Transcript

[0:01] (music)

Derek Bruff: [0:05] This is Leading Lines. I'm Derek Bruff. I am starting to believe that one of our unstated goals on Leading Lines, is to interview each member of the podcast producer team. In past episodes, we've interviewed John Sloop, Melissa Mallon, former producer, Ole Molvig and well, me. Today we'll continue the trend with an interview with Cliff Anderson, Associate University Librarian for research and digital initiatives here at Vanderbilt and another Leading Lines producer. I really like how this interview came together. Cliff has been teaching a new course called, "The Beauty and Joy of Computing" for a few semesters, now. It's an introduction to computer science and computational thinking aimed at students who aren't majoring in computer science. This semester, another Leading Lines producer, Gayathri Narasimham, Research Assistant Professor in Electrical Engineering and Computer Science, has started teaching it. Gayathri thought it would be interesting to interview Cliff about his experiences designing and teaching the course. I thought that was a great idea and I'm happy to present their conversation here on Leading Lines.

[1:09] In the course, Cliff and Gayathri use NetsBlox as their programming language. It's a blocks-based language like Scratch or Snap, designed to teach computing concepts visually, without having to work through lines of code. In the interview, Cliff discusses the pros and cons of this approach to teaching computer science. And he shares a little about his interdisciplinary background as a scholar of religion, turned librarian, turned technologists. (music)

Gayathri Narasimham: [1:37] So I want to ask you, what got you interested in NetsBlox on which this course is based?

Cliff Anderson: [1:43] So I guess it was kind of a long story, in a sense, that it goes back to like how I got interested in programming languages and the teaching of programming languages and maybe the teaching of computer science and programming outside of computer science as a discipline per se. So, like a lot of people in the eighties, I grew up with a personal computer and learned how to program in the basic programming language. So, the idea of learning as you go and experimenting and trying out new programs and sort of seeing how they worked. And maybe even getting inside of programs that have been written and editing their source code and seeing how things would change, was something that I was kind of used to. That idea of like tinkering and playing with code was something that was very much a part of my own education in software, but also, I think it opened up my mind to sort of the idea of educational programming environments. Because in the eighties, there was a large blossoming of efforts to make programming more accessible to children. And I think I've benefited from that. So you know, you kind of fast forward to perhaps after I got my PhD and I'm working in a library, and I was given responsibility to build a digital library. And so I had to reactivate a lot of the programming that I had learned as a kid and formalize it. I'd, I'd never really stopped. But, you know, when you're given a goal then you, you'll learn a lot more to achieve that goal.

[3:17] So I worked with a bunch of colleagues and eventually started really concentrating a language called XQuery, which is an XML programming language really great for handling large quantities of text. But also what I needed to do, because there weren't a lot of XQuery programmers, was to think about how to train my colleagues to use XQuery effectively, as well. So I had been teaching myself, but then was also teaching colleagues how to program. So that led to an interest in what are the best ways to communicate the teaching of programming. How do you do that effectively? They're definitely some ways where it's not so effective, just like throwing up a bunch of code and just saying like now copy this and paste that. That's not too helpful. So the idea of sort of breaking down code into units of meaning and exploring, what they do. And again, like tinkering with them with them to see the different outcomes was, was a way that I'd sort of integrated some of that earlier learning from those educational programming environments into my practice of teaching XQuery. And that eventually led to a book a colleague and I wrote together, Joe Wicentowski, called, *XQuery for Humanists,* which we basically encapsulate a lot of our learning. Joe had a similar experience teaching colleagues at the State Department how to master this programming language. So when faculty here, in particular, Professor Ákos Lédeczi and Doug Schmidt came to me and said we might want to do something like an introductory programming course for non-majors or potential majors. I thought that'd be really interesting. Of course, I knew there'd be big challenges there because I had to just jump up another step, which is to say, you know, I needed to make sure that I was giving a well-grounded introduction to computer science, not just practical programming. And I needed to make sure that we're doing this with the latest tools and sort of best practices for teaching computer science. Fortunately, a lot of that had been already encapsulated in this Beauty and Joy of Computing curriculum and we can talk about that some more. But I was able, as you say, to adapt that for an audience here at Vanderbilt and then teach it for three semesters. And I'm really glad that you're teaching it now and continuing with that effort. So that's a bit of a long-winded introduction, but let's maybe dive into aspects of that.

Gayathri: [5:33]Yes. Yeah, I would definitely like for us to do that. I want to come back to a couple of points that I wanted to, as I was listening to when you said this. One is you had experimented with programming from a very young age. And, and that's not intuitive to a lot of people, especially the, the audience of this Beauty and Joy, the students who are taking the Beauty and Joy of Computing class. And that builds on my second question, which is, it is aimed towards the humanists or its aim to non-computer science majors. And so for them, it is not intuitive at all. It's not completely, it's not something that, that is in their repertoire of thinking. And so they're learning some of these skills from scratch and I meant scratch as a word, not as a code. So I want to ask you about how you built this course, Beauty and Joy of Computing. The curriculum that Berkeley has, upon which your course is based, does not have this comprehensive view of computer science. You for example, in your syllabus, you've used Martin Erwig's book*, Once Upon an Algorithm: How Stories Explain Computing* and you've used Claire Evans', *Braodband: The Untold story of the Women Who Made the Internet*. And both of those books, along with the additional readings, give a perspective outside of just a very programming focused environment. It gives a lot of context to the kinds of ideas that are popular, the concepts that are important to understand computer science and computing principles. So I want you to talk about all of that if you can.

Cliff: [7:22] Okay, so let's maybe just start with The Beauty and Joy of Computing. So this is an effort, I'm sure there are many more people involved than the ones that I can name right now. But, you know, I'm thinking primarily of Dan Garcia, Brian Harvey, Tiffany Barnes. These are computer science educators with long experience thinking about how to impart computer science principles to wide audiences and to make sure that it goes beyond sort of the traditional group that might immediately decide to major in computer science. And so their curriculum has been incredibly successful and not only at the undergraduate level, but also at high schools. So I very much benefited from seeing what they had done and thinking about how to adapt that into the context at Vanderbilt.

[8:05] And one of the things I have to say, just on a personal anecdotal level aside, I met Dan Garcia at the SIGCSE conference, it's called the Special Interest Group and Computer Science Education. It's, it's the largest computer science educators conference. And when I told him I was teaching this class, he give me a big hug, which I felt like was terrific because I was really worried a bit that I was maybe stretching the boundaries of the Beauty and Joy of Computing curriculum too far. But I think they recognize that it's, it's a kind of set of principles that are about imparting this knowledge in a friendly and accessible way. A way that really stresses the fun and the pleasure of computing, as much as the science and rigor, but they're both there and I think in nicely balanced parts. So, but one thing, we have a different format here at Vanderbilt. This course had been in two meetings without labs. And so in order to be able to, to teach this course, I had to make some adjustments just to fit our semester and to fit the schedule. And so walking through it, I also thought it would be helpful to have some textbooks that would focus also not only the social issues, but the one that, that has been traditionally used with The Beauty and Joy of Computing is *Blown to Bits: Your Life, Liberty and Happiness After the Digital Explosion* by Hal Abelson, Ken Leedeen, and Harry Lewis. That book was written in 2008. And it's a, it's a fantastic and excellent book. But just like you could imagine, the difference is 12 years later, in terms of where we are as a society, the digital issues that we're looking at, things that change. There is a second edition of that book that is coming out. Wendy Seltzer is also now a coauthor on that. And I was eagerly waiting for it to come out. And they said it was taking a little bit longer than we expected. And I think it's going to be a fantastic book in its second edition.

[10:00] But in the meantime, I added, as you mentioned Martin Erwig's book, *Once Upon an Algorithm: How Stories Explain Computing*, which I found extremely helpful because it's hard to find an introductory computer science textbook that doesn't focus on some particular language. So you could find one that might be like for Python or JavaScript, Pascal, whatever you, you know, you could name. But what I really wanted was one that just focused on the broad concepts because obviously I was going to be teaching this in conjunction with the visual programming language that we have in the "Beauty and Joy of Computing." We could talk a little bit about Snap, NetsBlox, Scratch maybe later. So that book was, was great because it helped me to just identify issues like what is a type? What is a control structure, without actually reference to code. I mean, it uses some pseudocode, but the idea is just to bring mental level of abstract, or even what is abstraction. That then I could highlight in the examples that we did in terms of practical coding exercises. And there wasn't a mismatch between what the book was teaching and the exercises that I was using in class.

[11:06] I think it's a really excellent book. And students seem to like it. It's, it can be demanding at times because it really is rigorous. But I think it sets up people well, if they decide to take a second computing course, they'll understand the concepts extremely well. And so it's really more that they have mastered the syntax and maybe additional semantics or whatever is involved in a particular language you're using. I would also say that the third book, which you mentioned I've used, Claire Evans' book, *Broadband: The Untold Story of the Women Who Made the Internet* was really important for another reason. I mean, one of the goals of this course was to expand the range of people that decide to take computer science, whether they use it in a different major, or they take a minor in computer science or decide to major in it. We wanted to expand access. And traditionally that's meant getting more women and minorities to take computer science. And so I think Evans' books was really important because it brought out the contributions that women have made throughout the history of computing at every stage, you know, in so many different ways. And like, like every story, some were extremely successful in their careers, others less so, others maybe fallen off the map and rediscovered. But they're great human stories. And I think one of the things that was really useful about that, using those stories in class was just to say, you know, we know a lot of stories about Alan Turing. Maybe we know some. I don't know how many people know all those stories and we do, I did put on, for example, some of Turing's work. But I wanted people to hear about stories that were told for perspectives that weren't as commonly known. And also focused on the contributions that women have made which had been extremely substantial. And so I think that was helpful and just sort of broadening the perception of what computing is, which was again, one of the goals of the course.

Gayathri: [13:02] Anecdotally, I want to share something today we discussed *The Longest Cave*. So it was about Patricia Crowder and how she discovered the links between the cave system in Mammoth Caves. And I think the story is really about how she discovered the cave, then took that, and put it into her programming and developed the algorithms and tools for mapping that out. The only, you know, the whole, the whole class, we were talking about this. This is an interesting story. It deals with these concepts very nicely. And I think overall, both with Martin Erwig's book and the Claire Evans' book, we are looking at the chapters and it's an easy read in many ways. And by giving us very good conceptual knowledge about all the different kinds of concepts that we need for the class itself. So I think those are great selections. Thank you.

Cliff: [14:00] Well you know, I'll say that I mentioned using computers early on, but when I was, you know, it was in the seventies, my father was taking a course at NYU and he got access to a mainframe computer. And I used to sneak onto the mainframe under his account, and then play Adventure online and it was the first computer game I ever played. And I just couldn't believe it. And so I used to play that game all the time. So to read about sort of the human side of how that story got written and you know, it was, it was really fascinating to come back around. So anyway, yeah, so I think, you know, stressing the humanity of computing and also just expanding like their perceptions of what it means to be a computer scientist. Definitely borrow what I was trying to do and what you're trying to do.

Gayathri: [14:50] Yeah. What were your experiences teaching this course? I mean, it's not the same as the, BJC, The Beauty and Joy of Computing that Berkley does. Yours is more well-grounded in the conceptual, the concepts, for example, to me. I mean, let me just say I haven't taken the BJC, so for me, I feel like this is really well-grounded, especially when it comes to teaching a course in a semester, that you have all the background knowledge about all the concepts. So give us some experiences, give us some of your insights into teaching. What were your experiences? What do you think you would have done differently?

Cliff: [15:34] Well, so I think one of the things that I really enjoyed about this, is the work that has been put into the visual programming's aspect of it. So just for people that are listening to this and haven't really played around with the technologies that we're talking about. So a lot of people know Scratch. And, and so that comes out of work at the MIT Media Lab in 2017, sorry, 2007, I believe they launched the first version of Scratch, I think was Mitch Resnick's Lifelong Kindergarten program. The idea was to make blocks that you could connect together and snap together like Lego essentially. And then by making bigger and bigger units, you could create really amazing programs. And I think, you know, it took off, the success of Scratch is just undeniable. It's been widely, widely used in lots of elementary schools, middle schools. But there are some limitations for using Scratch to teach computer science. And so, you know, when you think about trying to design a language that will be as some of our colleagues like to say, "low threshold but high ceiling," in a sense that, easy to learn, but you don't want it to be like a stopping point. You want them to actually then take what they've learned and move into a textual language, because most programming's still these days is going to be text-based.

[17:02] So one of the things I think in teaching this course is how do you make sure that students can make that transition well, but yet, try to make their entry point into computer science such that they have a really good experience and it feels sort of intuitive to them in this way that Scratch really makes that possible? So again, like building on the work of Brian Harvey and Jens Mönig, who have collaborated for a long time on building Snap. One of the things that they did was to basically take the syntax of Scratch, all those nice blocks, and model it onto the semantics of Scheme. Scheme is a variant of Lisp. And it, you know, it, you can teach all these wonderful concepts in computer science using Scheme, so now you can teach it using Snap, with those blocks. And so then, as you know, we had a prior show on NetsBlox, and some of the people listening to us may have listened to that. But just to encapsulate that, NetsBlox basically takes two innovations and sort of puts them on top of Snap. Those two being message-passing, the idea that you can pass messages peer-to-peer and develop applications in which students can develop programs that communicate with each other. Also, the idea of RPCs remote procedure calls, in which students can call out to some place on the internet, like get the weather or a map or whatever, and then bring it into their programs so that you're able to communicate with, but with peers, as well as with data sources that are external to your program. And that really makes programs that are very rich. So, you know, this is where I think, again, like Ákos Lédeczi and Brian Broll, who is a research scientist here, that are working on NetsBlox, have done a great job of making it possible for students to jump in and with just a few blocks, you know, develop a weather map or a guessing game with images from various cities and things like that.

[18:53] So I think, you know, coming back to your original question, one of the things that I think was really nice for me was, every time I tried to explain a computer science concept, I also then had to think about how to model it so that you had a kind of visual output that was, you know, would make students feel like they really accomplish something. Because it's easy and it's certainly something that I did. For example, to write a program that computes factorials. So you know, so that's kind of a fun way to teach recursion and people, you know, fun, I mean, I enjoy it, but yeah, it's, it's, it's a standard way to teach recursion. But it doesn't have a natural visual compliment that you see on the stage. And so one of the things I thought about was, how do you actually model these blocks and what their computing and then the visual representation that you see. I think that was the toughest thing for me because there was a tendency for me to just want to like move into having blocks put together that will teach a concept. But it didn't actually visualize anything, it just had an output. And, you know, typically when you program, you know, especially if you're thinking like I do, in a more functional environment than an imperative environment, that is to say in an environment in which you're working with pure function, so you're thinking about the inputs and then you're thinking about, okay, that produces these sets of outputs, but without any side effects. Then you're just thinking, okay, well I'm going to pass in these particular values and I'm going to get out this list or this integer or string. But, but to really make this successful for teaching purposes, you have to think beyond that and think, okay, what would this look like? What kind of visual display? Could this make a spiral or could this make a set of interlocking squares and things like that? So how the output looks to people beyond the particular values, that's really, really important. And I think that's something I had to think through for each example. And I don't always know that I did a great job of that, but I mean, I tried.

Gayathri: [20:53] I particularly like your examples. We've been working through those and looked at other examples, as well. The key aspect of NetsBlox is the visual programming that is helpful. Now I'm curious, you went from a syntax-based programming culture to this visual block-based programming and also for our audience, so I would like to remind you to listen to Cliff Anderson's interview with Ákos Lédeczi, who was one of the developers of the NetsBlox Program, here at Vanderbilt University. So I will ask you about that. So this is a very different way of thinking. And to be able to use this visual block-based programming to provide some kind of like really cool outputs. That's, where I think the, the tension is. I think that's where you really need to up the ante, so to speak. So going from the simple blocks, to producing something that combines these blocks in very unique ways and, and provides that output. How did you feel that? Did you write it out in syntax? Think about in syntax and then?

Cliff: [22:01] Well, so I think when you're, so, so I mean in a way like that, you know. So, so they both have their own syntax, of course, but like one is based on text and the other is based on this grammar of visual objects that you're connecting. But I think your question is a really good one, in a sense that, you know, when you're organizing blocks, one of the things that is a criticism of the blocks approach, is that your screen quickly get overwhelmed by lots of graphic objects. So, so one of the things that I think as we move along the semester, we had to think about is, how to organize your code. And this is, this is again, this is really important lesson for anybody learning to program because your tendency, when you first learn, is to make some huge function that's like a hundreds of lines long. I mean, this is a kind of a joke in programmer culture, but it does happen where you've just got this one thing that's really impossible to figure out what it does in between, you know, it might work, but if you try to change a single line, you could break it in all these unexpected ways and nobody knows how to fix it. So keeping your code modular and thinking about how to reduce the level of complexity, so that you kind of focus on like each of these blocks, just one thing at a time and then they work in concert together to produce some uniform output. So again, one of the things that I think is really distinctive about Snap is that it was actually originally, originally called "Build Your Own Block," BYOB. But as I understand, like some teachers didn't like the joke of the BYOB because it also stands for, "bring your own beer." So they changed it to Snap.

[23:36] But the underlying idea that, you know, we should really be thinking in terms of functions so that you, you create a block and you encapsulate the sort of functionality inside of that block. And it may happen inside that further blocks that encapsulate further functionality. But that you sort of tame complexity by working in those sets of functions that then get composed together to produce a result. That I think, is one of the things that I was always trying to communicate to students. And I think it's, it's, it's a software engineering lesson that everyone needs to learn, including myself. The technical term, of course, is you build something that's really hairy and does too many things at once, then you want to refactor to something that's simple. And so a lot of the classes, actually getting something to work the first time and then thinking, OK, now that we did this, how can we do this more elegantly? And so that's almost the, the beauty side of it is. It's not just that the beauty of the output, which can be beautiful, even if you're working on fractals, but it's also the beauty of the code. And so you're thinking, okay, I got this to work, but it doesn't look too beautiful. How can I make this more elegant in its approach?

Gayathri: [24:46] That's a great point about elegance, which is the neatness of the code, the ability to make it modular, like you said. There can be recombined in easy ways and not, not be cumbersome or bucky on the screen. It definitely is a big deal that block based programming, because you're really pulling in those blocks, yes. We've also been working with the computational thinking and learning institute, with NetsBlox. So where do you see NetsBlox in future? And based on your classes, you've taught three semesters now, The Beauty and Joy of Computing course for three semesters, have you had students who are interested in pursuing? Have you had like feedback about the course motivating students to go beyond these block-based programming classes, to actually going into syntax or learning coding in other language? And also, how does that segue into the computational thinking aspect?

Cliff: [25:47] So I, you know, I don't know because we haven't done any formal study on the number of students that continue on to do a computer science minor or major, or take other courses in computer science. But anecdotally, I think quite a few have, you know, it would be interesting to know, and I think this is an area of active research, how you move from the visual programming languages to the text-based languages. And I think some studies have shown that that people that start out with visual languages do better at least over some period of time in the text-based languages. But that I think is, that's an area of active computer science research. My own interest is a little bit different in the sense that I am interested in teaching coding to humanists and try to think about the best way to do that. And so, you know, the computational thinking learning initiative that we're both apart of, is thinking about computing across the curriculum in, in ways that it's not just about computer science, but as a discipline. So I think in the digital humanities, which is where I spent a lot of my time, one of the challenges is, we all want to, we want to compute for practical purposes. And I think that, that makes a lot of sense because we have research questions that we'd like to answer. We have data we'd like to analyze. And so there are tools that allow you to do that and some of them are really nice off-the-shelf, kind of almost visual tools, themselves. Like you could think about it, like Tableau, for example, for data analysis, fairly easy to use and very popular because it produces great visualizations without having to do a lot of hand-coding. There are also ways to do R and Python that are not too code heavy and people can pick them up and run with them and sort of learn as they go. But I'm also interested in, in thinking about OK, how do we sort of deepen our engagement with computer science concepts so that we can stay in dialogue with computer science researchers, who are thinking through these languages and, and help to think about how the humanities perspective can inform the development of computing paradigms, new languages, and things like that? I think there's, there's interests both sides, but I think in order to be able to do that, it's just this kind of two worlds problems. You have to be able to understand both sides well enough to be able to communicate across, what can sometimes be substantial barriers, even if there is goodwill on both sides.

[28:20] So for me personally, I'm working with a group of colleagues, including Lynn

Ramey and Corey Brady and Brian Broll on thinking about developing versions of NetsBlox that work with data and compute results that are of interest to humanists. So right now, we're focusing on text mining. There's an interesting project going on with the professor, Mark Schoenfield, who's interested in analyzing large quantities of 18th and 19th century British periodicals. And so the question we have that's a research question is, you know, we want to get the answers to him and we're going to be using tools, for example, like XQuery. We're going to be using things like Apache Spark. Gonna be writing code in Scala. These are all fairly high threshold technologies, in the sense that it takes a while to learn them to become productive in them. And we are working with faculty and students to get them up to those thresholds so they can work in those environment and we can answer these questions. But at the same time, we want to see, can we create an environment in NetsBlox that will allow people to understand these concepts without having to struggle with setting up clusters on Amazon and thinking about the distributed? Could we, could we make the conceptual approach simpler? So can we make the approach to these technologies simpler so that the concepts rise more to the floor rather than all the technical scaffolding that you need to actually accomplish the goal? So we presented, Lynn and I, on some of our early work, to develop this kind of environment and curricula at the digital humanities conference that took place in Utrecht last summer. We've got a proposal and we'll see if it's accepted for the upcoming digital humanities conference in Ottawa. But it is an ongoing research area which is to say, you know, if we can really think about the best ways to teach undergraduates about computer science across the curriculum, how about graduate students and faculty who are doing digital humanities?

Gayathri: [30:21] That's very exciting, Cliff, and good luck with your efforts in developing that program. I hope your proposal is accepted.

Cliff: [30:27] Thank you, Gayathri.

Gayathri: [30:27] So this has been great, I love talking with you about The Beauty and Joy of Computing course, because obviously that's very relevant to me being, teaching this course this semester. But also your larger background and your input, your insights into developing not just this course, but your interest in programming and how you came to be able to teach this course and so on. Thank you so much for talking with me. And before we end, I would like to ask you this question that we ask all our interviewees. So we've talked a lot about digital technologies, and this podcast is about EdTech. So what is your favorite analog technology?

Cliff: [31:10] So the first time I answered this question, when we all sat around together, I said it was a ruler. Because I like to, to write very straight lines when I'm underlining books, which I think people thought was fairly ridiculous. So now I'm going to say a pencil, which is the other side of that because I also don't like to markup books in any permanent way. I'd like to be able to erase what I write. No, but I also, you know, I have discovered and I think a lot of people have, that if you really want to remember something, actually writing it down, long hand can be better as a mnemonic device than actually typing it out. So I regularly carry around a pencil. We both got pencils in our hands and I think it's, it's a wonderful educational technology. It's hard to improve upon.

Gayathri: [31:55] That's very good. I completely agree with you about writing it longhand. It really does help to anchor some of the points that you want to talk about or discuss or remember later on. Thank you so much, Cliff.

Cliff: [32:07] Alright, thanks, Gayathri. (music)

Derek: [32:12] That was Cliff Anderson, Associate University Librarian for research and digital initiatives here at Vanderbilt, interviewed by Leading Lines producer, Gayathri Narasimham. As computational thinking continues to weave its way into a variety of disciplines, I think it's important to talk about the ways we introduce students to these concepts and prepare them for more substantial work with computing in the future. I really appreciate Cliff's thoughtful and creator approach to this topic. For more on teaching computational thinking, I'll direct listeners to a few episodes in the Leading Lines archives where we've addressed this topic. Back in episode 28, we interviewed Vanderbilt computer scientist, Ákos Lédeczi. He's the lead developer of NetsBlox, the visual coding language that Cliff and Gayathri use in their course. More recently, in episode 68, I talked with Ian Bogost from Georgia Tech about lots of fun things including his approaches to teaching computer science to non-majors. And just a few weeks ago, we shared my conversation with Mark Sample, who teaches digital studies at Davidson College. That was episode 72. Mark, like Cliff and like me, learned how to code back in the eighties, which meant learning to program in Basic. And we talk about that in our interview.

[33:25] Check the show notes for links to all of these past episodes, as well as links to more information about Cliff Anderson and some of the resources he mentioned in his interview.

You'll find show notes for this and every other episode of Leading Lines on our website, leadinglinespod.com. We'd love to hear your thoughts on ways to teach computational thinking across the disciplines. You can reach us via email at leadinglinespod@vanderbilt.edu or on Twitter @leadinglinespod. Leading Lines is produced by the Vanderbilt Center for Teaching and the Jean and Alexander Heard libraries. This episode was edited by Rhett McDaniel. Look for new episodes the first, and third Monday of each month. I'm your host, Derek Bruff. Thanks for listening. (music)