Podcast episode transcript: Juggy Jagannathan and Prem Devanbu

Juggy Jagannathan: Welcome to the Inside Angle Podcast from 3M Health Information System. This is your host, AI evangelist, Juggy Jagannathan. I'm excited to introduce our guest for today's podcast, my classmate and friend, Prem Devanbu. He's a brilliant and renowned expert in empirical software engineering and AI. He did his BTEC from IIT Madras and a PhD from Rutgers University. After a couple of decades in Bell Labs, he switched to academia and has been with UC Davis ever since. He retired as a distinguished professor emeritus of computer science at UC Davis and has been recalled by the university to continue as research professor. They don't want to let him go. He has won several awards for his contributions to the field, including the ACM SIGSOFT Outstanding Research Award in 2021 and Alexander von Humboldt Research Award in 2022. I'm looking forward to learning about this research in LLM, large language models and insights in software development and software engineering. Welcome, Prem.

Prem Devanbu: Thank you. Glad to be here.

Juggy: I know your background was in electronics when we were both in IIT Madras a very long time ago. When did you switch to software engineering?

Prem: That happened after I did my master's in computer science. I got very interested in programming even during my sophomore year at IIT Madras and that's when I learned programming more or less on my own with a bunch of other people who were also interested. And then when I went for graduate studies, I was definitely going to do computer science, that was very clear. And so my first several jobs were in industrial software engineering. First at a company called PerkinElmer which essentially at the time built many computers, mostly for government contacts.

And then I switched to Bell Labs after two years at PerkinElmer. And I spent about four or five years there doing just software engineering in various projects, mostly quite large projects. So that was what I really learned about how well-run large software projects are organized and what works and what doesn't. And then somewhere around that time I switched to doing AI audience research, this old-style AI that is more on formal methods and in knowledge representation and things like that.

Juggy: Symbolic.

Prem: Symbolic AI, exactly. And after a few years of doing that, I like to joke that the theory is real in formal AI, but the applications involve magical thinking.

Juggy: And we may have come full circle with our large language models.

Prem: It's gone the other way. The applications are real, but the theory is magical thinking. So, I switched to doing applications in software engineering and my thesis was mostly about software tools. And then since then I went into research in Bell Labs and I did research mostly in software engineering applications, sometimes applications of AI, software engineering at the traditional kind of AI. And then I switched to academia in 1997 and did a few different things in software engineering and some in security. And then about 2006, all these open-source projects became accessible for data analysis and

mining. And so I've been doing that since. And around 2012 is when we realized that natural language processing methods could be used for software engineering. So, we did the first language model application to software engineering in 2012.

Juggy: That was quite some time ago.

Prem: Yeah, yeah, yeah. So that was before deep learning.

Juggy: Language models were quite popular actually in the speech rec community. They were using n-gram models for various kinds to help with the prediction, but it wasn't that well-used in natural language understanding that much at the time, I don't think.

Prem: Right. I mean, there was some of it. So, things like there were translation applications that used language models. They used these noisy channel type models for translation.

Juggy: And language models in those days, basically employed n-gram models.

Prem: There were discrete models, that's right. There were discrete models. So some continuous models were used for things like labeling, spam filtering, and so on. They used n-gram counts and then used it in some sort of continuous model to learn to classify text and so on. But yeah, so our work was all at the time was all n-gram models. And about three or four years after that we realized that this is all going in the direction of deep learning models.

Juggy: So, you published some fairly seminal work on, what do you call it, naturalness of software or something like that?

Prem: That's right.

Juggy: What exactly does it mean? And that's the one which explored the large language, language models and then it became large, I guess.

Prem: Right, right. So, what had happened was in 2010, one of my colleagues who's an excellent, wonderful person to work with, he's since left UC Davis and went to TPH in Switzerland. He wrote a paper, he and the students wrote a paper where they said that software is very repetitive. It was a paper called The Uniqueness of Software. Almost no sequence of tokens in software is unique. Everything is repeated. And so this was more a purely curiosity driven study with no immediate applications visible. And so, when I saw this paper, and my colleague and I started talking so that that's in this course of this conversation of trying to figure out how to make this work. At that time I'd been reading textbooks on NLP, you could say I lack focus. I read all sorts of stuff. So, I've been reading Manning's textbook on natural language processing.

Juggy: Manning, yeah, he is the guru of NLP.

Prem: Yeah, it's a wonderful textbook, beautifully written, lots of examples, very well put together. It's also a beautiful book. So, I was reading this book and then it just occurred to me that maybe these NLP based methods could be useful for software. And the first application we came up together was code completion, which is something that was built into Eclipse. And at the time, Eclipse was the most widely

used IDE. So, we thought can we outperform Eclipse's completion engine with NLP based methods? And it turned out you can. So that was quite a revelation. So, this was somewhere around fall of 2011 when we figured this out and it was a really exciting time at Davis. It was like when the results came out, I didn't believe it. So, our postdoc at time was this guy called Abraham Hendel, great fella. He repeated the study a few times so that we were both convinced that it was real. And then we wrote a paper and as it turns out, the paper actually had a hard time getting published. The reviews are quite negative.

Juggy: Now it's probably the most oft quoted paper in this domain I would think. Yeah.

Prem: I don't know about that. But yeah, it is very well cited.

Juggy: So, when did these language models become large? So, you've been at it for more than a decade…

Prem: Right, right. So, one of the problems we noticed right away was that the vocabulary proliferation in software is very different than natural language. So, in natural language you could read two or three chapters of a book or even several books. And most of the vocabulary is roughly the same if you're working in roughly the same domain. In software, the rate at which people invent new vocabulary is incredible. It's nothing like natural language. So that's a problem.

And this is necessary in code, you have to do that. So because of this, the traditional NLP based models were not quite as effective. But as it turned out, the NLP people solved this problem for a different reason, which is many languages and agglutinative languages like German.

Juggy: Right. So yeah, you are looking at maybe camel case variable names or something like that, try to split it up. And that is something similar to this German word, concatenate word, or something like that.

Prem: Exactly. So agglutinative languages and morphologically rich languages where there's, you know in Tamil you might say “Vandutuirikirane,” which means identifies gender or person, all the stuff, “Vandutuirikiraan.” So those sorts of languages have to be split appropriately. And so the NLP people devised a purely statistical knowledge language neutral way of doing this. And that turned out to work really well for software, that solved the vocabulary problem. So, we had come up with another way of solving the vocabulary problem, using discrete methods, which worked, which was beating deep learning methods for a while, but that didn't last once the statistical splitting methods came up or beat everything else.

And then of course after that, the scaling of NLP is really, really because of transformers. Before transformers, they use these recursive methods, we still had propagation through time. And so because of the propagation through time, the training speed was inherently limited. You couldn't parallelize it. But when transformers came about, even though they're quadratic in the sense, in one sense, you could still scale them much better because they were inherently parallel. So you could make them bigger. And if you had more through more graphic scores at it, it would just speed up. So after that, it's just a question how much money you had.

Juggy: Yeah. This whole transformer has literally transformed the whole thing. It's before transformers and after transformers. It's just incredible what we see. And I know with ChatGPT and all these incredible literally the revolution going on in this area, everybody is focusing on prompts. In fact, prompt

engineering has become its own discipline like software engineering. What do you see? Where do you see prompts in software? I mean, I've been using Copilot and I find that I had to be very careful in how I phrase what I want to do so that it can actually give me decent code, otherwise it gives me some random stuff and sometimes it even spews out stuff even before I write something, which is irritating. Where do you see prompts going with the code completion and the use of large language models for developing?

Prem: Yeah, it's fascinating. We started looking at this somewhere last end of last winter, like February or March of last year is when we realized this stuff is getting really interesting and we had to start looking at how to do this. And fortunately we got access to GPT-3 around that time. Yeah, so it's fascinating. I mean, it's really interesting how quickly the whole idea of fine-tuning is disappearing and...

Juggy: Yeah, everything is zero or few shots.

Prem: Yeah, yeah. And chain of thought is another thing that is really surprising and interesting. So yeah, I think there are specific ways in which these kinds of techniques apply to software and that's where a lot of people are trying to figure this out. So for example, one difference is that software is highly contextual. Every project is different, every package in a project is different, every file is different, every method is different. And because these language models, essentially they're conditioned generators, they condition on the prompt to give it and then they produce text and the text is produced auto aggressively. Every word that's produced depends on the previous word that was generated and of course the prompt.

And so what happens is that because there are 175 billion parameters or 40 billion parameters depending on the model, the prompt, which is, you can think of it as a prior in a Bayesian sense, the prompt can be represented in this enormously rich space. So think of it as this multi-dimensional forest in which you can position yourself in different places and each place where you position yourself, you have some kind of mental state. I mean, I don't want to use mental too much. It's very dangerous to confuse what these things are doing with mental states because it's an entirely different-

Juggy: But it's a neural network.

Prem: Yeah, you don't want to anthropomorphize it. But it is prior and the prior space is very rich. And so by suitable prompt engineering, you're positioning the prior in different places. The problem is that this is an entirely empirical phenomenon. So some of this work is really interesting and can be built upon where how to arrange the prompt. So, for example, in software, one thing that we and others have done, which I think is a robust finding, which is that conditioning on context improves software performance. So if you want to for example, teach a model to summarize score in a project, which is what something we've done. So you can do a few things. So you give it, "Here's some code, here's some summary, here's some code, here's some summary," those are your priors. Now here's a new example. How would you summarize this? That works. Now if you're trying to summarize code in a particular project, you're better off giving examples from the same project. It signifies significantly performs better. So that's an example of something that seems like a fairly robust finding.

Juggy: So in this case, you are using a prompt, you're designing a prompt so that it can summarize actual software.

Prem: Yeah, summarize usually a method at a time. Another thing that's interesting that people have done that I think is a fairly robust finding is this idea of using some... It's similar but related, is using information retrieval to find relevant prompts from a pool. So let's say if a pool of 5,000 examples that are very good examples, human labeled, human constructed that are good examples to use, and you have a query and you take the query and use information retrieval to find relevant examples and pack them into your prompt. And then you're going to get good answers for the one that you're giving because the questions are related. What I think is not robust is people hand jiggling wording to get exactly what they want for a particular example. So if you have a robust theory of why this works, then fine. So I think what I find interesting in papers is when they have a sound theory on which they're doing their prompt engineering and they show... Again, it's purely empirical, they show through well-constructed experimental studies that this is a robust theory of how to construct prompts.

Juggy: Right now, I mean as far as I can see, whenever they say prompt engineering, it seems to be random things people are trying. So, you're saying people are actually working towards a theory of how to actually construct prompts which are more effective in generating the right output.

Prem: Yeah. I think that's going on and I've seen a few examples in our paper. There's some more general papers as well. They're not just software specific. I don't have a link right now, but I can make it available to you later. So one, we, for example, just recently put a paper on archive and our idea was when programmers try to explain code or try to patch it, they may look at the code and then they mentally analyze the code to see which variables are being used and how is the values being constructed and propagated and what's the control flow like.

So they do this little superficial analysis of the code in their own mind first, and then they can do debugging or summarization or whatever task you give them. So our hypothesis was if we do some sort of basic analysis of the cord and pack that also into the prompt in a few short settings. So code analysis, output, code analysis, output, and then you give the new code you have along with the analysis because analysis is free, it can run an algorithm to do it. It doesn't really require any intelligence and then it'll produce a better output. And that seems to be a fairly robust finding. It seems to hold up.

Juggy: So that's interesting. That is interesting. I've started seeing using the GPT underlying frame of large language model framework as different agents. So you have one which will help you generate some code, another one which go and inspect it or test it or critique it. I've seen that being used more and more in at least my health care application domain. Like, hey, summarize this thing and then you have another agent say, "Critique this thing." Have you seen something like that for maybe software testing?

Prem: Yeah, not yet. I mean, I think that would be really interesting. So what we've done is basically trying to figure out how often do language models that produce code make mistakes? There's been a bunch of work in it and we've done some of that. How often does it make mistakes? How often does it repeat mistakes? But I mean, what you're raising is a really interesting point. And you might have seen this thing called constitutional AI.

Juggy: No, I don't know. What is constitutional AI?

Prem: The idea is basically like what you said, use a language model to act as a critic. So essentially you're positioning the model in some part of the prior space where this part of the prior space model tends to behave like a critic.

Juggy: Yeah, this goes back to the old AI days when we have multi-agent work. You have different agents cooperating, collaborating in some fashion. So it's the same idea but with a different implementation of course.

Prem: Exactly. Exactly. So because of the rich prior space, you could take some text and say... Give it in the prompt and say, "Now think of yourself as a trainer for diversity in your organization. How would you respond to this text?" And then the model completes the text and gives you some sort of feedback. And then you could take that feedback and think of it as positive or negative, and now teach the model that this kind of text would be viewed positively and suitable for corporate correspondence. And this kind of text is not suitable for corporate correspondence.

Juggy: This is how you try to address toxicity and bias and those kinds of things.

Prem: Right. So this is how, for example, ChatGPT was trained but using human feedback. Now what they've discovered is in constitutional AI you can have the model think of different criteria and critique itself and then you can use this text for supervision.

Juggy: So, the feedback is from the model itself as opposed to a human. Yeah.

Prem: Right. So this area it is really, as you said, it's a really cool idea and really exciting and this is something that I think there are people looking into this, including us, like how do you use this to improve software quality?

Juggy: Let me pivot slightly. You are a computer science professor. I used to be one and I'm a gen professor. I'm really genuinely worried about what is going to happen to U.S. education and software education. What do you think next semester, if you're going go and teach... This is something all educators are worried about. I think at some level, at least in other discipline, other than computer science, they're probably worried about people cheating with ChatGPT or submitting assignments. Any thoughts on this? And this is also related to something which is, I think you wrote a opinion piece on AI safety. I mean, they are interrelated but distinct and different. I'm just curious, where is this all going? I mean, it seems to be almost spiraling out of control and there seems to be two streams of thought.

One group is saying the sky is falling, another group is saying super optimistic. Google yesterday had a big IO event where they were promoting all kinds of new applications using large language models. And of course Microsoft is doing the same thing. So, on the one hand you have people with a super optimistic view of what's going to happen and the other side, gloom and doom, we better take safety very seriously. And I don't know what to think, honestly. It seems pretty crazy. So, I know you're in the thick of it, so as I am. So what are your thoughts?

Prem: So, I mean, I guess I'll confess I've been more focused on the research problems and less on... I'm starting to get interested in this issue of policy, I haven't been a policy person. I'm hoping to go to a meeting in DC in June where these DC issues I think will come up. But the meetings aren't finalized yet, but I'm hoping that I'll get a chance to meet people who are thinking about this more from a policy

perspective. So, you asked a lot of questions. I mean, just about education, you and I have probably both taught classes with 400 students teaching them how to code.

That's where the problem is going to be most intense because I think a lot of people may be motivated to use ChatGPT to do homework when they're under time pressure. I don't know what the answer is. The way we used to do it, which is automated testing, it may be problematic. So, we have to probably supplement that with some kind of manual in-person interviews and so on. Now we can't do it with 400 students, so do we do it with TAs? Do we do it with random sampling of students? Would that be fair, unfair? I don't know. It'll be a while before I go back to teaching since I'm now a research professor.

Juggy: Actually, I'm also not teaching that big groups. I'm teaching more smaller groups, advanced courses. But still it is an issue, it's a problem.

Prem: Yeah, yeah. So, I'm an engineered research professor and I'm not paid by the university. So if they want me to teach, they'll have to pay me. So I'm not sure they're excited about it soon, but I might do a seminar for free at some point, but that's not the same thing. It's much smaller and much smaller situations.

Juggy: So they really have to worry about how to construct meaningful experience for students.

Prem: Right. And it probably has to be interviews to check what the student understands. So what's going to happen to the way people actually code in practice and staffing issues is another big question. I mean, I think a lot of people... Things are easy to write, I mean, much easier to write. I was doing something for my wife with the React, which I've never used before and it was actually not that hard.

Juggy: So it is incredibly easy to code. I mean, I've been pleasantly surprised and genuinely impressed how Copilot helped me code. So it is amazing and I haven't tried to do some random coding in different languages, but I'm sure it's just as good. So it brings up not only the CS education, there has been all these efforts to teach young kids how to code. And MIT has this whole program called Scratch and how to teach young children. Will that all go away? I mean, all these kids, the new generation kids will be programming using their basically natural language.

Prem: So I mean, some people argue that this leads to calculators in arithmetic.

Juggy: That is true. I know I've seen that analogy, but I don't know whether I buy that completely. But...

Prem: Well, here's something that's interesting I think, which is that a lot of people say that the instruct GPT, which is basically chain of thought reasoning, which is a new function that these language models have demonstrated they can do. When that paper came out, a lot of people are saying, "Well, the reason why these models can do this is because they were trained on code." So this indirectly tells you that this style of reasoning, computational thinking as Jeanette Wing called it, is a helpful way to think for human beings. So I still think that knowing how to code is a helpful human skill, which doesn't necessarily mean you have to be a programmer. It just helps you think in a certain way.

Juggy: It's the logical reasoning. You're saying you have to do this, you have to do this, you actually have to lay out a plan of action.

Prem: And you have to think about efficiency and all this kind of stuff, divide and conquer, these sorts of algorithmic design things. The useful way to learn to think. There are downsides for sure to this style of thinking, but it's a useful skillset in a larger scope of human endeavors where you also know have some ethical values, some moral values, some sense of humanity and what's important and not important. In that context this is a useful way to think, I think.

Juggy: So that's a good segue to alignment, AI safety and alignment. So I mean, I just read it on your opinion piece about alignment. Can you explain what alignment is so that most people may not be aware of what alignment actually means?

Prem: Right. So the classic example is Nick Bostrom's paperclip making machine. So you get an all powerful AI and you tell it to make paperclips and Nick Bostrom's argument is it'll turn the universe into paperclips. You cannot stop it because it's super intelligent and it knows it wants to make paperclips. It'll take turns, you and me, and melt us down and extract the iron from our blood and make paperclips. So the argument there is that you have to align an AI's goals with human goals. So the AI, whatever it's asked to do, it'll always have primacy given to human goals in relates to Asimov's laws of robotics. So that's the overall idea of alignment. And so people are trying to figure out what that means and how to program AIs to do it.

Relating back to your other question essentially about policy and this wild difference between this enthusiastic presentation that you get from somebody like Yann LeCun about what the future of AI holds and the very negative and pessimistic views that you get from somebody like Timnit Gebru, I don't know who's right, frankly. I mean sometimes I think Yann's right, sometimes I think Timnit is right. I don't really feel like I understand everything as deeply as they do. That said, I really think it's important to have policy from government. I really do. I'm glad the EU is taking a very serious look at it. There's been some white papers coming out of the White House, so I think this is all very good because the way business works, you can't expect them to do anything other than-

Juggy: Make money.

Prem: Yeah, exactly. That's what they're supposed to do. I mean, that's the whole theory of fiduciary responsibility or whatever. So I hope there will be innovations.

Juggy: Yeah, yeah. It's scary at one level and exciting at the other level. I really don't know. Every year I keep thinking you can usually see what's coming down the pike for a little bit, but now it looks like all bets are off. The technology is evolving at a breakneck speed and the applications... I mean, ChatGPT went from nobody has heard it to 100 million users or something like that. I mean, this is even beyond exponential scale.

Prem: Right. And a lot of that is driven by investment. So people are putting enormous amounts of both brain power and money into this. And people have different motivations. For me and my students it's like, "Let's do something impactful that actually other people want to build on." And for companies it's like, "How can we get many users and get them to engage with our products?" So yeah, there's a lot of motivation behind this. And so things are changing very quickly. It's the rate at which amount of reading I have to do is enhance. I've read two, three papers a day to keep up.

Juggy: We need to get ChatGPT to summarize these papers and present it to you.

Prem: There are tools that'll summarize papers for you. There's a semantic reader from open AI.

Juggy: Yeah. I do subscribe to semantic. I need to use it more. Yeah, I need to use it more. It's just impossible to keep up with it.

Prem: It's very difficult. I mean, the best way to do this, get a grant, hire three very smart PhD students and tell them to summarize papers for you.

Juggy: May work for you. I'm a poor researcher in a corporate office. Excellent talking to you, Prem. Any closing thoughts on large language models, software engineering, AI safety, future of coding and-

Prem: It's a peculiar time. On the one hand, there's so much to be explored in the context of these models and what they can do in terms of prompt engineering, in terms of different types of training methods. There's a lot to be done. On the other hand, I'm often frustrated by the lack of theory. There's not really any solid theory. I remember back to the old days of AI when all we had were theories. And it's really a shame that it's very difficult... Maybe some people do, maybe physicists or people like that have a deep understanding of these very large networks. But that's the frustrating part is that it's very hard to have any concrete theory that you can just solve.

Juggy: Yeah, there is growing work right now on looking at the theoretical underpinnings of why these large language models are indeed having these emergent properties and things like that. That has to be critical, I think, when you have this super intelligent system suddenly emerging out of nowhere.

Prem: Yeah. Well we don't understand how the brain works either, so...

Juggy: Right. So we have created another. Yeah, somebody mentioned this is more like an alien brain. We need to figure out how this works, just like we don't know how our brain works. Well, thank you, Prem. This has been wonderful talking to you.

Prem: Nice talking to you too, as always.