Published by the Center for Engaged Learning at Elon University
www.CenterForEngagedLearning.com/podcasts/limed/

Limed: Teaching with a Twist

Season 3, Episode 8 – AI as Friend or Foe in Computer Science – Part 2

Matt Wittstein (00:11):

You are listening to Limed: Teaching with a Twist, a podcast that plays with pedagogy. Welcome to Limed: Teaching with a Twist. This month's episode is part two of two, featuring Dr. Ryan Mattfeld's curiosity about AI in the computer science classroom and some perspectives from professionals. Last month, we talked with students Arav Patel and Anna David. If you haven't listened to that episode yet, it is not too late.

As a quick refresher, Dr. Mattfeld was looking for advice on ways to implement AI appropriately in his introductory computer science courses, and wondering aloud how allowing and even encouraging AI use could shift the paradigm for teaching computer science. This month, we get the perspectives of Robert Duvall, a computer science educator from Duke University, and Nisha Talagala, CEO and founder of AI Club and AI Club Pro. You will not be disappointed by their ideas. If you care to go back and listen to part one, you can find a link in the show notes or find all of our episodes at www.centerforengagedlearning.org/podcasts/limed. Remember, please rate, review, and share our show. Now, enjoy this episode of Limed: Teaching with a Twist. I'm Matt Wittstein.

(01:49):

Hi Nisha. Hi Robert. I am so excited to have you on the show. We're going to talk a little bit about artificial intelligence in the computer science classroom, and before we get there, I just want you to introduce yourselves to our audience: your name, where you're coming from, what your perspectives might be, and I want to know the answer to this question: Do you think that artificial intelligence is harming student learning and opportunities within the field of computer science?

Nisha Talagala (02:13):

Hi, I'm Nisha Talagala. I'm the CEO of AI Club and AI Club Pro. We are an AI literacy education technology company. We teach AI to a wide range—we call it A to 80. So we have a segment that focuses on K-12 education, and a segment that focuses on college and professional education. So that's where I'm coming from. My background is I got a PhD in computer science a long time ago. I'm a classically trained computer scientist. I've been in the corporate world and the research world for a long time, and I've been in the AI space now for about 10 years. This is my second AI company.

To answer the question of how AI is affecting computer science opportunities, I think AI is changing computer science opportunities in an incredibly fundamental way. Whether that results in good or harm will depend entirely on how we react to it.

A lot of the things I learned as a computer science student, I am incredibly glad I know them—they make it much easier for me to use AI—but I don't necessarily have to use them in my day-to-day. One of the challenges I struggle with is: how do you give today's students enough understanding so they're

empowered and capable of dealing with AI tools, but also have enough understanding to navigate them, and at the same time, fully leverage them? For example, I've written code by hand in assembly language and a series of other languages that these students don't even know existed. I'm happy I did it, but they certainly don't need to do it if they don't want to. The question is, how do you find that balance? The new problems they can solve are beyond anything that was there when I was at their stage of education. If we can channel them into being able to solve those bigger problems with enough understanding, then I think the potential for computer science is very positive.

Robert Duvall (04:09):

Hello, my name is Robert Duvall. I'm a senior lecturer in the computer science department at Duke University, and I've been teaching there for almost 30 years. I have certainly seen things change over those decades as technology has become more and more ubiquitous and easier to use and leverage to do really amazing things. I'm excited to be here talking about this topic, which has definitely been an obsession of mine for the past few years since it's come on the scene.

In terms of answering your question, I don't think that it's harming our students in the sense that their technical abilities will help them navigate this world. Their ability to understand the algorithms behind the scenes or the technical nature of things will be a benefit overall. My concern is more about how they get integrated into the corporate culture now that the role of a junior developer is less clear—because why would you delegate a task to a newly hired engineer when you could delegate it potentially to AI?

Those opportunities for learning in the business setting become less frequent and less useful. So how do they get up to the senior level? How do they master those skills? Does that become something that we teach in the university now, or is that something that becomes a special program tailored to specific businesses? So I guess that's more where I would use the word "harm" in terms of the changes. But otherwise, I agree completely with Nisha that the challenges and the changes are very exciting and interesting going forward, and I'm happy to be part of figuring out how that happens.

Matt Wittstein (06:32):

Well, I am already excited for this conversation as we're bringing some career readiness and some learning principles into it. I want to just briefly share with you the conversation that I've had with Ryan Mattfeld, who's a computer science faculty member here at Elon University. We had a conversation—he was among some colleagues at a coffee shop/bar and they were venting a little bit about AI use in some of their computer science courses. I think they had just come out of an exam. It didn't seem like their students were grasping the material as well as some of their other assignments had shown. That conversation led to more in-depth thinking about what are the opportunities to really enhance student learning with artificial intelligence in that Computer Science 101 course, without abandoning some of the traditional foundational learning outcomes of computer science—really understanding those basics so you can leverage the technology in more meaningful ways down the road.

As that conversation continued, we also started to talk about what would happen if you had a course where there are absolutely no restrictions on AI use. What would that look like for a computer science course to ask students to do a certain amount of programming and develop a project? We explored that a little bit. We took that conversation and shared it with a couple of students, Anna David from Carnegie

Mellon University and Arav Patel from the University of Maryland. They shared that they think most of their peers want to use artificial intelligence ethically. There's probably some confusion over what some of the requirements are in different classes. They also pointed out that when the assignments are relevant and aligned with their values, they're really likely to do the work themselves, but if it doesn't seem relevant, they're more likely to take a shortcut to just get the assignment done, which I thought was a really important point.

Matt Wittstein (08:42):

As we explored that topic of the unrestricted AI course, they had a couple of good points. Anna suggested that we really try to focus on making sure we understand the capabilities of programmers and programming. So perhaps a manager in the future is better able to communicate what they want from their junior programmers or their more senior programmers. Even Arav and Anna both agreed that the possibilities by allowing AI really expand greatly—that you can actually do some really cool big projects in a relatively short period of time, coming with the caveat that it might be really hard to do things like debug if you don't really understand the foundations, though. So there were some really good insights there.

Where I would love to start our conversation today is really just getting a sense of, in that Computer Science 101 classroom, do you believe that the access and the abilities of AI—AI is pretty good at coding, it's pretty good at programming, and it's also really accessible right now, even the free versions are very good at coding skills—how do you think are some of the best ways to leverage that in that Computer Science 101 classroom to actually make our computer science majors, our future program leads in the corporate world, really successful and knowledgeable without over-relying on AI to do the work for them?

Nisha Talagala (10:15):

I think there are a couple of different—and there are so many different elements, and I'm really interested to hear what Robert has to say about it. One thing that I have, and I share—you mentioned a comment about debugging, and it was something I was actually going to bring up in this conversation, which is I have found that for many students, particularly ones who are in the early stages of learning computer science, at least in my experience, debugging is—they consider it to be about, "I'm writing code, and if the code doesn't work, I don't know what to do with that" kind of thing. Whereas in the corporate world, for every line of code you write, you read many, many, many lines of code because you're contributing code to a world that already exists. You don't just get to sit in your own computer and write your own program from beginning to end.

You're contributing a very small part to a very big program. So I do wonder if perhaps the skill of reading code, which is always essential, is now even more important if AI is the one writing it. So you have to become extremely good at reading code. And the question is, how do you become good at reading code? I know how to read code because I was forced to write code. That's actually how I learned to read code. So now the question is, are we asking students to read code when we're not asking them to write it? It's kind of a little weird. And so that's a little bit of a dilemma.

And then, I'm personally a fan of letting them use the AI because I think Robert mentioned this a little earlier—look, they're going to use it, it's there, and you might as well show them how to use it responsibly. One thing I've noticed that I wonder if might be the beginnings of a solution, along with whether they care about the problem, is that there's a level of personal engagement when they feel it's theirs and it's about to be evaluated by someone whose opinion they respect. So one exercise I've taken is I ask students to build an AI and then I ask them to share it with their parents, their little brother, this and that, and the little brother can be guaranteed to find everything possible wrong with it. And the lengths that student goes to make sure it works is unbelievable. And so the key might not be, it doesn't so much matter who wrote it. What matters is are you accountable for the fact that it works and have you done what needs to be done to make sure it does in fact work? And that might include reading it, testing it, thinking about it, whatever it is. So maybe kind of inserting that cycle of personal accountability into something that's a little more creative and something they care about might encourage the right kind of engagement where their thinking skills are still fully engaged, but they can use the tools as best as they can with whatever the tool is capable of doing.

Robert Duvall (12:55):

I agree completely with everything that Nisha just said. And I'll add that one of the things that we're wrestling with right now is how much experience do they need with a blank page, in essence, an empty function. So when they're trying to understand how a loop works in order to, say, debug it or figure out what it can do and what it can't do, how many of those do they have to write in a current computer science curriculum? I'd estimate that students probably write a thousand loops over their four-year career. That's clearly too many, but is it a hundred? Is it fifty? Is it ten? Is it one? We don't know what they gain from the experience of wrestling with how do I make the code do what I want? It's one thing to read the code. Most students can read the code and go, "Yeah, that makes sense. I get that. I can see why that's there." But it's a qualitatively different experience to create that from scratch and to make those instructions. And so we're trying to kind of figure out how many of those experiences do they need? Because I do believe that we can use AI by alternating between writing from scratch or debugging from scratch or that kind of thing, versus having AI do it. I think we can actually accelerate learning for students. I think that it has a lot of advantages. One is we show students that we understand that AI is out there and we're using it within the classroom. We're getting them to respond to it, and we're also able to maybe better target the nuances that we want them to gain from each learning experience. If you think about a current problem that they might solve, we're kind of hoping that they solve it maybe in a particular way or they stumble upon some particular issue that we want them to do, and some may do it in the order and the expectations that we have and some may not.

But if you give them specific examples or specific prompts to generate things, then we can actually control that a little bit better and focus the learning more clearly. And so we get kind of a double benefit there. And so the question is really just how much needs to be from the beginning, from scratch, versus the reading code. And the other thing that I'll add to echo what Nisha said was one incredibly positive thing that I think has come out of all of this is five years ago, I would argue that 98% of our assessment was "write code from scratch." Like, here's a problem, write the code to solve it. And since AI has come on the scene, the community has probably come up with a hundred different ways to assess your understanding of code or your ability to analyze code or to think critically about code or all of these

different aspects of it instead of just "write the code and that's all we're going to grade you on." And I think that that really has opened up the creativity in our field and is better overall for everyone.

Matt Wittstein (16:34):

One of the things our students brought up is not really knowing what the appropriate foundational skills they should have before they should do AI and incorporate AI in their own practice as they're doing stuff in class, but they're also doing stuff extracurricularly for fun or for clubs. And so what do you two view as, before you get to AI, these are some of the knowledge, skills, ideas, habits of mind that we would want you to have before you really just go full-fledged AI?

Nisha Talagala (17:09):

I think there are a couple of different things. Until very recently and until the arrival of tools like ChatGPT, access to AI wasn't really that mainstream. I mean, you were using it if you did a Google search, but it wasn't so much that you could direct how the Google search responded to you. So since the arrival of tools like ChatGPT—and I don't mean that specifically, but I think the use of AI for day-to-day has become so much more broad-based. For example, I might use it to find a recipe the same way that I might have used Google in the past to find a recipe. So that doesn't necessarily mean that I'm a computer scientist, and frankly, I probably don't need to know anything about AI to use ChatGPT to come up with a recipe for a chocolate cake.

So I think if you are using AI to solve problems that are sort of core to what the technology is—so the technology is fundamentally a pattern matcher. It scans data for patterns, it learns patterns, it knows how to represent them, and it knows how to build upon them to use it that way. It is helpful to, first of all, know that it does that, and so secondly, have some understanding of how it does that simply because it helps you understand what it's good at and what it's not good at. There was a time when using AI required you to know how to code. That hasn't been true for a very long time. Industries, for example, in corporates, AutoML is very common. It has been common for at least five years, which is completely no code. You drop your dataset into a tool, it'll come up with the 200 top AIs ranked on a leaderboard and it will tell you, "This is the top one, it costs this much, this is the second one, it costs that much," and then you go pick your poison, it'll generate code to deploy it and you're off and running.

Does that mean that's a good thing? Not necessarily. If you don't know what it just did, you could set yourself up to a great deal of trouble by putting that in front of your customer. So some of it isn't about programming it, but it is about understanding its reliance on data. I call that concept learning. If you appreciate the core concepts of how it does what it does and your responsibility as a human in wielding that tool, then you are in a place to use it very effectively. Now, if you decide to become an AI researcher or someone who creates new types of AI, then certainly a solid mathematical foundation, a solid computer science foundation is very important. But if you're a practitioner, then who is using it for, say, business problems, not necessarily. If you're a practitioner using it to generate new drugs, for example, in the bioinformatics field, a deeper knowledge than the first group, maybe not all the math, right? But a decent understanding of the computer science behind it is usually very important. So it depends on what you're trying to do.

Matt Wittstein (19:48):

So Robert, I want to throw that to you. Thinking about your own teaching computer science introductory courses, at what stage would you unleash the hounds and say, "Hey, I want you to use AI with this"? I'm assuming it's not day one, but maybe it is.

Robert Duvall (20:04):

I'll just respond first by saying that the entire community is wrestling with this question of when do we let students, bless students using it, incorporate it? What are the skills that they're going to learn? We do absolutely agree that the skills that they might come out of a CS 101 course will be different than it was five years ago. That critically analyzing code, that thinking about code, justifying why it's written the way it is, those are all things that may become foundational skills that weren't even considered five, ten years ago.

I will also say that there is a fantastic book by Leo Porter and Daniel Zingaro from UCSD that actually does make an effort to see what a day one curriculum would look like. So they've taught the course, I think for two semesters now using this book. And it's an interesting approach because they don't even attempt to explain the typical programming concepts of loops and if statements and that kind of thing until the fourth chapter of the book, after they've already done several problems, had those solutions generated by AI, asked the students to evaluate those solutions. All the while they're talking about the problem solving that goes into it, recognizing that the students may not understand the syntax completely, but that they should be able to test it, evaluate it. They even go into some of the ethical considerations of using AI both for their own work in the broader context. So I'm glad that they did it and I think it's going to be a good touchstone. It's certainly not going to be the final word on where we go, but it's a good start.

I also want to emphasize that students are going to need to be able to problem solve. That's still going to be a major component of what they need to be able to do. And so that's something that we may not have been emphasizing as much lately, but has certainly become more and more important as people realize what is a tangible skill that they get from a computer science education and breaking problems down, generalizing them, relating them to different contexts. All of those kinds of things are going to be very important and transferable skills no matter what technology they end up using.

Nisha Talagala (22:47):

Yeah, no, I completely agree with everything Robert said. Actually, two things sort of came to mind. So one of them is that I've been noticing that the involvement of AI as a coding contributor, because that is what it is—in fact, it's now generating 25 to 30% of all the corporate code in many large companies—it actually changes the way software is built. And so students who want to be in the software field in the future will need to understand how the software development life cycle is changing because of AI. And one of the points is there's this concept of "N of 1" programming, for example, where the idea is when I had to write all the code myself, I was extremely motivated to find code that was already written that I could leverage. Now I have no such motivation because I can ask it to write it from scratch regardless of how many copies of it exist already, which basically means soon we are going to be flooded with code,

most of it doing the same thing because I asked, you asked, somebody else asked—right now, suddenly N copies of code, which is why "N of 1"—write my code for me.

So that changes fundamentally the way the code is indexed, the way it is stored in repositories. Do I even reuse it? Do I even try? Stuff like that. And then the second thing about being problem solving with the students I teach, I'm a huge proponent of "find a hard problem and solve it." So we run a research institute where at this last one we have over 30 professional publications by high school students solving every serious kind of problem from genetic biomarkers to satellite imagery for fire safety detection. And the key premise is we teach you what you need to know, but we ask you to solve a hard problem and we teach you the skills it takes to leverage whatever's available and then also communicate because communication is so important. So the reason we prioritize research publication is because if you don't communicate what you've done and you don't put in the effort to show another human what you've done, then the only person who's going to know what you've done is an AI and that's not going to help you. So I think we're seeing a transformation fundamentally and definitely problem solving is at the core of it.

Matt Wittstein (24:56):

So I want to transition us to sort of the second part of Ryan's question: what's possible if we remove all of the restraints on AI use and we encourage adamantly that our students engage with artificial intelligence to enhance their code, enhance their projects, et cetera, but specifically within the context of a computer science class? And for this sort of exercise, I want to be mindful that we're going to think about this in the context of computer science for non-computer science majors because I don't think we want to quite yet ruffle the feathers for the computer science majors, but think about the problem a little bit more broadly. So I want you to think imaginatively—what's possible?

Robert Duvall (25:45):

I think it's possible for students from day one in that CompSci 101 or non-majors with no experience course to, by the end of that course, feel like they are empowered to solve a problem that is meaningful to them, that requires some kind of computation. That's kind of been our goal for our current CS 101 course, but at least at Duke, I don't think that we've really succeeded because the students see it as a skill, but they don't necessarily translate it. I don't feel like they leave our course feeling like they could download a dataset, apply it to their research in their area, and come out with some kind of meaningful result, or that they would want to do some kind of visualization or something like that based on the skills that they've learned in that course. But I think that if we ask the students to bring the problems and we teach them the problem-solving skills, we teach them the prompt engineering...

Most of my students right now, prompting seems to be limited to the problem statement that we gave them just copied and pasted in, or effectively a Google search for the answer. They don't seem to really understand any of the conversational nuances that are really required for good answers. And they also don't understand that getting really good answers requires domain knowledge, that it requires a good understanding to explain a little bit about what it is that you want. So I think that there's a lot of really interesting learning to be done in prompt engineering, especially for prompting for different stages of the project, because the types of prompts and answers that you want in the brainstorming phase versus

the forming stage or the value stage versus the implementation stage versus the result stage—I think that all of those outcomes that you want there are different.

And I think that that's valuable for students to see that. And so I really do think that there is a significant academic curriculum around solving problems with essentially an AI companion and actually multiple potential AI companions, because you could create diverse personas to help you engage with the project that you're working on, to give you feedback along the way. And I'd love to see something where a class is really teamwork-oriented and collaborative, where the students are presenting their prompts, presenting their outcomes for critique, getting feedback from everybody—artificial and human—to try to build the kind of thing that they want. And then at the end of the day, yes, they do have hopefully an actual implementation and they are really empowered to do that. And they learn at the end. They learn a very transferable skill: I did it for this project, I can do it for future projects. So that's at least an initial grand vision.

Nisha Talagala (29:17):

I have a bit of an issue with the word "prompt engineering" because sometimes I feel that it is used in terms of how do you construct a prompt. And the thing is, everything you said about domain is so critical, right? It's really about if you don't have the domain knowledge, it's like AI is good, but it's not psychic—it cannot answer the question you don't know how to ask. And sometimes that's not an engineering exercise, that's actually about knowing. If I don't know the depths of cell biology, no amount of prompt construction is going to help me ask the right question. So I really hope that we really, really formulate a language that says we are about teaching these students the domain expertise to leverage the companion in the best possible way. So that's a little just a, if I have the word "prompt engineering," but I completely agree with what you mean there.

Robert Duvall (30:02):

I just want to "yes, and" that by saying that, and that provides the motivation for them to actually learn the stuff. If they can better explain it to the AI, then they're going to get better results. And then when they see those better results, then they'll want to learn more. And so it becomes a motivational learning cycle.

Nisha Talagala (30:21):

Absolutely. Absolutely. I just encountered a student at some event, a college student at a SU environmental science, and she told me something that was really interesting. She said, my professor has decided that we are going to use AI for our term research paper. And the basic thing is, I'm going to judge you on the quality of your paper, how innovatively you search for and use them. So if you want to get a good grade, you'll generate a good paper using the AI in the best possible way. And I thought that was amazing, rather than necessarily penalizing. What I'm saying is, be creative in your use. Did you find the best way to use the tool? Did you find the best tool? If you found an amazing tool, an amazing way to use it, and you got an amazing result, that's great. I thought that was amazing.

So I just feel like maybe that's the kind of orientation that we need to have. And speaking of non-computer science majors, one of my favorite experiences was a few years ago I taught an art student.

She was an artist and she was in my AI class and I let them pick the project of their choice. She wanted to pick a project where she would build an AI tutor that would tell you whether your drawings were good. She generated drawings with bad lighting, bad symmetry, and she built this AI where you can draw something and you can say, and it'll look at the picture and say, your symmetry is off by this much, that much. And what I loved about it is she got what she wanted, she was happy, and then she came and asked me, what other computer science courses can I take? And the thing is that there are many—I'm a CS geek. For me, computer science is simply interesting without purpose even. I like the topic, but a lot of people solve problems. So if you can show them, and AI is really wonderful at showing how computer science is useful to solve problems. It can bridge that gap between problems and computer science really, really effectively. And so I think it actually has a potential to make more students appreciate the value of computer science.

Matt Wittstein (32:23):

Nisha, I want to come back to your story about the student that had a professor that let them use AI. I think it was an environmental studies student, and they were assessed on the quality of the paper and the quality of their use of artificial intelligence. And I just point out the tension there between process and product. And I know in computer science, if you're really grading on the final product, does the code work? It's really hard to give partial credit. It's hard to really conceive how did they get through to those steps. So one thing I appreciate that, I assume that they had some guidelines on how to demonstrate the quality of their AI work, but I'm trying to think in the context of this "all rules are off, use AI as you want," how do we avoid that trap of only caring about "does your thing work" at the end of the semester on maybe a big project as opposed to the process that got you there?

Nisha Talagala (33:26):

In the early days of ChatGPT, its code quality was not very good, and I asked it to write a piece of code at one point. Did it work in the sense that would you give me the right result every time? Yes. But logically it was the equivalent of going from San Francisco to Los Angeles through New York. Yes, I can technically go from San Francisco to Los Angeles by going through New York. That doesn't mean it's a good idea. So even in code, there is a difference between code that works—that could be a really, really bad idea, it still works, right?—and code that's good. And so I think students have to be able to tell the difference. And so maybe the difference, maybe we don't care so much about whether it gives you the right answer. You should think about the inevitable trade-offs that exist, and if you can justify the decisions you made.

For example, if you argue to me that, okay, fine, you know what? Maybe going from San Francisco to Los Angeles through New York is a bad idea, but guess what? There's only one flight every three weeks direct and there's 18 flights a day through New York. Maybe it turns out to be a good idea. The point is, if you can explain to me what logic led you to believe that this is the right way to solve the problem, then I appreciate that. I mean, it's much easier to test code for whether it works. It's much, much harder to look into code and say, what were you thinking when you decided that this was acceptable? Maybe "wrote" is the wrong word—when you decided this was acceptable, what was the logic flow you went through and explained to me why you thought it was a good flow?

Robert Duvall (34:51):

I think process is vitally important, and that is something that when we get into the upper-level courses, we do spend a lot of time actually judging them on the creation of the code beyond just "does it work?" I mean, imagine a computer science curriculum where you never actually looked at the code. You just, like you said, literally checked, does it work or not? I sort of imagine that that would be doing an English course or an art course or something like that where you never actually looked at the work. You just looked at the graded assessment of "was this essay persuasive" as judged by some automated, or as Nisha mentioned earlier, the automated means of figuring out whether or not the art was good. If that became our only oracle and nobody ever actually looked at the art, then I think that's a problem.

So to kind of circle back the conversation, Shannon Duvall at Elon, she actually uses experiential assessments to grade whether or not she thinks her students actually did the work in the 101 course. So beyond just asking them to write a test case or to write some code, she asks them things like, "Well, how did you discover that this was a bug? Or how did you actually go about debugging this and figuring out what the problem was? How did you deal with this situation when it came up or solve this problem when you got stuck?" And so because she's able to ask the students those questions, and given all of her experience, she's able to judge whether or not it's BS or whether or not they actually did it, then it gives her a much fuller picture of how much the students actually got out of the process of creating the program rather than just the end results.

Matt Wittstein (36:55):

So I want to ask, with that idea of product versus process, the idea of a no-holds-barred, you can do what you want with AI, we could do these really big projects. Ryan specifically has the question of how does he set expectations on a project or a learning outcome when he himself doesn't really know what is or is not possible for those students that are maybe trying this course for the very first time?

Robert Duvall (37:23):

I'm not sure I would put limits on the first time I offered the course. I would want to see what the students could do, and I am usually surprised by their creativity and their passion and what they're actually able to accomplish. One of my favorite inspirations along this exact line was Ethan Mollick, who is at the Wharton Business School and is well known for his interest and knowledge about the current AI landscape at that business school. He would always ask his students to add one impossible thing. So their project, if they didn't know how to code, then ask it to create an algorithm or a piece of code for them. If they didn't know how to create a website, ask it to generate the website for them, things like that. All of his assignments now have this "add one thing that you think is impossible."

And there are also a number of—there's a CS50 course at Harvard where students go from nothing to building, again, a project of their own design at the end of the semester, and they have a lot of support, but you're often surprised by what they're able to create. So I don't think that I would initially place limits. I would kind of trust your, or I would ask Ryan to trust his intuition about what's possible so that the students don't go down a complete blind alley. But beyond that, I would kind of let them experiment and see what they can do. And the one thing I wanted to add to my vision that fits in here is I would also have to ask the students to keep a journal throughout the semester where they practice metacognition

practices so that they are actually trying to think about what they're learning about the whole process, about what the creation is. And I think that will give insight into what is possible by following their thinking and their chat logs over the course of the semester that will help inform future versions of the course.

Nisha Talagala (39:47):

I mean, I think it's a very interesting question. So I'm generally also in favor of what Robert said about not having limits. I was thinking about something that maybe is not necessarily limits, but also comes back to a comment that Robert made a little earlier, which is the concern about what AI will do for entry-level engineers. There was a time when entry-level software engineers could be given small tasks. Those tasks can now be done by AI. And so one of the reasons why they're given the small tasks is because they're not familiar with the processes in the place where they've started their job. I wonder if it's possible to, along with having nearly no limits on what they are doing, I wonder if it's possible to insert some process elements—not necessarily limit them, but just get them used to it. For a simple example like, okay, you had the AI generate the code, totally fine, submit it for code review.

I'm also okay if the other person uses AI to review the code. This is more about what is a code review? Why am I doing it? What is the process? Did I check it into a repository? Am I willing to present it to someone? Did someone who isn't me write the test case for it? Did it pass that test case? Did I explain what I'm doing well enough that that other person could write the test case? This is real world stuff. And if the students come in just with this way of thinking baked in, it's going to become so much easier for companies to put them to work right away. They will have less of a struggle to try to figure out, what am I going to get this person to do that's productive while they're learning? So maybe the guardrails you put are not about the topic.

Maybe it's about that process and showing that process early. And now that they're creating these big amorphous things, show them how to do it the way that they would be expected to do it in the world. So that could be something that you could look into and any tool can be used. There's so many open source tools out there that do that. Another article I wrote a while back was about what English teachers can—I think I wrote it on Forbes—what English teachers can show us about how to teach computer science. And it comes back to what you were saying, Robert, about we don't grade an essay automatically and say, is it persuasive? We teach people to read essays, discuss them, write essays, right? There's so much more in the nuance and in real code. There is nuance.

Robert Duvall (42:05):

I'm totally excited about that. And I think that there's a real opportunity here, given the fact that we may not have to teach as much technical stuff. And so we now have room to teach process, to teach social responsibility, to teach teamwork, to teach all kinds of skills that maybe we've left on the side because our courses are crammed full of all the technical details that students may not really need to learn at the beginning anymore. I think there's a lot of possibilities for creating a much more human-centered course by bringing AI.

Nisha Talagala (42:49):

And don't forget debugging. One of my standard interview questions for an engineer of any entry level or otherwise is, "Tell me your worst debugging story. Tell me the most horrible debugging story you've ever had." And if you have ever worked in the depths of an operating system, it should be a really nasty story—as in machine crashing, catching fires kind of story—because the mess of that story tells me exactly how much real world experience you have. So I think students need to have that. They need to know that real things are complicated and they break, and sometimes they break in places that you didn't have anything to do with, but it's still your problem. So I think if we can help them appreciate that, help them appreciate that investigative process, and the first time it breaks, you can't throw up your hands and run to the teacher and say, "Hey, it didn't work. Can I write another one?" Right. So I think if you can show them that, then they will come out being able to—AI will be a benefit to them. They will be valuable humans in the workforce who will leverage AI, which is kind of what we want.

Robert Duvall (43:39):

Absolutely.

Matt Wittstein (43:41):

Nisha, Robert, this has been a fantastic conversation. I am always surprised, but at the same time, never surprised that most of our goals in teaching and learning revolve around just creating humans that can exist in positive ways in our world. And so I love this note to end on. Thank you all again, and I look forward to sharing this with Dr. Mattfeld.

Nisha Talagala (44:03):

Yes, thank you for having us.

Robert Duvall (44:05):

Yes, thank you very much for having us.

Matt Wittstein (44:21):

Hey, Ryan, it's great to have you back. I can't wait to share what our professional panelists talked about.

Ryan Mattfeld (44:26):

Yeah, I'm glad to be here. Happy to hear what they said.

Matt Wittstein (44:30):

So AI in the classroom is always a really fascinating topic, and this conversation was really no exception. Our professional perspectives came from Nisha Talagala, founder of AI Club and AI Club Pro, and Robert Duvall, a senior lecturer at Duke University with 30 years of experience in computer science education.

The discussion covered how AI is changing coding education, what students actually need to learn, and how instructors can adapt their teaching strategies to support learning in this new landscape.

One major takeaway: the shift in how students engage with code. Nisha emphasized that while writing code is important, reading and understanding existing or other coders' code is just as critical, especially in an industry where developers spend much of their time debugging and working with others' code. Robert agreed, pointing out that students still need to write code, but assessments can now balance writing, debugging, and AI-assisted coding.

Another key point was accountability. Nisha noted that students engage more deeply when they feel their work will be evaluated by someone they respect. This might be a way to tap into our students' ideas of making the assignment relevant by maybe having a parent or a close friend or a professor that they really admire actually review and give them direct feedback on their assignments. Robert suggested that instructors should alternate between having students write from scratch and use AI. This will ensure that they develop both problem-solving skills and the ability to critically assess AI-generated code.

When thinking about employable skills, with the advances in AI, the problem-solving actually becomes a lot more essential to the workforce. Prompt engineering, although maybe a misnomer, was also discussed. Simply asking AI for help isn't enough. Students need domain knowledge to craft effective prompts. Robert framed this as a shift from searching for answers to shaping better questions.

Nisha added that AI isn't psychic. It only works with the information it's given. So students really have to learn to articulate their needs clearly and really understand the foundations of the problems they're trying to solve and the tools they're trying to use. This point's really interesting because it kind of says you can't abandon the foundations altogether just because AI can do some of those tasks really well and very quickly.

The conversation also explored evolving expectations for introductory courses. Robert suggested AI could help educators shift focus from technical minutiae to teamwork, social responsibility, and problem-solving. He referenced Harvard's CS50 course and Ethan Mollick's work on AI education, proposing open-ended challenges, perhaps asking students to add an impossible feature to their projects. This might help them push their thinking.

Nisha raised concerns about AI's impact on entry-level engineers. If AI assists with traditional programming tasks, how will new graduates fit into that workforce? Her solution is to focus on process-oriented tasks. Instead of just writing code, students should learn to submit it for review, write test cases, and understand the full software development cycle.

The conversation closed on a note of opportunity. I asked Nisha and Robert about how to set expectations, and they both agreed that actually not setting them—at least the first time around—might help you learn a lot more about what is and isn't possible in that "use AI as much as you want" course. Robert expressed some concern that setting clear expectations vis-a-vis a rubric might actually limit students in some ways. AI isn't about taking skills away from students. It's about expanding their possibilities: more ways to think, more ways to engage, and more opportunities to apply their learning.

So Ryan, I know you wanted some hints on setting expectations, and the response may not be satisfying, but what do you think about having students both identify an appropriate problem and then try to solve it without any expectations or limits?

Ryan Mattfeld (48:26):

So certainly having some sort of definition for expectations appeals to me. Being a computer scientist, I tend to more objective learning and grading styles, but it makes sense, I think, for this course not to set limits. However, the fewer limits you create, the more general the course gets. I think the question now shifts to how do I make sure students are motivated enough to put significant effort in so that we can actually see what they're capable of, which is a question we're having to answer more and more nowadays anyway. So I think Robert's point about letting them choose the problem is a key piece to that, right? So it's making sure that they set their own definition, set their own limits, set their own expectations, and yeah, that could absolutely lead to better outcomes. I think it's just always the challenge of students are often motivated by grade, and so as we've talked about many times in our department, we really need to shift and make this more of an intrinsic motivation, wanting to do this because they want to do this. And then, yeah, I think that would make sense. So yeah, I would like clear definitions, but this idea is also good. It just is now making me reframe what problems I need to solve in creating the course.

Matt Wittstein (49:51):

As I've done some of my preparation to think about storytelling and podcasting, I came across the idea of an XY structure of a story, which basically says, "I'm writing a story about X, and it's interesting because Y." And so I'm trying to think how to apply this to your class. I feel like your XY structure of the class you're designing has shifted from, "I'm designing a course about computer science, and it's interesting because I'll allow students to use AI as much as they want," to, "You actually don't really know what the course is about anymore." So I want to go backwards a little bit and say, what would you envision in this sort of computer science introductory course for non-CS majors? We've also talked about it being sort of a capstone course offline a little bit. What would you really envision as what is the course about?

Ryan Mattfeld (50:48):

From the discussions we've had, clearly with this structure, of course the emphasis is not as much on the technical pieces of computer science, what you'd normally consider a computer science course. However, it pretty clearly is still about problem solving. It is still about, and it still is finding a technical solution in a way, but it's with a lot of the technical heavy lifting—or at least chunks of the technical heavy lifting—being done by AI. So maybe it becomes more of a certainly heavy focus on problem solving, but also a focus on being able to identify and maybe build confidence that you can find something that is interesting and important to you, but that requires a technical skill with knowledge beyond what you're capable of, and leveraging these AI tools to help you do it even without as much background or foundational knowledge. And I guess the question becomes, kind of going back to what Nisha was talking about, can you ask the right questions and what level of foundational knowledge do you need for somebody to be able to pick this up and solve a problem that they're interested in, do something that they may not have been able to do before—solve an impossible problem? What does it take to be able to do that?

Matt Wittstein (52:12):

Robert and Nisha also expressed some concern about how this is shifting the employment landscape for computer scientists. And I'm curious, with recent graduates, if you're already seeing that in your role at Elon and how you might adapt your initial course to accommodate some of those recommendations.

Ryan Mattfeld (52:33):

Yeah, I mean, that's a good question and I don't think anybody has a perfect crystal ball about what will happen. Certainly there is concern, and we've known there would be some concern about the junior developer role in businesses, but I think our question has always been, wouldn't it be very shortsighted for businesses to cut the junior developer roles that don't seem necessary anymore? Because if they do cut the roles that AI can help with, the thing is that you still need senior developers. AI can't do the job that senior developers do. So if you don't have a population of junior developers who are learning and gaining experience in their job, how can you ever develop senior developers? And I guess the answer to that question is, can the really good students with AI jump straight to the senior developer level?

Personally, I don't think that you can. I think you have to have that experience to be able to do it and we'll see. I mean, obviously right now it does seem as though the market is much more difficult for new grads, but we've had fluctuations in CS markets for decades where we'll have periods where it's hard and then four years later it's much better. The question is, will we rebound again? This time, history has shown we will, but this is a new problem. I still think we will, and I think it's largely because of the problem that I discussed. I don't think that we're going to have enough senior developers if people cut the junior developer jobs and the companies are going to realize, oh, actually we still need to hire these people to train them up.

Matt Wittstein (54:11):

And that goes back to kind of a previous question that you were asking: with AI in the classroom, what are the foundational skills? It sort of shifts that a little bit to say, what are the skills you need to develop to have students fresh out of university ready to jump right into that senior development skill? You believe they still need that experience and that time to mature and practice a little bit, but I do wonder if it shifts learning outcomes of your programs to have them more prepared to oversee the AI junior developer.

Ryan Mattfeld (54:49):

I think that we are all wondering the same thing, that we're all questioning exactly what is necessary, how much foundation do we need? Obviously, we still believe you need some foundation to be able to ask the right questions, right? As Nisha was talking about, you have to ask the right questions, whether it's called prompt engineering or whether we call it something else. That process of coming up with the right questions to ask these AI tools to get good results, and then also to understand whether the results they give you are good or not, does take practice. I guess the question is, can we identify the right foundational knowledge that is required to be able to do these tasks with the help of AI? How much of the current information can be cut that we teach in our courses and refocus to make people get to that senior developer level more quickly?

Is it possible to cut enough to make it quick enough that they can do it coming right out of college? And that's what time and experience and practice will tell, and that's part of what this course may help identify—is to be able to write that type of program that we'd expect a student to write at the end of

CS2. Can we do that in one course instead of two? And then are those foundations they learn in that one course enough to let them continue down the path to become, move towards more senior developing roles more quickly? I don't know.

Matt Wittstein (56:16):

Well, on that note, I'm going to thank you one more time, Ryan. It was awesome having you on the show for a couple episodes and sharing with our audience. I'm really curious how AI is going to change our educational landscape, and I can't wait to see what you all do in computer science as well.

Ryan Mattfeld (56:32):

Yeah, thanks for having me. It's been great hearing feedback and getting these ideas and certainly will be helpful.

Matt Wittstein (56:47):

Limed: Teaching with a Twist is produced in collaboration with the Center for Engaged Learning at Elon University. For more information, including show notes and additional engaged learning resources, visit www.centerforengagedlearning.org. Limed: Teaching with a Twist is a creation of Matt Wittstein, Associate Professor of Exercise Science at Elon University. Episodes are developed and hosted by Matt Wittstein and Dani Ani, Assistant Director of Learning Design and Support at Elon University. Olivia Taylor, a class of 2026 Music Production and Recording Arts major, is our Summer 2024 intern and serves as a producer and editor for the show. Original music for the show was composed and recorded by Kai Mitchell, an alumnus of Elon University. If you enjoyed our podcast, please take a few moments to subscribe, rate, review, and share our show. We aim to bring insightful and relevant content to educators each month, and we would love to hear from you. If you're interested in being a guest on the show, do not hesitate to reach out. Our most updated contact information can be found on the Center for Engaged Learning website. Thanks for listening and stay zesty.