The On Cloud Podcast

Mike Kavis, Managing Director, Deloitte Consulting LLP

Title: **Unlocking innovation: Deloitte's Ryan Lockard on the Generative AI revolution in software development**

Description: In this episode, Deloitte's Mike Kavis and Ryan Lockard talk about how Generative AI is revolutionizing the software development lifecycle. It's all about speed, accuracy, and innovation. According to Ryan, Generative AI moves beyond yesterday's code generators to include context, as well as codebase and pattern recognition. In short, Generative AI will help software applications—and the developers who build them—be more productive and innovative, which is a win for everyone.

Duration: **00:23:05**

Mike Kavis:
Hey, everyone, welcome back to the Architecting the Cloud podcast, where we get real about cloud technology. We discuss all the hot topics around cloud computing and AI with the people in the field who do the work. Mike Kavis, your host, chief cloud architect at Deloitte. And today I'm joined by a good friend, Ryan Lockard. Ryan is a Deloitte Engineering lead for banking and capital markets. So, welcome to this show, Ryan, and tell us a little bit about your background before we get started.

Ryan Lockard:
Yeah, thanks, Mike. Thanks for having me. Pretty much what you said, but a little bit more color. I grew up on the enterprise side of the desk and worked in mostly highly regulated companies doing, I like to say, every single job in technology under the sun, from actual coding through project management, technology team management. Did a little bit of testing. Touched pretty much every tech stack you can imagine from Oracle E-Business all the way through object-oriented and scripting and front end and all the fun stuff there, and then found my way into consulting in the second half of my career.

Got to really help grow a boutique startup consulting business that focused on cloud and DevOps for highly regulated enterprises and had a good run with that prior to joining the firm. And since then, I've been doing exactly what you said, my friend. I work in Deloitte Engineering. I get to lead Deloitte Engineering for banking and capital markets, and I get to serve some of the biggest clients in the world with some of the most cutting-edge problems to solve for. So, it's a fantastic space to be, and I'm surrounded by some of the smartest people in the industry to help me solve these problems.

Mike Kavis:
That's cool. And we're going to talk about some of this cool technology you just mentioned about. You were at Google Next. I was watching it streaming, AI, AI, AI, and AI. So, what were some of your takeaways from Google's spin on AI and what are some of the new things they're talking about that you look forward to getting your hands out to you.

Ryan Lockard:
You kind of nailed it in the question itself. It was an AI conference. But as you know, the best conference talks happen outside of the rooms. They happened in the hallway. They happened at the—where the food is being served. They happen at the bar. And just the sentiment from the folks that I was talking to was just that, like Google pushed all the chips in. They're trying to build out enterprise-grade utility tooling for some of the most cutting-edge problems as it relates to AI.

And what's interesting is where they're finding those intersection planes, and this is sort of answering the second half of your question. They came out firing, TK came out talking about Gemini 1.5 Pro, talking about one million tokens being able to pass through, the amount of video it's allowed to—it can now leverage inside the LLM and provide near real-time responses, audio, very complex file formats that it's now able to ingest and act upon in really quick time. And when you think about the possible use cases for that—I'll put my engineering hat back on and I'm trying to think through how do large

enterprises operate in a more nimble way where they got cost pressure, they got regulatory pressure and they also have market demand for new or reimagined features and offerings from their clients and their customers? How can something like Gemini 1.5 Pro solve new problems in new ways?

And you really want to start thinking about the art of the possible, how AI could be absorbed into different use cases across the SDLC, for example. Is there a way for us to better capture business requirements and convert those business requirements from possibly a conversation and start converting those into tasks that engineers can start acting on? Maybe they're going to act on those with AI as well in that type of space. So, when I listened to it, there was some really interesting conversation obviously around Gemini, a lot of conversation around some of the creative agent things that they're rolling out for the Google productivity suite

Mike Kavis:
Yeah, it's pretty cool. And they gave a lot of industry examples. A lot of it was focused on retail, and obviously that didn't apply to everyone. A lot of people think Gen AI is just code generation, but it's more than that. So, putting the engineering hat back on, walk us through the software development life cycle and all the places where this touches.

Ryan Lockard:
Yeah. And you're right. You also saw Goldman Sachs onstage and talking about their AI journey and what they see as the art of the possible. And I'll go through the SDLC in a second. But I think the other thing that's interesting is just how AI is removing some of the routine actions that an analyst would be doing to sense the market and figure out where they want to do additional investments and leveraging AI so that they can focus their creative brain on the harder problems to solve and remove them from the routine. Going through and parsing a bunch of press releases or P&L statements or whatever SEC filings that they need to go through, a lot of that can be managed through AI.

But back to your question on the SDLC, there's a beautiful parlay from what I just described of a financial services analyst and what a software development team can do. So, let's, just define the term, the SDLC. In my simple definition of that, it's every activity that's involved in going from a businessperson—or sometimes a non-businessperson, but we'll start there—having a brilliant idea. And that brilliant idea can be, "We need to improve this service. We need to release this new feature. We want to go to this new, interesting market in a new novel way, so we want to open new channels." Whatever the idea is to either grow top line revenue or reduce bottom line cost. So, from that moment, all the way through, every single step in the process to getting something out to market, to hitting release, to publishing code into a production space so that we can get insights from the actual users. Was our hypothesis from their business person originally proven true or false in the market?

So, when you think about everything that's in between those two bookends, you have product owners, or product managers, or BAs, or whatever that function is that provides voice of customer to the technology team and can help really land what the business problem is and ensure what we're building resonates with that initial business problem. We have architects that really help to take that business problem and both describe it in a way that makes sense to technical folks and assert that it's in line with the corporate governance or the guidance on how we ought to be building software, infrastructure, data flows, services, whatever.

Obviously, the product owner will take that with the scrum master, or whatever methodology our clients may be working under, and take that architecture and that business understanding and start describing it in technical terms, consumable chunks that will take the larger picture and reconstruct it in a way that a developer can pick it up and say, "Right, this makes sense. I know how to write code for this now." That usually is going to be ramped with some acceptance criteria and then we invite our testers into the space. That will give us some form of functional and nonfunctional test steps that sort of make sure that what we build is compliant and doesn't break anything upstream or downstream, or at least that's the theory.

Finally, we do have that developer persona, the person that's responsible for ingesting all of what we just referred to, the business requirements, the architecture, the tests, the acceptance criteria, the tasks and all of that, and convert it to code. Once we ideally have code and that code passes all of our testing, be it automated or manual, we now have something that can be put into a pipeline. Well, it might have been put into the pipeline for the testing phase, but it can go through a build interrogation and ultimately a deployment process where it then goes into production, and we may have some operations folks or some support folks that are keeping an eye out to make sure everything stays healthy. If you're living in an SRE world, maybe you're doing more of a, "you build it, you run it" type space.

What if we can bring AI into each and every one of those personas, not just the developer? You can do a search right now for Gen AI for software development and I'm willing to bet you're going to get hundreds of thousands or millions of results. And in those results nearly all of them are going to be focusing on one persona and one activity for that persona, and that would be when the developer's writing code. Developers do a lot more than just write code, but we really seem to index on that. And there's a lot of really good products in the market to help with AI-driven code generation. But if I'm a tester or if I'm an architect or a product owner, that's really good, but that doesn't help me?

So, what I've been working on inside of Deloitte Engineering is absorbing AI into each and every one of those personas. As an architect, wouldn't it be great if I could take my business requirement and push it through an LLM so that it can come back with a cursory rendering of a sequence diagram or a flow diagram that is both driven by the business requirement itself, but also capturing the context of all of the other architectures that we've drawn? And maybe it should be compliant to our standards? We're doing that today very much the same way we're doing that with the product owner, where we're taking those architectures into business requirements and we're pushing them through an LLM so that our product owners can be given a first set of user stories with acceptance criteria and subtasks that they can then improve.

And by the way, every other person is the same. Software developer. Wouldn't it be great if from my IDE I can directly interface with my system of record, and ingest all of the context from my user stories and/or my other requirement documents so that I can get pseudo code right in the IDE that then converts once I go through review and do that pair programming with the LLM's pseudo code, start writing actual code? Each one of these things, it's not aspirational. We've built these in Deloitte Engineering today and we're going to market and we're seeing amazing return with it. And I think what we're starting to realize is you don't necessarily need a new, shiny tool to bring AI into the enterprise. You can actually piggyback off of the existing tools, and involve the AI there. Therefore, it makes it more contextual for the user. I don't have to learn a new tool and it stays inside the boundary constraints of a lot of these regulated companies.

Mike Kavis:
Yeah, that's pretty powerful. And that's a lot more than the next code generator. So, I'm an old guy and I went through JCL and COBOL generators and they'd spit out code and then I would spend more time fixing that code than I would to write it. Why is this different than all the other generations of code generators? Is it the fact that we're no longer having a one-way conversation with a tool where we just put in a bunch of parameters and it spits out and that's it? Is it because we're now iterating the tools? I don't know. What's your take on why this time code generation, even though it's only part of the equation, why is it going to work this time?

Ryan Lockard:
Yeah, and I'll answer the question around code generation, but the answer is effectively the same for every other persona as we've built it. The one-way conversation is very spot on. This is not a one-way conversation. When you look at those code generators for COBOL and so forth, those are very linear. They're just a series of nested if-then statements where it's expecting an input. If that input exists, it's then going to convert it to this output. And it uses a very fuzzy logic in doing that generation. What we're seeing with Gen AI is the off-the-shelf LLMs have a pretty good understanding of most widely used code bases.

And then there's others. There's some boutique LLMs that are very well-tuned for specific tech stacks. And that allows for better understanding, and it allows for additional context beyond just the prompt. You can pass along additional code so that the LLMs can actually learn the patterns. You can point towards golden copy architectures or patterns so that the LLM now knows what you're looking for from a result set. I love using the example of a logging service. We need to implement logging for this new service that we're going to build. If I send that into a code generator, God only knows what logging service it thinks I'm going to need.

I work with clients that actually have their own logging service that they built in-house. With LLMs we can pass along the additional context. With Gen AI we can pass along the additional context to tell it, to give it some instructions as to what we're expecting as a result so that the code is generated in service to those expectations. And beyond that, half steps or less than full steps each time to assert, "Hey, this is what the AI or the LLM thinks the right next thing to do is. Do you agree or disagree as the human?" If you agree, it'll progress onto the next step. You're not just getting some garbage spaghetti code at the end saying, "Good luck with that, pal." You're actually being involved. You're effectively pairing with the LLM as you go through.
But what we can do is we can use the AI to strip away all of the non-novel pieces of engineering. How many times—do you go and copy a bunch of bootstrap code, paste it into your IDE, and then just fill in the blocks? Or copy code from another piece of your repo and paste it into the project that you're currently working in and then fill in the blocks? What if you didn't have to do that? What if that boot-strapping was done for you and all you had to do was focus on the hard problems to solve? They can't take that away.

Mike Kavis:
So, I'm going to ask you another question. This could be a podcast on its own, so we'll keep it short. And we were talking about this a couple weeks ago, but there's this—trillion-dollar problem of legacy code, old mainframe systems. And when we talk about what Google is talking about expanding how much you can shove into an LLM, you can put—large segments of code or large diagrams or large videos. Now, do you see a future where we can now think twice about not touching those and actually grab code diagrams, whatever exists, throw it into LLM and kind of reverse engineer systems we were afraid to touch without the workforce that is all retiring and going away? Is that realistic?

Ryan Lockard:
So, off the top what I would do is I would bifurcate: Are we psychologically prepared to do it? The psychological safety piece of that is a bit more than 20 minutes of an answer. I think, to your point, there are tools today that go through and figure out what the business rules are that exist inside of these legacy systems. But we have tools today that'll go through and understand what those business rules are, and that's where a lot of the fear comes from. It's not the COBOL code itself. Yeah, sure, there's not as many COBOL engineers and programmers as they once were.

It's just understanding what the upstream/downstream impacts are. It's just understanding what this function actually does. And I think today what we have is the ability to go through and service by service start figuring out, okay, where are the dependencies? Where are the interfaces? And what is the intent of this code? That's a today thing. But what that does is that enables a strangler pattern, which is the ability to take service by service all from the mainframe, almost like you're peeling back the onion is sort of the metaphor that I was trained to use when I was learning about strangler. But at the end of the day you can strangle it down so far, but then you're still stuck with this core that's really, really hard to both understand, synthesize, modernize, and then migrate. So, you're constantly paying for not just the operations of that core but all of the infrastructure around it.

I think to your point, as more of these extremely large and extremely quick processing LLMs come on, I think what that'll do is it'll allow us to get much more instrumentation and observability into how we can reimagine these services that are very much part of the fabric of our economy. How can we take those and reimagine those in a modern language? And the day of this problem with the mainframe apps are that they just keep working under load, so how can we also not just reimagine them, but assert resiliency, assert reliability, and assert the ability to scale with demand the same way that the mainframe is doing today?

I think that is one of the next steps that we're going to see beyond just the ability to trace the business requirements and do some of that COBOL to JBOL migration that we're doing today. That then should influence a little bit more of that psychological safety that goes into pulling the plug on the mainframe one day.

Mike Kavis:
So, last question here before we wrap it up. I get a lot of questions about, "Okay, show me the numbers. This is all great. It's all pretty, it's fun, but show me the numbers." And I know you and I have been doing some writing on this topic and we're seeing something like initial pilot results of 10 percent gains, but we think the future fully baked, fully adopted it much bigger than that. So, tell us about—the typical numbers, if we kind of roll it all up, we see in performance improvements?

Ryan Lockard:
Yeah, I mean, I was having a conversation this morning with a client, and this was one of the areas we spent some time. Just going back to grade school math, I'm just going to start with, "Denominators matter." So when you see some of these very aggressive percentages, like 50 percent faster code writing

or 50 percent time less writing unit test, or something along those lines, you've really got to step back and not just say, "Okay, does that mean I need 50 percent less testers, 50 percent less developers, whatever?"

The denominator matters so much because—as a developer I don't write code eight hours a day. I do a lot of complex problem solving in my head. I used to have—as a developer just detesting the idea of having to go to meetings. I would have to go to meetings. I'd have to pull somebody aside and work through a problem on a whiteboard or on a piece of paper. So, the amount of time that I actually spend typing on a keyboard, If I'm saving 50 percent of that, I don't think it's that honest to say that I'm saving 50 percent of my developers' time. What I'm doing is I'm saving 50 percent of my developers' time on a particular task.

So, when I work with clients what I like to do is talk about the total cost of ownership of productivity across the entire SDLC for productive and nonproductive times. If I walk away from my desk and I take a 15-minute phone call to talk to a family member and come back, it's not like they don't pay me for those 15 minutes. I'm still getting paid for my nonproductive time.

Ryan Lockard:
But the only number that matters is the total cost that went out the door to get this feature developed. And that's not just code writing time. That's not just like meeting time. That's everything that went into that full stack team's ability to deliver the code. So, what I'm seeing with those tools, what I'm seeing is single-digit improvement from a percentage perspective. There's debate on whether it's closer to five or closer to nine, but we're talking single digit. We're not talking 50 percent, 30 percent, 15 percent, any of those. But if you change the denominator and say, "Okay, what percentage improvement am I seeing on code writing?" that's when you start seeing those double-digit numbers creeping up into the 30s, 40s and 50s.

Mike Kavis:
I mean, it should only get better over time, I would think.

Ryan Lockard:
One hundred percent correct. Yeah, I mean, we're starting to hear more and more about forks of some of the LLMs, and I think once you start seeing that the results that we're getting from the LLMs today. And, by the way, the same can be said about the product owner, the architect, and all of those things. The reason that that number isn't higher is because we still need human intervention to improve the results. The less human intervention that we need to get something that the human says is of quality is just accretive to that number.

Mike Kavis:
Yeah. So, the moral of that story is as the technology improves, the numbers should improve with it. Well, cool. Well, that's all the time we've got for today. Tell us where we can find you out there. Where can we find any of your content if you're still pumping it out there? And where can we find you on the socials?

Ryan Lockard:
Well, what I would say is the best place to find me is on the Deloitte Engineering website. I do have a couple pieces out there. You can obviously find me on LinkedIn. And if you're in the Dallas area I'll be rejoining the conference speaking circuit. I have a session at the Agile 2024 conference in, I think, July in Grapevine.

Mike Kavis:
Cool. Well, it's great having you. And we're going to have to check back in in about six months to see how advanced everything's got.

Ryan Lockard:
Sounds good.

Mike Kavis:
So, we'll book you later down the year. So, that'll do it for this version. I hope you enjoyed the podcast. Make sure to like us, leave a review and subscribe. You can also check out all of our past episodes in whatever medium you like to follow your favorite podcast. You can always find me on Twitter, @madgreek65 or reach out to me directly at mkavis@Deloitte.com. Feel free to write me with questions and let us know about future topics you may want us to talk about. Thanks for listening and we'll see you next time on Architecting the Cloud.

Operator:
This podcast is produced by Deloitte. The views and opinions expressed by podcast speakers and guests are solely their own and do not reflect the opinions of Deloitte. This podcast provides general information only and is not intended to constitute advice or services of any kind. For additional information about Deloitte, go to Deloitte.com/about.