems, and network analysis applications.¹ This growth has likewise sparked the curiosity of theological librarians in supporting teaching and research in the digital humanities. Indicative of this interest are the recurring presentations about the digital humanities at Atla annual meetings and webinars.² As theological schools prepare students for careers in the church and social ministries, is there a way to provide them with a humanistic and theological perspective on our increasingly computational and algorithmically-driven society? The COVID-19 pandemic has reinforced the need for understanding of the digital world, as pastors seek to communicate with their congregations using tools like Zoom, Facebook, and the Adobe Creative Cloud, and as they worry about the distorting effects of filter bubbles, echo chambers, and the digital divide on their church members.

The teaching of digital humanities at theological institutions raises distinct challenges, however. At university-based divinity schools, students and faculty may discover like-minded colleagues at digital humanities centers or digital scholarship labs. But the staff members in these digital scholarship units may not have expertise in religious studies or theology. At independent theological schools, by contrast, the task of teaching digital humanities may fall to intrepid

1 An initial version of this paper was presented at Snap!Con 2020 on July 31, 2020 (see Clifford Anderson and Gayathri Narasimham, “Teaching BJC with NetsBlox,” Short Talk, [July 2020]). The authors would also like to thank David Michelson and Mark Schoenfield for their comments and suggestions for improvements on prior drafts of this paper.

2 Clifford B. Anderson, Christopher P. Benda, and Eileen Crawford, “Doing Digital Humanities in Theological Libraries,” in *American Theological Library Association Summary of Proceedings*, vol. 69, 2015, 189–97; Bobby Smiley, “Theological Librarianship in the Age of Digital Humanities,” ed. Tawney Burgess, *Summary of Proceedings, 70th Annual Conference of the American Theological Library Association*, vol. 70, 2016, 22–32; Richard Manly Jr. Adams et al., “Digital Humanities and Libraries and Archives in Religious Studies,” in *American Theological Library Association Summary of Proceedings*, vol. 73, 2019, 42–48; Richard Manly Jr. Adams and Clifford B. Anderson, “Digital Humanities and Libraries and Archives in Religious Studies,” Webinar (November 2020), <https://vimeo.com/480835487>.

faculty or librarians who, having taught themselves, have learned enough to offer workshops or experimental courses in digital humanities.

We argue that the best way to prepare students for ministry in an increasingly computational context is to teach the fundamentals of computational thinking and algorithmic literacy rather than specific tools and techniques, which arise and fade away at an accelerating pace. We believe that theological librarians may learn from the innovators of the *Beauty and Joy of Computing* (<https://bjc.berkeley.edu/>) curriculum about how to provide these foundations to students, either in collaboration with faculty or as co-curricular library instruction in digital literacy. In this paper, we discuss the origin and implementation of a BJC course at Vanderbilt University as well as subsequent efforts to adapt its core ideas to teaching the digital humanities.

1 Collaborators

Clifford B. Anderson approaches the teaching of digital humanities primarily from the perspective of librarianship and theological studies. Anderson is a librarian who also holds a secondary appointment as professor of religious studies. Over the past decade, he has taught digital humanists, primarily from disciplines like English or history but also foreign languages and theology, among others, to write programs to address questions in their disciplines. Given his interest in digital editions in the Text Encoding Initiative (TEI), Anderson's primary programming language is XQuery, a functional programming language for processing XML and JSON.³ For several years, he ran a weekly seminar on XQuery at the Vanderbilt University Libraries.

Gayathri Narasimham has a background in psychology and digital learning. She has taught courses in statistics and research methods and more recently organized a workshop in data visualization. She has programming experience in Python and R, and wanted to explore teaching a first-level, introductory computer science course for undergraduates who did not have a background in CS or coding. In addition, Narasimham also teaches sections of the required ethics course for undergraduate computer science majors.

The authors adapted the course from curriculum developed at the University of California, Berkeley as well as from its implementation at the high school level

³ See Clifford B. Anderson and Joseph C. Wicentowski, *XQuery for Digital Humanists* (College Station: Texas A&M University Press, 2020).

to teach AP CS Principles.⁴ While seeking to stay true to the vision of the BJC, we tailored the course to fit our academic schedule, meet the needs of our undergraduates, and expand the range of programming concepts we taught. This experience leads us to believe that the principles of the BJC, suitably adjusted, might become a vehicle for teaching computational thinking or algorithmic literacy to students and scholars of religious studies and theology. In this paper, we discuss the difficulty of provisioning instruction in the digital humanities to students of religion and theology and suggest why customizing the BJC and a blocks-based environment would go a long way toward meeting those pedagogical challenges.

2 Challenges with Teaching Coding in the Digital Humanities

The question whether there is an intrinsic connection between computer programming and digital humanities has turned out to be fraught. While, as Bethany Nowviskie reminds us, the so-called ‘hack’ versus ‘yack’ (or coding versus theorizing) debate may have turned out to be a straw person argument all along,⁵ the question of how much programming students of the digital humanities should learn in the course of their studies remains a live issue.

If you plan to teach students to program as part of a digital humanities course, to what degree should you contextualize that instruction? This question surfaces across many disciplines. As computation becomes increasingly intrinsic to many disciplines, ranging from computational biology to computational musicology and beyond, instructors grapple with the best way to integrate programming skills within disciplinary courses. As Mark Guzdial has noted, contextualizing computer science education promotes student engagement and retention, but at some cost to instructional efficiency and transferability of knowledge between domains.⁶

⁴ Daniel D. Garcia, Brian Harvey, and Luke Segars, “CS Principles Pilot at University of California, Berkeley,” *ACM Inroads* 3, no. 2 (June 2012): 58–60, doi:10.1145/2189835.2189853.

⁵ Bethany Nowviskie, “On the Origin of ‘Hack’ and ‘Yack,’” in *Debates in the Digital Humanities* 2016, ed. Matthew K. Gold and Lauren F. Klein (Minneapolis: University of Minnesota Press, 2016), 66–70, doi:10.5749/j.ctt1cn6thb.10.

⁶ Mark Guzdial, “Does Contextualized Computing Education Help?” *ACM Inroads* 1, no. 4 (December 2010): 4–6, doi:10.1145/1869746.1869747.

A ready-to-hand solution is complementing a major with a minor or major in computer science.⁷ For students who are motivated to learn computer science in its own right, undertaking a double major or adding a minor is a possibility, though it leaves students with the task of integrating the disciplines. As data science emerges as an independent interdisciplinary field of study, students from quantitatively-focused disciplines may benefit from adding a minor in data science since the connections are already latent. But, as Susan Mitchell and Katharine Cole observe, for students from other disciplines such as the humanities, fulfilling the mathematical prerequisites may prove challenging and, in general, they may also find few areas of overlap between the courses in their majors.⁸ An alternative is to develop a CS+X program, which aims to teach computer science from the perspective of particular disciplines.

David J. Birnbaum and Alison Langmead make the case for what they term “a task-driven programming pedagogy” when teaching digital humanists to program.⁹ Drawing inspiration from contemporary trends in foreign language instruction, they contend that the emphasis in programming instruction should be on accomplishing goals relevant to the digital humanities, not trying to master the syntax and grammar of a computing language encyclopedically (at least not at first).

But, just as in the case of the movement away from a grammatical focus in language teaching to an oral-proficiency focus, there is another type of programming textbook and another type of course: the task-based, proficiency-oriented one. Books of this sort sometimes include the word “cookbook” in the title, and what characterizes these books (and courses that follow the same model) is that they are organized not around learning, say, all of the numeric data types and then all of the non-numeric simple types and then all of the complex types, but around accomplishing specific coding tasks.¹⁰

The approach they counsel is to remain agile, approximating the practices of digital humanities practitioners by learning how to break up big conceptual problems into smaller executable tasks (“algorithmic thinking”), relying on second-

⁷ Susan Mitchell and Katharine Cole, “X+CS: A Computing Pathway for Non-Computer Science Majors,” American Society for Engineering Education, in *ASEE Mid Atlantic Section Spring Conference*, (2020), 3

⁸ Ibid.

⁹ David J. Birnbaum and Alison Langmead, “Task-Driven Programming Pedagogy in the Digital Humanities,” in *New Directions for Computing Education: Embedding Computing Across Disciplines*, ed. Samuel B. Fee, Amanda M. Holland-Minkley, and Thomas E. Lombardi (Cham: Springer International Publishing, 2017), 63–85, doi:10.1007/978-3-319-54226-3_5.

¹⁰ Ibid., 77.

ary sources in print and online to address those tasks (“looking stuff up”), and iterating toward solutions, refactoring and adding functionality at each pass (“incremental development and iteration”).¹¹

In our own teaching of the digital humanities, we have erred in both directions, trying either to cram too much programming instruction into a single digital humanities course or to tailor computational concepts too narrowly to the tasks at hand, leaving the students with conceptual gaps.

In spring 2014, Anderson and David A. Michelson, Associate Professor of the History of Christianity at the Vanderbilt Divinity School, co-taught a digital humanities course titled *Topics in Digital Humanities: Introduction to Digital Text Editing and Analysis*.¹² The course provided a general introduction to digital humanities with a focus on the Textual Encoding Initiative (TEI) but also coverage of subjects such as natural language processing in R and geospatial analysis. The class, with seven students enrolled (primarily M.Div. and Th.M. students), met weekly as a seminar for two hours. The programming backgrounds of students in the course varied widely. In most cases, they had no prior experience with programming at all, but a few had extensive experience.

The seminar format turned out to be the most challenging aspect of the course for the teaching of programming. While the seminar format worked well for the theoretical discussions (led by Michelson and Anderson) and adequately for the introduction of methodologies (such as how to employ TEI XML, led by Michelson), Anderson found these weekly sessions were not sufficient to cover programming techniques. While the instructors assigned technical readings in advance, students found it frustrating to read them as the concepts and presentation style were unfamiliar. As a result, Anderson and Michelson spent more time lecturing about these concepts during class than they had intended in order to cover what students had failed to grasp from the readings. This left little opportunity for students to program together during class, increasing the likelihood that they would run into issues when writing code for their projects during the week. To their credit, students managed to submit uniformly excellent final projects. But, looking back on the syllabus, Anderson considers that he tried to introduce too much programming in too short a span. As Mark Guzdial has cautioned, teaching programming in a contextualized course requires that instructors cover less material than they would in a non-contextual-

¹¹ Ibid., 78f.

¹² For the course website, including the syllabus, see: <http://paralipomena.com/>. Anecdotally, Anderson and Michelson believe that this was the first course explicitly on the topic of digital humanities offered by an accredited institution of the Association of Theological Schools; as such, the course was expressly experimental.

ized introduction to computer science.¹³ Certainly, it was too ambitious to try to cover XML, TEI, XPath, XQuery, and XML databases along with the R programming language and GIS libraries during a weekly seminar. While the goal was to provide students with a top to bottom perspective on the technology stack of multimodal digital editions, they probably should have focused solely on the XQuery/XPath/XML technologies; as Birnbaum and Langmead indicate, there was no need to teach the fundamentals of computer science, from data types to functions to recursion, to serve the purpose of enabling students to create their own digital editions of religious studies texts. In the next iteration of the course, Michelson and Michelle Taylor, then a postdoc at Vanderbilt University and now Continuing Instructor of English at the University of South Florida, successfully streamlined the course to focus primarily on TEI and the making of digital editions, limiting their coverage of XPath, XQuery, and XML databases to a few case studies rather than providing a fundamental introduction to these tools.

A subsequent digital humanities course Anderson co-taught took a different tack. In fall 2014, he began an informal teaching collaboration with Joy Calico on a one-credit course for undergraduates that explored the history of Berlin in the twentieth century through a series of contemporary maps. By spring 2018, the course had turned into a graduate seminar in the Department of German, Russian and East European Studies, meeting twice weekly. The title of the course, “The Digital Flâneur: Mapping Twentieth-Century Berlin,” indicates their approach to teaching technology.¹⁴ Rather than aim to teach students any single programming language systematically, they explored a range of technologies. The first part of the semester examined geospatial information systems, including GeoJSON, online georeferencing tools, and spatially-aware databases. During the second half of the semester, they paraded more quickly from tool to tool, introducing everything from CityGML, Neo4j, the R programming language, to Wikidata. At each stop, Anderson and Calico demonstrated how the particular construct or tool shed new light on the understanding of the city of Berlin.

Intrinsic to digital flânerie is that the instructors never pause long at any single technology. To cover this amount of ground, the class had to keep moving forward. Consequently, students dedicated significant effort outside of class delving into technologies on their own to create their final projects. From a pedagogical perspective, this experience of teaching yourself replicates the learning process of professional digital humanists, preparing graduate students to be-

¹³ Guzdial, “Does Contextualized Computing Education Help?” 5.

¹⁴ Clifford B. Anderson and Joy H. Calico, “The Digital Flâneur: Mapping Twentieth-Century Berlin,” in *Quick Hits for Teaching with Digital Humanities: Successful Strategies from Award-Winning Teachers*, ed. Christopher J. Young et al. (Bloomington: Indiana University Press, 2020).

come self-reliant when learning new tools and techniques. That said, the instructors had to assist students with their projects in several cases when the technologies could not be assimilated so quickly. For instance, Anderson wrote custom SPARQL queries to retrieve geospatial data from Wikidata and XQuery to transform that data into GeoJSON. Like any tourist operation, digital *flânerie* works best when students explore the territory in the company of seasoned guides. But how much of such a whirlwind tour remains in students' minds after the semester ends? A course of this sort depends on students' interest in returning to their favorite landmarks, tarrying over overlooked details while also exploring the intersecting boulevards and byways.

3 Teaching the BJC

The proposal for a Beauty and Joy of Computing course arose from discussions about computational thinking with an interdisciplinary group of faculty and staff, including Anderson and Narasimham. In the context of those discussions, Akos Ledeczi (Professor of Computer Engineering and Director of Graduate Studies in Computer Science), Doug Schmidt (Cornelius Vanderbilt Professor of Engineering, Professor of Computer Science, and Professor of Computer Engineering), and Julie Johnson (Professor of the Practice of Computer Science and Director of Undergraduate Studies in Computer Science) invited Anderson to develop a version of the BJC for the School of Engineering. Anderson created the initial version of the course in Fall 2018 and, with lots of help from fellow faculty members, Ledeczi, Brian Broll (research scientist at the Institute for Software Integrated Systems at Vanderbilt University), and Corey Brady (Assistant Professor of Mathematics Education, Department of Teaching and Learning, Peabody School of Education), taught its first three iterations. Narasimham taught the final version of the course in spring 2019. As noted above, Anderson and Narasimham both come from disciplines other than computer science.

The computer science faculty approved the course in spring 2018. The course description read: “Fundamental concepts of computing including abstraction, algorithms, design, and distributed computation. Hands-on curriculum focusing on translating ideas into working computer programs and developing a mastery of practical computational literacy.” In practical terms, we planned to address these topics in seven two-week units, promising to teach everything from “control constructs, variables and functions” to “recursion, efficiency, and the limits of computability” with a good measure of networking and distributed computing as well. As we designed the course, we also tried to balance examples from mathematics, science, and the humanities. There is distinct beauty to each of

these domains. How to bring it more to the fore in our teaching? Once again, we sought to exemplify a design principle of the BJC: “To help them feel ‘CS-smart,’ students need to see their code not just as a means to an end with an effect they like but as ‘poetry,’ code with structure, elegance and power.”¹⁵ Our intention was to teach computer science as a way to uncover the inherent beauty of students’ home disciplines.

3.1 Goals

We aimed to keep the goals of our offering of the BJC consonant with its original intentions. First, we wanted to provide a rigorous but accessible introduction to computer science to students who felt unprepared for the sequence of programming courses (taught with Java) for majors and minors in computer science. As Dan Garcia stated in a 2012 keynote address, “The purpose of this course is to attract nontraditional computing students (especially women and minorities, but also English majors) to the breadth and depth of ideas in modern computer science.”¹⁶ The authors of a recent paper on the design principles behind the BJC underscored this point: “In BJC, our absolute top-level goal is broadened participation in computer science.”¹⁷ As with other colleges and universities, we sensed a gap at Vanderbilt in our course offerings for students who wanted to explore the relevance of computer science to their disciplinary interests without committing to learning Java over the course of one or two semesters. We wanted to broaden access to computer science, encouraging students who lacked prior exposure to computer science in high school to consider a minor or double major. But we also sought to convey the fundamentals of computer science to students in the humanities and social sciences who might not have the mathematical prerequisites for a minor or major.

Our course unabashedly taught computational thinking through the discipline of writing, analyzing, and refactoring computer programs. In addition to utility and efficiency, we devoted attention to matters of elegance and style as we examined blocks of code. We stressed functional programming concepts in our teaching because of their inherent elegance without dwelling on the theoretical differences between functional and procedural programming. For instance,

¹⁵ Paul Goldenberg et al., “Design Principles Behind Beauty and Joy of Computing,” in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 2020, 223.

¹⁶ Daniel D. Garcia, “The Beauty and Joy of Computing (BJC), AP CS Principles, and the CS 10k Effort,” *Journal of Computing Sciences in Colleges* 27, no. 5 (2012): 126–27.

¹⁷ Goldenberg et al., “Design Principles Behind Beauty and Joy of Computing,” 221.

after teaching students procedural programming concepts such as for and while loops, we showed them how to simplify their programs with functional constructs such as map, filter (“keep”), and reduce (“combine with”). This underlying emphasis on the simplicity and parsimony of functional programming reflects the roots of the BJC curriculum in Scheme. “Berkeley’s approach to introductory computer science has been profoundly shaped by the seminal text *Structure and Interpretation of Computer Programs...*,” explained Brian Harvey and Jens Mönig in 2010. “In our new course, we wanted to preserve the big ideas we gleaned from their curriculum, including recursion and higher order procedures as organizing tools for flow of control.”¹⁸ As we customized the BJC to our context, we aspired to keep that pedagogical heritage intact.

In the first week, we asked students to watch Dan Garcia’s presentation of the BJC in a TEDxBerkeley talk from March 2015.¹⁹ During the talk, Garcia uses the program “Vee” to illustrate the surprising elegance of Snap! For those who have not seen “Vee” in action, the program creates a tree with an arbitrary number of leaves and branches. As Harvey and Mönig explain, Vee incorporates sophisticated ideas such as “arbitrarily deep recursion” and treating functions as first-class data while also rendering these concepts more accessible by “focusing students’ attention on the recursive case rather than the base case” through the use of randomness.²⁰ After the midpoint of the semester, students had learned enough to code Vee themselves. In similar fashion, students learned how to create loops during the second week of class and, by the end of the semester, they had mastered enough concepts to create their own looping control structures using higher order functions.

A challenge we faced when adapting the BJC to Vanderbilt was keeping its broad scope while accommodating our course schedule. As Brian Harvey relates, “Berkeley has a 14-week semester, and our course ‘The Beauty and Joy of Computing’ (BJC) meets seven hours per week: two lecture hours, four lab hours, and one discussion hour.”²¹ By contrast, our course meet twice a week for seventy-five minutes with no labs or precepts. We divided our classes into two parts, lecturing and leading discussion for the first twenty minutes and then conducting programming exercises during the second two thirds of the class sessions. In stu-

18 Brian Harvey and Jens Mönig, “Bringing ‘No Ceiling’ to Scratch: Can One Language Serve Kids and Computer Scientists?” *Proceedings of Constructionism*, 2010, 3.

19 Daniel D. Garcia, “The Beauty & Joy of Computing,” TEDxBerkeley (YouTube, March 25, 2015), <https://www.youtube.com/watch?v=ozRovyDwKEM>.

20 Harvey and Mönig, “Bringing ‘No Ceiling’ to Scratch,” 4.

21 Brian Harvey, “The Beauty and Joy of Computing: Computer Science for Everyone,” *Proceedings of Constructionism*, 2012, 33–39.

dent feedback, we regularly heard that we should devote more time to programming during the course. As with Anderson's prior experience teaching programming in digital humanities contexts, providing the right balance between programming instruction and disciplinary contextualization proved elusive.

3.2 NetsBlox

The Beauty and Joy of Computing course at Berkeley and elsewhere uses the Snap! programming language as its language of instruction.²² Snap! is a block-based programming language rather than a text-based language. That is, programming in Snap! involves the manipulation of color-coded blocks that snap together to form units of computation. LEGO, the classic children's construction toy, provided the metaphorical inspiration. The most prominent block-based programming language, Scratch, drew inspiration from LEGO bricks when creating its visual syntax. As Mitch Resnick, the Lego Papert Professor of Learning Research and Director of the Lifelong Kindergarten group at MIT, remarks, "In many ways, Scratch is the digital equivalent of the LEGO construction kit."²³

A significant departure from the standard BJC was our use of NetsBlox rather than Snap! as our programming environment.²⁴ NetsBlox is a visual programming language which builds on Snap! but which extends its functionality to create a more social experience. Ledeczi leads the development of NetsBlox along with Broll and several graduate students at the School of Engineering.

NetsBlox adds three major features to Snap! to enable communication across networks. First, NetsBlox provides a new primitive for message passing. As Broll, et al., explains, "Messages are very similar to Events already present in Snap! Basically, a Message is an Event that contains data payload."²⁵ NetsBlox enables message passing between sprites in a project but also across projects, using a unique "public role" identifier that functions something like an email address. Second, NetsBlox adds call and run blocks for making remote procedure calls (RPCs). The blocks encapsulate the APIs of web services, permitting students to connect with remote data sources in their programs. Third, NetsBlox adds the concept of a "room" with "roles" to projects. A "room" provides a way to con-

²² Harvey and Mönig, "Bringing 'No Ceiling' to Scratch."

²³ Mitchel Resnick, *Lifelong Kindergarten: Cultivating Creativity Through Projects, Passion, Peers, and Play* (Cambridge: The MIT Press, 2017), 46.

²⁴ See <https://netsblox.org/>.

²⁵ Brian Broll et al., *NetsBlox: A Visual Language and Web-Based Environment for Teaching Distributed Programming*, 2016. <https://netsblox.org/NetsBloxWhitePaper.pdf>.

figure a network for a project, allowing users to work on sub-projects in parallel and to connect them (via message passing) using diverse network topologies, allowing for applications like client-server applications.

We drew on these features throughout the course, introducing remote procedure calls during the fourth week while teaching about lists; handling data in the form of lists is the prerequisite for using most RPCs. Right after the midterm, students started exploring message passing, built collaborative whiteboards, and then moved on to a client/server chat application.²⁶ Our favorite exercise in the class was illustrating person-in-the-middle attacks by asking students to encrypt the messages that they passed through the server (using a substitution cipher) and then having our server break the encryption and subtly alter the message.²⁷

3.3 Assessments

Our version of the course had four major assignments. The first was a set of five problems of increasing difficulty. The second was a midterm examination, meant to cover the fundamental programming concepts we taught during the course, from assigning variables to processing lists with higher-order functions. The third was a short paper focusing on a social, historical, or political topic related to computing. We also asked students to develop and present a creative project of their own devising by the end of the semester.

Starting during the second week, we assigned a so-called Parson's Problem every week. For any who are not acquainted with the concept, Dale Parsons and Patricia Haden introduced the idea of creating programming "puzzles" for students to solve rather than asking them to write programs from whole cloth.²⁸ Parson's Problems seek to make assignments more engaging for students while also limiting the range of possibilities (and errors). An apt comparison would be to chess problems, which set up pieces on the chess board to help learners to rec-

²⁶ Brian Broll et al., "A Visual Programming Environment for Learning Distributed Programming," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, SIGCSE '17* (Seattle, Washington, USA: Association for Computing Machinery, 2017), 86, doi:10.1145/3017680.3017741.

²⁷ Broll et al., *NetsBlox*, 6.

²⁸ Dale Parsons and Patricia Haden, "Parson's Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses," in *Proceedings of the 8th Australasian Conference on Computing Education*, vol. 52, 2006, 157–63.

ognize and solve certain kinds of tactical problems. We adapted some of the BJC exercises into Parson's Problems while introducing a couple of our own devising.

3.4 Readings

We also departed from the standard BJC approach with our assigned readings. The primary textbook in the BJC is Ken Ledeen, Harry R. Lewis, and Hal Abelson's *Blown to Bits: Your Life, Liberty, and Happiness After the Digital Explosion*.²⁹ The chapters provide coverage of legal and social topics ranging from copyright law to cryptography. As Brian Harvey noted in 2012, *Blown to Bits* "presents some of the social issues of the Internet era in a style that manages to be both accessible to lay readers and deeply informed by specific technical issues."³⁰ *Blown to Bits* has aged well for a technical book. Its coverage of §230 of the Communication Decency Act of 1996 remains stubbornly relevant as we continue to debate the merits of its 'safe harbor' provisions for free speech in the age of Facebook and Twitter. Still, the undergraduates taking our course were in primary school when the book first came out, making its discussions dated from their perspective. And the text lacks coverage of topics like algorithmic bias and filter bubbles, which have become major topics of conversation today. A second edition (adding Wendy Seltzer as a co-author) was published in 2021, but not in time for us to use in our course.

We picked up the slack with two other books. The first was Martin Erwig's delightful *Once Upon an Algorithm: How Stories Explain Computing*.³¹ As faculty with Ph.D.s in disciplines other than computer science, we wanted to strengthen our presentation of computer science concepts with a strong introductory textbook. While evaluating potential candidates, we ran smack into the problem that most teach computer science via some programming language or another. Eckart Modrow's *Computer Science with Snap!* is unique in its teaching computer science with reference to Snap! but was a bit too advanced for our students—reading the book proved challenging to us!³² Erwig approaches computer science

²⁹ Hal Abelson, Ken Ledeen, and Harry Lewis, *Blown to Bits: Your Life, Liberty, and Happiness After the Digital Explosion*, 1st edition (Upper Saddle River, NJ: Addison-Wesley Professional, 2008).

³⁰ Harvey, "The Beauty and Joy of Computing," 1.

³¹ Martin Erwig, *Once Upon an Algorithm: How Stories Explain Computing* (Cambridge: The MIT Press, 2017).

³² See Eckart Modrow, *Computer Science with Snap!* (Göttingen: Emu-online.de, 2018). See <http://ddi-mod.uni-goettingen.de/ComputerScienceWithSnap.pdf>.

conceptually, discussing difficult topics like types, recursion, and the limits of computing by reference to books and films like Sherlock Holmes, the “Back to the Future” series, the Indiana Jones movies, and the Harry Potter stories.

Our second assigned reading was Claire Evans’s *Broad Band: The Untold Story of the Women Who Made the Internet*.³³ In each chapter of *Broad Band*, Evans presents a vignette or two about women who made significant contributions to computer science. Given her emphasis on the development of networking and communication protocols, Evans’s book provided support for our discussions of distributed computing. But her vignettes also helped us to broaden students’ perceptions of innovators in computer science, highlighting key women in computing history like Ada Lovelace and Grace Hopper as well as contemporary scientists like Wendy Hall.

We also assigned primary source readings like Alan Turing’s “Computing Machinery and Intelligence”³⁴ during our treatment of Artificial Intelligence. Turing’s paper introduces the “imitation game” as a way of evaluating whether a machine had learned to think. Turing also discusses nine potential objections to the notion of thinking machines, which we asked students to consider and evaluate. At the same time, we noted that Turing also introduced problematic opinions, for example, about race relations and Islamic beliefs to his argument. To give students a contemporary perspective on the topic, we paired the reading from Turing with a video by Stephanie Dinkins, new media artist and associate professor in the Department of Art at Stony Brook University. In her video, Dinkins interacts with Bina48, a humanoid robot, on topics ranging from racism to robotic rights.³⁵

Our emphasis on course readings perhaps inadvertently gave away our disciplinary origins. As Amy J. Ko, professor at the iSchool at the University of Washington, wrote on Medium: “As a computer science major, I regularly received explicit signals from CS faculty that not only were books irrelevant, but that they were to be explicitly devalued.”³⁶ As a librarian with a background in the humanities and as a social scientist, books are our stock-in-trade. The Beauty and Joy of Computing curriculum encourages reading, discussing, and

33 Claire L. Evans, *Broad Band: The Untold Story of the Women Who Made the Internet* (New York: Portfolio, 2018).

34 A. M. Turing, “Computing Machinery and Intelligence,” *Mind* LIX, no. 236 (October 1950): 433–60, doi:10.1093/mind/LIX.236.433.

35 See <https://www.stephaniedinkins.com/conversations-with-bina48.html>.

36 Amy J. Ko, “Computer Science Taught Me That Books Weren’t Important. It Was Wrong.” *Medium*, July 2020. See <https://medium.com/bits-and-behavior/computer-science-taught-me-that-books-weren-t-important-it-was-wrong-63ead2fdad7a>.

writing about social issues related to computing, but the “Get Ready to Read” flairs we saw from students on Rate My Professor probably meant the quantity of assigned readings surprised them.

3.5 Final Projects

We dedicated the second half of the semester to the final project. After the mid-term, we lectured on topics ranging from recursion to computational complexity, but we did not test the students on those subjects, wishing to avoid anxiety about these traditionally difficult concepts. Students devoted themselves instead to final projects that they presented to one another during the final week of the semester. Our goal with the final paper and project was to encourage students to contextualize what they had learned about computing to their domains of interest.

The range of projects varied widely from simple games (like Slither.IO clones) to literary and scientific programs. A few students worked on computational poetry generators. An ambitious student implemented and visualized a single linkage clustering algorithm that she had learned in her biology course. A significant number of projects drew on data from NetsBlox RPCs. If students wished to work in pairs, we required them to make use of NetsBlox message passing features. Students who took this team option developed two-player soccer games and also a peer-to-peer Venmo-like application.

We also asked students to argue critically for an ethical or social stance related to computing. Over the four semesters we taught the class, several students selected topics that stood on the boundary between religion and computer science. Students, for instance, considered that Turing had dismissed the soul as a dividing line between human beings and machines too quickly.³⁷ This is perhaps not too surprising as limit concepts in computer science, from artificial intelligence to the possibility of super-intelligence, touch on theological issues. When Anderson was invited as a guest lecturer for another course, “The Ethics of Artificial Intelligence,” students had expressed similar curiosity about what makes human beings distinctive and whether artificial intelligence would become a companion, savior, or devil to humanity.³⁸ In her teaching of ethics to students of computer science, Narasimham emphasizes the human factor in computing errors and considers situations from multiple humanistic perspectives

³⁷ Turing, “Computing Machinery and Intelligence,” 443f.

³⁸ See <https://my.vanderbilt.edu/aiethics/>.

(as well as legal precedents, where applicable) when deciding on their ethicality, calling for more nuanced reasoning rather than just reliance on rigid algorithmic principles. Students in her class also debate freedom of speech issues with respect to the use of technology and the internet to express religious thoughts that may be contentious among certain groups but acceptable to others.

3.6 Reception

The course never took off in the way that we hoped at Vanderbilt. We offered the class over four consecutive semesters and enrollment averaged fewer than twenty students per offering. While the enrollment statistics would have been sufficient for most humanities courses to make, they fell short of expectations at the School of Engineering. Given the surging interest in computer science, any course that does not approach its enrollment cap cannot be sustained.

The most significant mismatch was the institutional setting. The School of Engineering offered the course, but the majority of students came from other schools at Vanderbilt, primarily the College of Arts and Science but also Peabody College of Education and the Blair School of Music. From our initial survey of students' goals for the course, these students recognized the significance of computer science for their future careers, but most had not had the opportunity to study computer science at the high school level. The sequence of courses for majors and minors in computer science would represent too great a commitment and possibly prove overwhelming. But the Beauty and Joy of Computing proved too elementary for most students enrolled at the School of Engineering. (The exception were a few advanced CS students who said they took this course to get a perspective on programming in non-CS fields.) The problem was that our course did not fulfill the graduation requirements required for students at the College of Arts and Science, which meant they took it as a pure elective. For better or worse, the students who took our course wanted to be there as they had no extrinsic reason for enrolling beyond interest in computing (and generic course credit). We speculate that the course would have received more registrations if it had been listed in a college other than the School of Engineering. This raises the question, what would a more contextualized version of the BJC, say for the Divinity School, look like?

4 Computational Thinking and Learning

In the past year, the Computational Thinking and Learning Initiative (CTLI) at Vanderbilt University has explored questions about the contextualization of computer science instruction.³⁹ The CTLI is an institutionally-supported interdisciplinary group of faculty and staff that seeks to expand computer science across the curriculum. The group gathers together researchers from disciplines such as computer science, education, information science, and the humanities to strategize about scaling computational thinking to all students at Vanderbilt (and beyond).

The members of the CTLI are broadly concerned with the question of teaching programming in disciplines other than computer science. As mentioned at the outset of this paper, programming is already being taught in many non-CS contexts at Vanderbilt and other colleges and universities, albeit often not in systematic ways. Michelson and Anderson trained students in the rudiments of the XQuery programming language. In the “Digital Flâneur” course, Calico and Anderson instructed students about data structures when showing them how to encode geospatial data in JSON and also gave them a practical introduction to remote procedure calls by walking through how to send that encoded JSON data using REST calls to CouchDB. Narasimham led an interdisciplinary working group on data visualization for several years while a staff member of the Vanderbilt Institute for Digital Learning. A colleague in Cinema and Media Arts is teaching Processing to students and a faculty member in the Department of French and Italian is training undergraduates in C# to create historical games with the Unity game engine. During the past year, faculty in the digital humanities have begun to put together a curricular sequence to link disparate courses. What can be learned from the way that programming is being taught in these contexts? This question harkens back to a remark by Eric Roberts during SIGCSE discussions about the Beauty and Joy of Computing from 2009: “If we want to produce software artists, it might be useful to experiment with alternative educational strategies similar to those used to prepare artists, writers, and musicians.”⁴⁰

The CTLI sponsors several working groups that bring together faculty, staff, and students to connect computer science education with topics from specific

³⁹ See <https://alo.ees.vanderbilt.edu/ctl/>.

⁴⁰ Daniel D. Garcia et al., “Rediscovering the Passion, Beauty, Joy, and Awe: Making Computing Fun Again, Continued,” in *Proceedings of the 40th Acm Technical Symposium on Computer Science Education*, 2009, 65–66.

domains. These working groups are exploring how students can learn to program in NetsBlox with reference to the problem of climate change, on the one hand, and textual analysis, on the other. In a related ongoing study, Narasimham is collaborating with Gautam Biswas (Professor of Engineering and Professor of Computer Science, Computer Engineering, and Engineering Management, as well as member of the CTLI) and researchers from the department of computer science to explore how high school students may use NetsBlox to learn both computing and physics concepts.

Anderson and Mark Schoenfield, Professor of English in the College of Arts and Science at Vanderbilt, coordinate the “Modalities of Textual Analysis” working group with Brady and Broll. The group is co-sponsored by the CTLI and the Vanderbilt University Libraries. The libraries support the activities of the working group through its endowed Buchanan Fellowship program, which provides fellowships for undergraduates to work together with librarians and faculty on topics of strategic value to the libraries.⁴¹ During the spring and fall semesters of 2019, fellows explored the affordances of NetsBlox for conducting textual analysis of ProQuest’s British Periodicals database. The database consists of the full text of nearly five hundred British periodicals and several million articles, dating from the seventeenth to the twentieth centuries. To make it possible for students to search across these articles for keywords and phrases, the principals put the XML documents into a BaseX database. Broll built an integration into NetsBlox to send and receive requests to BaseX while Anderson built a small library of blocks to encapsulate the XQuery requests. The blocks allow students to look up information from the British Periodicals without having to write any XQuery code, send REST requests, or parse JSON data. Results from British Periodicals come back as a list of lists, which students can pass directly to other functions in their program. By the end of the semester, students were able to carry out the main steps of text analysis, from gathering a dataset, cleaning the data, applying various natural language processing operations, summarizing the results, and producing visualizations—all within NetsBlox.⁴²

⁴¹ Library Communications, “Buchanan Gift to Endow Vanderbilt Library Fellowship Program,” *Library News Online*, January 2018. See <https://newsonline.library.vanderbilt.edu/2018/01/buchanan-gift-to-endow-vanderbilt-library-fellowship-program/>.

⁴² For more information about the pilot, see Brian Broll et al., “A Blocks-Based Introduction to Text Analysis,” Snap!Con 2020, July 2020. See <https://forum.snap.berkeley.edu/t/short-talk-a-blocks-based-introduction-to-text-analysis/2576>.

5 The Telos of Block-Based Programming

There is ongoing debate about the utility of introducing students to programming through block-based environments.⁴³ How does starting students out in a block-based environment affect their long-term success as programmers?

On the one hand, the visual metaphor of programming by snapping blocks together seems to strike an intuitive chord with students while also encouraging a sense of playful exploration. The sense of playfulness shows the line of descent of these environments from Seymour Papert, whose “constructionism” encouraged children to learn by assimilating examples from a domain and then playing with them.⁴⁴ In a talk at Digital Humanities 2019 in Utrecht, Anderson and Lynn Ramey, professor of French at Vanderbilt University and the faculty director of Vanderbilt’s Digital Humanities Center, argued that block-based environments, when suitably adjusted, could help digital humanists learn to code more effectively.⁴⁵ While preserving the ludic experience of programming with blocks, they suggested creating a version of NetsBlox with RPCs for popular digital humanities services, sprites that represent relevant objects of computation, and a curriculum aimed at humanists.

A drawback to using block-based programming environments is the perception of them as programming cul-de-sacs, if not dead ends. While learning to program in block-based environments may speed the learning process for students by reducing the possibility of making syntax errors and encouraging an exploratory and tinkering style of coding, it seems inevitable that students will eventually need to transition to text-based languages like JavaScript, Python, or XQuery. But can that transition be delayed? The aspiration of making software environments “low threshold” but “high ceiling” is attributed to Papert, inventor of the Logo programming language. As Brad Myers, Scott E. Hudson, and Randy Pausch put the idea, “The ‘threshold’ is how difficult it is to learn how to use the

⁴³ Zhen Xu et al., “Block-Based Versus Text-Based Programming Environments on Novice Student Learning Outcomes: A Meta-Analysis Study,” *Computer Science Education* 29, nos. 2–3 (July 2019): 177–204, doi:10.1080/08993408.2019.1565233; Yue Hu, Cheng-Huan Chen, and Chien-Yuan Su, “Exploring the Effectiveness and Moderators of Block-Based Visual Programming on Student Learning: A Meta-Analysis,” *Journal of Educational Computing Research* 58, no. 8 (January 2021): 1467–93, doi:10.1177/0735633120945935.

⁴⁴ Seymour A. Papert, *Mindstorms: Children, Computers, and Powerful Ideas*, Revised edition (Basic Books, 2020), 133.

⁴⁵ Clifford B. Anderson and Lynn T. Ramey, “Thinking Computationally in the Digital Humanities: Toward Block-Based Programming for Humanists” (Utrecht: DataverseNL, December 2019), doi:10.34894/7QDKAP.

system, and the ‘ceiling’ is how much can be done using the system.”⁴⁶ Proponents and critics alike typically agree that block-based environments are “low threshold.” In our classrooms, we have witnessed the delight of non-programmers experimenting with stacks of coding blocks in Snap!, generating shifting graphical effects on the stage. Playing with graphics of a similar sort in C++ or Java would be considered an advanced exercise. But do block-based environments provide sufficiently “high ceilings”? A motivation for creating the Snap! programming language was a concern that Scratch had, in fact, oversimplified the language. The goal of creating Snap! was “to extend the power of Scratch to more advanced computer science ideas, suitable for an undergraduate CS course.”⁴⁷ While professors of computer science have used Snap! to demonstrate advanced concepts in computer science,⁴⁸ the attitude persists among students that block-based languages are intended for children or rank beginners. In a computer science course that used Scratch during the first two weeks of an introductory course before switching to Java, the instructors reported that “some students associate Scratch with pre-university education, thus they consider its use childish and ‘non-serious.’”⁴⁹ An active area of research in computer science education is when and how best to transition students away from these environments. “Many uses of block-based tools in formal educational contexts presuppose that such tools will help prepare students for later instruction in text-based languages,” write David Weintrop and Uri Wilensky. “This transition is often a part of the larger computer science trajectory where block-based introductory courses are intended to prepare students for the transition to professional, text-based languages.”⁵⁰ If students eventually need to leave the block-based environment behind as part of their programming journey, they may be forgiven for wondering whether they should have started there in the first place.

46 Brad Myers, Scott E. Hudson, and Randy Pausch, “Past, Present, and Future of User Interface Software Tools,” *ACM Transactions on Computer-Human Interaction* 7, no. 1 (March 2000): 6, doi:10.1145/344949.344959.

47 B. Harvey and J. Mönig, “Lambda in Blocks Languages: Lessons Learned,” in *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, 2015, 35, doi:10.1109/BLOCKS.2015.7368997.

48 Eckart Modrow, *Computer Science with Snap!*

49 José Alfredo Martínez-Valdés, J. Ángel Velázquez-Iturbide, and Raquel Hijón-Neira, “A (Relatively) Unsatisfactory Experience of Use of Scratch in CS1,” in *Proceedings of the 5th International Conference on Technological Ecosystems for Enhancing Multiculturality*, TEEM 2017 (New York, NY, USA: Association for Computing Machinery, 2017), 1–7, doi:10.1145/3144826.3145356.

50 David Weintrop and Uri Wilensky, “Transitioning from Introductory Block-Based and Text-Based Environments to Professional Programming Languages in High School Computer Science Classrooms,” *Computers & Education* 142 (December 2019): 2, doi:10.1016/j.compedu.2019.103646, internal references omitted.

In a Viewpoint piece in *Communications of the ACM*, David Weintrop questions whether students must all eventually transition away from block-based programming: “If it is possible to do significant, non-trivial tasks in block-based environments, should we still expect all learners, even those not likely to pursue a degree in computer science, to learn text-based programming?”⁵¹ In Anderson’s use of NetsBlox in textual analysis with undergraduate students, he has observed a different phenomenon. The students who take part in the working group come with different levels of exposure to computer science. Some students arrive with a major or minor in computer science while others have never had any formal or informal coding experience. To level the playing field, we start out the entire group in NetsBlox. The functional programming structures that we tend to favor in our programming exercises are sufficiently new to students with previous coding experience that everybody learns together during the first few weeks. As the semester progresses, we encourage students to complement NetsBlox with other programming languages and frameworks. Experienced students may, for instance, export a list from NetsBlox as a CSV file to a natural language processing library in Python or a visualization tool to carry out a procedure not yet achievable within NetsBlox. Students can also re-import that data to NetsBlox, converting CSV or JSON into a list of lists. By fostering a porous computational environment that makes data readily transferable, the question of when and how best to transition to a text-based environment seems less pertinent. Rather, students come to appreciate text-based programming tools as well as other visual development environments alongside NetsBlox. In the end, our goal is not to promote either modality, but to encourage a “human-centered” design approach to teaching computation to humanists where one size does not fit all.⁵²

In “Design Principles for Tools to Support Creative Thinking,” Mitchel Resnick et al. added the concept of “wide walls” to Papert’s “low threshold” and “high ceiling.”⁵³ What they mean by “wide walls” is that a creative learning environment should serve many purposes and not be tailored toward predeter-

51 David Weintrop, “Block-Based Programming in Computer Science Education,” *Communications of the ACM* 62, no. 8 (July 2019): 25, doi:10.1145/3341221.

52 Clifford B. Anderson et al., “Human-Centered Computing for Humanists: Case Studies from the Computational Thinking and Learning Initiative at Vanderbilt University,” Short Paper, DH2020, July 2020 (Not held due to COVID-19). See https://dh2020.adho.org/wp-content/uploads/2020/07/681_HumanCenteredComputingforHumanistsCaseStudiesfromtheComputationalThinkingandLearningInitiativeatVanderbiltUniversity.html.

53 Mitchel Resnick et al., “Design Principles for Tools to Support Creative Thinking,” January 2005, 3, doi:10.1184/R1/6621917v1.

mined uses. “The design challenge is to be specific enough so that users can quickly understand how to use the features (low threshold),” they write, “but general enough so that users can continue to find new ways to use them (wide walls).”⁵⁴ The metaphor of “wide walls” provides a different frame for students’ use of technologies to supplement NetsBlox. Given that developers can easily incorporate new remote procedure calls (RPCs) to NetsBlox, encouraging students to explore related technologies also helps to identify potential integrations.

For instance, Broll added an integration for ParallelDots, a commercial natural language processing service, to NetsBlox so that students could, for instance, analyze the sentiment of strings from texts.⁵⁵ After experimenting with using the service to discern the sentiment of eighteenth and nineteenth century texts, students also came to appreciate the limitations of applying a NLP model evidently trained on contemporary internet discourse (like YouTube comments and restaurant reviews) to periodical literature from a different time period. Later during the semester, Broll incorporated Stanford’s CoreNLP⁵⁶ as another RPC for natural language processing. CoreNLP provides a wide variety of annotators, including a sentiment analysis tool. This integration permitted students to compare the results of natural language toolkits, sharpening their critical sense of their respective uses and limitations.

Anderson is also experimenting with a remote procedure call to link NetsBlox with Wikidata for easy access to cultural heritage data.⁵⁷ As mentioned above, Anderson and Calico introduced graduate students to Wikidata in their “Digital Flâneur” seminar. Students took quickly to the user interface for data entry, but experienced greater difficulty with the query interface and also had trouble converting data from Wikidata into other formats. To address these issues, Anderson and Broll have collaborated on a remote procedure call that allows NetsBlox users to send parameterized SPARQL queries to the Wikidata Query Service and to receive data back in the form of a NetsBlox list of lists.⁵⁸ If Anderson and Calico were to teach the course again, they would be able to

⁵⁴ Ibid., 4.

⁵⁵ Broll et al., “A Blocks-Based Introduction to Text Analysis.”

⁵⁶ Christopher Manning et al., “The Stanford CoreNLP Natural Language Processing Toolkit,” in *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations* (Baltimore: Association for Computational Linguistics, 2014), 55–60, doi:10.3115/v1/P14–5010.

⁵⁷ Clifford B. Anderson, “Teaching with Wikidata: A Case Study,” *Wikiconference North America 2018* (Columbus, 2018).

⁵⁸ Ibid.

use NetsBlox as an environment for much of their *flânerie*, permitting students to spend more time analyzing the different ways of representing urban environments digitally while spending less time on software engineering concepts like source control systems and data structures like JSON and XML.

The customization of NetsBlox into a platform for playfully exploring with computational techniques in the digital humanities while achieving digital humanities *tasks* will require careful balancing. From the perspective of professional tools, NetsBlox may seem too barebones. A Python, Node, or R library exists for every conceivable computational task. But Snap! and NetsBlox come only with a few libraries of routines. A reason for this parsimony is the desire to maintain simplicity. An additional reason is that Snap! and NetsBlox seek to encourage students to explore *how* to build these libraries for themselves. For this reason, they do not come prepackaged with searching and sorting routines, but students can learn to instantiate their own algorithms for this purpose. As Mary Angelec Cooksey remarks about her use of Raspberry Pis in the classroom, “In a course like the Philosophy of Computing, teachers want students to see what is going on with computing at its most basic level.”⁵⁹ As NetsBlox becomes capable of solving more and more *tasks* in the digital humanities, we must also be careful to maintain this kind of pedagogical minimalism.

The ability of educational environments like Processing⁶⁰ successfully to span the gap between learning platform and professional programming tool in new media encourages us about the potential of NetsBlox to serve similar ends in the humanities. As the number of RPCs for NetsBlox increases, students of religious studies can already carry out a wide range of context relevant tasks: mapping the locations of houses of worship in a particular region, identifying people or events mentioned in sermons, and searching for trending religious topics on Twitter, among much else. As students and scholars of religious studies and theology increasingly engage with the digital humanities, we hope that these activities can be integrated into a version of the Beauty and Joy of Computing that faculty may use to teach computation in a context-relevant way in seminary and divinity school classrooms.

⁵⁹ Mary Angelec Cooksey, “Teaching the Philosophy of Computing Using the Raspberry Pi,” in *Quick Hits for Teaching with Digital Humanities: Successful Strategies from Award-Winning Teachers*, ed. Christopher J. Young et al., Illustrated edition (Bloomington, Indiana, USA: Indiana University Press, 2020), 56.

⁶⁰ Casey Reas and Ben Fry, “Processing: Programming for the Media Arts,” *AI & SOCIETY* 20, no. 4 (September 2006): 526–38, doi:10.1007/s00146–006–0050–9.