Google Just Dropped a Masterclass on Agentic Engineering (It's SO Good)

21 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=ZBMUIAPUINM
https://www.youtube.com/watch?v=zbmuiaPuiNM

SUMMARY

The video discusses a new masterclass on AI coding released by Google, which outlines the evolving landscape of AI-driven software development life cycles (SDLC). The presenter emphasizes the importance of understanding the entire process from idea to production, highlighting the shift from traditional coding methods to AI-assisted approaches that streamline implementation but still require human oversight for requirements gathering and validation.

- *The AI-driven SDLC encompasses the entire process from idea generation to production, emphasizing the importance of both upfront requirements gathering and final validation.*
- *Implementation time has significantly decreased with AI coding assistance, but bottlenecks remain in the specification quality and validation stages.*
- *AI coding is described as a spectrum, ranging from casual "vibe coding" to structured "agentic engineering," with the latter being more reliable for production-level code.*
- *The "harness" is crucial, comprising context, rules, tools, and workflows that enhance the AI coding assistant's effectiveness, with the model itself being only 10% of the system.*
- *Context management is vital; static context provides reliability, while dynamic context allows for scalability and efficiency.*
- *Engineers should adopt roles as both conductors (micromanaging individual tasks) and orchestrators (overseeing larger projects), depending on the task complexity.*
- *Initial investments in building a robust harness lead to lower operational costs over time, as agentic engineering becomes more efficient than vibe coding.*
- *Continuous improvement of the system is encouraged, with a focus on evolving the harness to enhance reliability and performance.*

So, a new master class on AI coding was just dropped by Google, and it is really good. It's a highle overview of pretty much everything that I teach on my channel. In fact, a couple of people actually sent this to me last week, and they said, "Hey, Cole, this literally looks like it could have been written by you. It's the cleanest packaging I've seen for everything that the industry is converging on right now as far as best practices and terminology for AI coding.

It's very well written, definitely worth a read. So, I'll link to it in the description, but it's also 51 pages long, so it takes a while to get through this, which is why I wanted to make this video just to disseminate everything nice and quickly for you. And even if you're already pretty comfortable with agentic engineering and AI coding, it's worth going through this, right? The old adage is you don't truly understand something until you can teach it

well.

So, it's important to take the instincts you build over time and turn that into a clear visualization, mental model, and precise terminology. And so that's what we really get with this on everything the industry is converging on. And so I've reordered things a little bit what I'll show you here. I think there's a better ordering than what they present. But I want to go through this all with you along with a diagram that I have prepared and just give the good parts to you really fast. So let's get into the meat of it here. So the first big question we have to answer here is what the heck even is an SDLC? If you don't come from a technical background, you're probably not even familiar. And there's the new phrase AIdriven SDLC. that's being thrown around all of the time now.

So, it's short for software development life cycle. And quite simply, it's the process to go from idea all the way to production. So, requirement gathering at the start all the way to review, deployment, and maintenance. And so, it's a lot more than just writing the code that sits in the middle. And with a traditional SDLC, you spend a good few days gathering requirements with your stakeholder meetings and the product manager creating the PRD, like all that documentation upfront. And then you have a couple of days of designing and then the implementation is usually what would take most of the time. The engineer spending weeks writing the actual code before you then go into the final steps of testing, reviewing, deploying and maintenance. And usually that would take a week. Obviously, it depends a lot per company. Just a general idea through generalization here. And so now with the AIdriven SDLC, the important thing here is that everything we do up front and at the very end, it's not actually that much faster. Now the specification quality is the new bottleneck. And that is so true because there's so much that still has to be human-driven with the validation at the end and the requirement gathering up front. And so really, it's only what's in the middle here. The implementation has gone from 1 to 3 weeks to minutes or hours with AI coding assistance. The same thing is dozens of times faster, especially because agents can iterate with their own system of tests and eval.

And so we have the bottleneck at the start and at the end. I firmly believe that a lot of the next $1 billion plus companies are going to be platforms that help speed up the requirements gathering and the validation because we've solved way more for what we have in the middle now. And so that's why you hear so many statistics around like AI coding assistants 10xing the engineers output but not actually 10xing the output of the business is because we're bottlenecked by other parts of software engineering. Software engineering is a lot more than just writing code. But the thing is, as much as you can, you want to remove implementation as the bottleneck because that is still going to save you a considerable amount of time. And so doing that and just generally making everything else in the AIdriven SDLC as fast as possible is what this article focuses on. And so that brings us to the first thing that I want to cover in the diagram. So I took all the big long ideas from the article, made it nice and concise for you here.

And so the first thing that they talk about is that AI coding is a spectrum, not a switch. And I really appreciate that because most people think of it as something that's binary. Either you're vibe coding or you're doing agentic engineering. But it is a spectrum depending on the level of your system. And so we'll talk about the system and the harness in a bit. But vibe coding is where you send in a prompt without much planning. And then your validation is, hey, does it seem like it work? Right?

like you'll test the application a little bit and then you'll just move on to the next iteration. With structured AI assisted, we have more detailed prompts. We're doing more spot-checking and then we get all the way to aentic engineering where we have a entire engineered set of resources and workflows for our AI coding assistant with specs and automated evals and CI gates. So, the agent has a way to really iterate and figure out things that go wrong before you have to correct it.

This is where the real power comes in. And so it's not like we always need agentic engineering. Sometimes vibe coding is actually enough for proof of concepts or you just want to create an MVP. I used to just always dismiss vibe coding. But I think there is genuinely a place for it. And so the spectrum Google is saying is not just like you're evolving yourself. It's you pick the right one for the job. It's just agentic engineering is usually where you want to be because this is where you're really creating reliable code. And in the article, Google also has this table that I really appreciate. It makes things nice and concrete. So for each level, what does it look like for these different dimensions? And so for intense specification, for example, which is just how do you communicate upfront what you want. For vibe coding, it's just casual natural language prompts. So you're just describing at a very high level what you're looking for. With structured AI assisted coding, the middle of the spectrum, you're getting more detail, but you still don't really have a workflow for creating formal specs, architecture docs, like when you get to aentic engineering, this is where you really have a repeatable process and you have specifications that are actually engineered just like the code.

And then for verification, like we covered this a little bit already, but for Vive coding, it's more does it just seem to work. You're not doing much of a deep dive at all. With structured AI system coding, you're getting a little bit into it with more manual testing and spa-checking of the code maybe. And then for agentic engineering, this is where you have the whole process for the agent to iterate itself with tests and CI/CD gates. Also, LLM judges, you have a separate code review process for yourself and another agent. And I don't need to cover everything here, but getting down to the risk profile with vibe coding, it's high, right? like acceptable for disposable code like I was saying earlier but then if you really want the most reliable code possible that's where you want systematic verification at every stage that comes with aentic engineering okay so if aentic engineering is the way to go most of the time how do we actually do it like what separates aentic engineering from vibe coding and really everything can be wrapped up in the harness so the harness is the set of context rules tools and workflows that you bring into the AI coding assistant.

It's the layer that you control. And the big thing that Google is claiming here is that the large language model that you use for your AI coding assistant is only 10% of the system or it only matters 10%. Everything else like your instructions and tools and context and guardrails and orchestration and observability like there's so much here that makes up the other 90%. And that's actually a really good thing because the model is what we don't control. The harness is what we get to create for our specific code bases, architectures, and tech stacks.

And it really is true that the industry is converging on a lot of these things. Like we have this article from Anthropic that I covered a couple of weeks ago on my channel, just best practices for using cloud code in general. And one of the headlines that they have here is that the harness matters as much as the model. And so now Google and myself as well were taking this even further to say not only does it matter as much but it

actually matters more than the model.

Like the model only being 10%. Clearly Google is like okay you need to put your focus on the rest of the harness here. And they also have a very similar definition of what goes into the harness. So cloud code right here they say it's your global rules. It's your hooks like the deterministic actions you want in your life cycle your skills. So, the workflows that you have packaged up, your ways that you search your codebase, the MCP servers, and your sub aents, like these are all of the primitives, as I call them, for working with literally any AI coding assistant.

And if we go now into Google's article here, they say the agent is the model plus the harness. And they have this diagram that lays out exactly everything that goes into the harness. And you can see this is where I got the numbers, by the way. So, the model being 10%. So you have the large language model in the middle still matters to an extent because it is the brain. It is the reasoning in your system but everything else around it is a huge deal. So you have your instructions MCP servers guardrails and hooks. I mean everything is the exact same as what anthropic presented in their article. And then the layer above is where you have all of the testing infrastructure. So the eval to iterate itself. And then the top layer is more for you and for production. So the observability and tracing, the scaling, right? Like that's pretty important when you want to take anything an AI coding assistant produces and actually take it all the way to production. The sponsor of today's video is Better DB, a self-tuning Valky/Ris caching and observability platform for AI agents, and it is open-source. So, we're talking all about the AI SDLC in this video, but not covering that much tools we can use to help us with reliability and monitoring in production, that end stage of the SDLC.

And better DB is a fantastic example of an AI native tool that can help us with this. So, monitoring our database in production, using our AI coding assistant with it to suggest changes and improvements based on live production data, and a semantic cache to help us scale our database. Let me show you how these things work really quick. My favorite part of Better DB is the semantic cache. Take a look at this. You'll see how it works very quickly. If I ask what's the capital of France, it's not in my Better DB cache yet, so it's a miss. And it calls a model to get the answer. But the next time I ask something that is similar, we get a cache hit. It doesn't even have to be the exact same wording because it's semantic similarity search like traditional rags. So, we get a much faster answer. And we have an MCP server so we can connect our AI coding assistance directly to our better DB cache. So we can ask how it's doing. We can have it suggest improvements and even make those directly. So it's very easy to improve our system over time with the help of AI. And then also we have a dashboard to monitor everything.

So we can see how our agent and our cache is performing in production with real user data. And the best part is better DB is open source and free to get started. So I'll have a link in the description. I'd highly recommend them as a tool to help you scale manage your costs for agents you're deploying to production. And so now Google is saying with harness engineering we have the idea of the factory. So instead of the engineer writing the code or the product manager writing the PRD by hand, instead we are responsible for designing the system, creating the harness and then the agent is the one that is actually producing our code and documentation.

And so this is more of an investment upfront than vibe coding because we have to create the specs and

guardrails, but then we use that to then go into this repeatable system of we plan with the agent, we have it build, and then we have our quality gates at the end for testing and evaling with an iterative loop here for the agent to improve its output autonomously and then get to the point where we have something for us to review and ship. And so this entire thing, we want to delegate all of the coding to the AI coding assistant. Even with agentic engineering, you are delegating all of the coding. So this is not a spectrum of how much do we write by hand versus trust the agent. It is just a spectrum of how evolved of a system do we actually have here. So Google does get a little bit repetitive here because when they talk about the factory model for the first time and what goes into it, it's really the same thing as what goes into building a harness or the AI layer they already talked about. So it's your your context and rules, your test and quality gates, your workflows, your guardrails and your hooks, right? They have a really good visualization for where the developer actually stands in the process. Now, so we define our specs, context, and requirements up front and you use those specs for your planning agent. So every single time you build anything with an AI coding assistant when you're doing agentic engineering is you're going to have one agent that does the plan for the bug fix, for the new feature, whatever it is. And then the guard rails that you design and like the sandboxed environment, that is what's going to be used by the actual coding agent. But it's important here that you do split this into two separate sessions because your planning agent is going to build up a lot of context. You want to avoid context rod and it's going to build up a lot of bias. And so you take the plan as an artifact. You send that into the coding agent and then you do your test and verification and iterate there. And this is also where we can come in the loop to review and approve things ourselves because you definitely fall more into vibe coding if you're not reviewing the output yourself. Even if you do have quite an autonomous system, right? Like even if it's just that pull request at the end for agentic engineering, generally you want a human to be reviewing that before you mark it as pass and you go on to the rest of the process for deployment to production.

And throughout this entire workflow, that's where we have our guardrails like token limits and security policies, everything that you are engineering upfront. And the really cool thing about this whole system is that we can make it better over time. Just like we evolve our codebase over time, we can evolve our system. So I I call this the system evolution mindset. Whenever you encounter an issue with your AI coding assistant, like something comes up here where it has to iterate more than you would want or you have to step in before you ship, instead of just fixing the bug and moving on, you actually talk to your coding agent like you have it do some retrospection and say, "Hey, where could we make our workflows or our rules like any part of our AI layer better so that issue is less likely to come up again?"

And so that way every single time you go through this process over and over and over again, you're making it more and more reliable. And the harness is worth investing your time into. Like it it really is the 90%. I mean, there's a lot of studies that are done like terminal bench 2.0. It's one of the biggest benchmarks we have out there. Like every single time a new model comes out, this is one of the percentages that you see.

There's a lot of studies done where like they were able to take a model from outside the top 30 into the top five just by creating an AI layer of rules and workflows for it to run through the things you usually test for the benchmark. Lane chain was able to increase it 13.7 points. Like that's the difference between Sonnet and Opus. Like you can make sonnet work as well as Opus if you have the right system, the right process that you're having it go through as the harness. So if the harness is the most important part of agentic engineering, then it's clear

that the most important skill within that is how do we engineer each of the individual components of the harness like our rules, workflows, and a guard rails. And so we've covered the different components already, but a key delineation that Google makes here that I really like is the static context versus dynamic context. And this is really important because it's all about context management. Context is your most precious resource when working with AI coding assistants, both for the sake of cost and avoiding context rot. We don't want to fill the window of our LLM, our coding agent, too much because LLMs get overwhelmed with information just like people do. And so nice visualization here. They talk about what goes into static versus dynamic. So static context is things like your rules and core guardrails, the system prompt. It's loaded into the coding agent session guaranteed every single time. time. That makes it reliable because the agent doesn't have to seek out this information, but it's expensive because you're filling the context window up front. And so, it's important to have at least some rules and guard rails up front, but you want to make them very lean. And then everything else goes in dynamic context so it's efficient and scalable because it's information that the agent has to actually seek out. Like you might have an an agent skill for planning like it loads that skill when you want it to do the planning workflow or you have conventions for a part of the codebase you want it to load when it operates on that part of the codebase and so it's very scalable so you're not shoving it into the context up front but the risk there is the agent might not grab for that context when it should like it might not load the skill or perform the rag search when you would hope it to or when it would be optimal to do so. But large language models are getting better and better at relying on dynamic context and loading it when it should. And so like agent skills are becoming very very important right now, right? So they say rather than embedding every piece of specialized knowledge into the agent system prompt, skills allow the agent to remain a lightweight generalist that flexes into specialist roles on demand through progressive disclosure. And this is so important because the underlying lesson here is that we really only need one agent for everything and then we can make it specialized with our skills, i.e. our workflows. And so something that people used to do way too much before is they would have these really complicated multi- aent systems with all these specialists or they use a ton of these specialized sub aents they would create.

And really the industry is moving away from that because we can just have one generalist agent that we make specific with the skills that we have at load. Like we can have it become a code reviewer or become a planner. That session can turn into the specialization that you need thanks to dynamic context. So keep it simple. You really only need one agent to drive most of your agentic engineering. Okay. So the article has been very valuepacked already. There's just two more things that I want to cover with you here. I want to talk about your role as the conductor and orchestrator and then also the token economics. And so an interesting thing that Google presents here is the idea of you as the engineer are going to move between two modes as you're using your AI coding assistant. And so the conductor is more how we used AI coding assistants when generative AI was first a thing. Like we had our tab complete.

We're still steering every move, working in individual files. That's the conductor. The orchestrator is a lot of what people have been focusing on more recently where we have a coding agent handling much larger tasks spanning entire code bases, maybe even multiple code bases. We're reviewing the outcomes instead of changes to individual files. We have agents running in parallel. We're really scaling our output with AI coding assistance here. And almost everybody is focusing entirely on this. And this this is like the one part of the article I don't know if I agree with Google because they're saying that you actually want to move between both. Like there's

still a time and place to be micromanaging the AI coding assistant at a single file level. Honestly, I don't know if I agree with this. I think when you build the harness to be reliable enough and you're confident in your rules and workflows, you can always live at this level. But they do make some interesting arguments where it's like any kind of like deeper debugging you have to do or just initial exploration like you are going to get very granular with the coding agent because that's the times where you might need to really be in the loop and guide it. So I think there's a time and place for it but I feel like when you have the right system and it's working well for you, you don't really like you kind of graduate from being the conductor. I don't think you're always moving between the two.

But it is an interesting idea you know especially as an organization when you have a lot of traditional engineers and you're first getting into aentic engineering I think it is good to have this mental model just until you have the system developed where you'd graduate to only ever staying here. Cool. And then the very last thing that I want to cover here is the token economics. I really love how they frame things here. So, like we said, vibe coding, you don't always want to avoid it, but there is a big cost that comes if you lean on it too much because at first when you're first adopting AI coding assistance for yourself or a company, Vive coding is going to be cheaper. It's lower capital expenditure because you don't have to dedicate yourself or a team to design the initial harness. But the problem is it's very high operational expenditure because you start burning through millions and millions of tokens iterating on slop code because you don't have a system for your AI coding assistant to follow your workflow and your conventions. And so agentic engineering it has that high capital expenditure because you have to dedicate your time up front or you have to like in a larger organization usually you create a smaller forward deployed engineer team to build up that harness to then scale to the entire organization. So you're dedicating manpower to build something initially, but then it scales extremely well because the output of your AI coding assistants are better and better and better over time and you have that grounding in a system that you just build once upfront and evolve over time.

So high capital expenditure but then low operational expenditure and you know you have that crossover that you reach extremely quickly like you want to just take the dive and build that system up front because yeah you're going to get to the point where agentic engineering is three to 10 times more reliable and cheaper than vibe coding because you're not burning through millions of tokens. So there you go. That is everything you need to know at a high level for the new AIdriven software development life cycle. It is worth building that harness and investing in it. It is an engineered resource that lives in version control just like the code itself. So, I hope that you found this useful. Let me know in the comments what kinds of content you want me to create to expand on any of these ideas here cuz this is my bread and butter. If you appreciated this video, you're looking forward to more things on Agentic Engineering, I would really appreciate a like and a subscribe. And with that, I will see you in the next video.