Full Walkthrough: Workflow for AI Coding — Matt Pocock

96 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=-QFHIOCO-KO
https://www.youtube.com/watch?v=-QFHIoCo-Ko

SUMMARY

Matt, a teacher specializing in AI, discusses the integration of AI into software engineering, emphasizing the importance of foundational coding principles while working with AI. He introduces concepts such as the "smart zone" and "dumb zone" of large language models (LLMs), the significance of breaking down tasks into manageable parts, and the necessity of maintaining a clear understanding of codebases to enhance AI's effectiveness.

- *AI is transforming software engineering, but traditional coding principles remain crucial.*
- *LLMs operate within a "smart zone" and "dumb zone," where task complexity can lead to decreased performance.*
- *Effective task management involves breaking down larger tasks into smaller, manageable phases to stay within the smart zone.*
- *The "Grill Me" skill helps align developers and AI by asking probing questions to clarify project requirements.*
- *Using a Kanban board allows for parallel task execution, optimizing workflow and efficiency.*
- *Test-driven development (TDD) is essential for ensuring quality and feedback loops in AI-generated code.*
- *Maintaining a clear architecture with deep modules enhances AI's ability to navigate and improve codebases.*
- *Continuous QA and code review are vital to ensure high-quality outputs from AI implementations.*

[music] >> Yeah, we're good. Okay, folks. We're at capacity. Let's kick off. I don't want you waiting here for 25 more minutes before we some arbitrary deadline. So, welcome. My name's Matt, I'm a teacher, and I suppose now I teach AI. Um We have a link up here, if you've not already been to this, which is has the exercises for the um stuff we're going to do today.

This is going to be around 2 hours, so we might just sort of kick off 2 hours from now. Is that all right, Mike? Yeah, perfect. Um and the theory behind this talk, or at least the thesis under which I've been operating for the last kind of 6 months or so, is that we all think that AI is a new paradigm, right? AI is obviously changing a lot of things. You guys are obviously interested in this, and that's why you've come to this talk.

And I feel that when we talk about AI being a new paradigm, we forget that actually software engineering fundamentals, the stuff that's really crucial to working with humans, also works super well with AI. And this is what my keynote is on tomorrow, really. I'm going to sort of be fleshing that out a lot more. And in this workshop, I'm hopefully going to be able to direct your attention to those things, and uh hopefully show you that I'm right. But we'll see.

Um can I get a quick heads-up first? How many of you guys um are coding have ever coded with AI? Raise your hand if you've ever coded with AI. Perfect. Okay. Uh keep your hand raised. Uh let's all uh share those armpits with the world. Um how many of you code every day with AI? Cool. Okay. Uh right, keep your hand raised if you've ever been frustrated with AI. Okay, very good. You can put your hands down.

Thank you for that show of obedience. I really appreciate that. And we are also being live-streamed to the Gilgood room as well. I've not uh Did we send someone up to the Gilgood room to just check they're okay? Don't know. But I see you, and there is a way that you can participate, which is we have the um a Q&A. We're going to be doing kind of have a sort of hatred of Q&As cuz they're not very democratic. They're mostly the sort of um most talkative people get to um get to participate and share. And so, we're going to be going through this um Q&A here. So, why do we have to wait till 3:45? The room is packed, the doors are closed. 100% agree.

And so, if you want to uh ask a question, we're going to be I would like you to pile into this async, and then we can vote on each other's questions, and hopefully get the best questions surfaced so the for the entire room to enjoy. So, I want to talk about first the kind of weird constraints that LLMs have. And those weird constraints are sort of what we have to base a lot of our work around.

Now, there's a guy called Dex Hardy who runs a company called Human Layer, and he came up with this idea, which is that when you're working with LLMs, they have a smart zone and a dumb zone. When you're first kind of like working with an LLM, and it's like you've just started a new conversation, you start from nothing, that's when the LLM is going to do its best work. Because in that situation, the attention relationships are the least strained.

Every time you add a token to an LLM, it's kind of like you're adding a team to a football league. You think of the number of matches that get added every time you add a team to a football league, it just goes it scales quadratically. And that's because you have attention relationships going from essentially each token to the other that are positional and the sort of meaning of the individual token. And so, this means that by around sort of 40% or around I would say around 100K is kind of my new marker for this. Cuz it doesn't matter whether you're using 1 million uh context window or 200K, it's always going to be about this.

It starts to just get dumber. So, as you continually keep adding stuff to the same context window, it just gets dumber and dumber until it's making kind of stupid decisions. Raise your hand if that feels familiar to you. Yeah, cool. So, this means that we kind of want to size our tasks in a way that sticks within the smart zone. Right? We don't want the AI to bite off more than it can chew. This goes back to old advice like Martin Fowler in refactoring. Uh like uh the pragmatic programmer talks about this. Don't bite off more than you can chew. Keep your tasks small so that you as a developer, a human developer, don't freak out and don't start acting and going into the dumb zone.

But how do you tackle big tasks? How do you take a large task like I don't know, cloning a company or something, or just doing something crazy, and how do you break it into small tasks so they all fit into the dumb zone? One way, of course, you could do is I mean, kind of what the AI companies maybe want you to do, or the

natural way of doing it is just keep going and going and going, you end up in the dumb zone, charging you tons of tokens per request.

You then compact back down. We'll talk about compacting properly in a minute. And you keep going, keep going, keep going, compact back down, keep going, keep going, keep going. And I think that's doesn't really work very well because the more sediment I we'll talk about that in a minute. So, the theory here is then, and this is what I was doing for a while, is I would use these kind of um multi-phase plans. Where I would say, "Okay, we have this sort of number four thing here, this large large task. Let's break it down into small sections so that we can then kind of chunk it up and do each little bit of work in the smart zone." Raise your hand if you've ever used a multi-phase plan before.

Yeah, really common practice, right? This is kind of how we've been doing it. Certainly, this is how I was doing it up until December last year, really. And any developer worth their salt will look at this and go, "This is a loop." Right? This is a loop. We've just got phase one, phase two, phase three, phase four. Why don't we just have phase N? Right? Phase N. Where we essentially just say, "Okay, we have, let's say, a plan operating in the background, and then we just loop over the top of it, and we go through until it's complete."

And this is where um Raise your hand if you've heard of Ralph Wiggum as a software practice. Okay, cool. Raise your hand if you've not heard of Ralph Wiggum as a software practice, actually. That's more like it. Okay. So, there's this idea called Ralph Wiggum, uh which is kind of um sort of based on this, which is essentially all you need to do is sort of specify the end of the journey, where you just say, "Okay, we create a PRD, a product requirements document, to say, 'Whoa, okay, let's describe where we're going.'" And then we just say to the AI, "Just make a small change. Make a small change that gets us closer and closer to that."

And Ralph works okay, but I prefer a little bit more structure. So, that's kind of where we got to in terms of thinking about the smart zone, and that's kind of where I want you to first start thinking about here. Another weird constraint of LLMs is LLMs are kind of like the guy from Memento, right? They just continually forget. They could just keep resetting back to the base state. Let me pull up this diagram. I sort of I I I really should use slides, but I just prefer just like randomly scrolling around a uh infinite uh TL draw canvas. Thank you, Steve.

Um So, let's say another concept I want you to have is that every session with an LLM kind of goes through the same stages. You have, first of all, the system prompt here. This gray box here is essentially the stuff that's always in your context. You want this to be as small as possible. Cuz if you have a ton of stuff in here, if you have 250K tokens, like I have seen people put in there, then that you're just going to go straight into the dumb zone without even being able to do anything.

So, you want this to be tiny. >> [snorts] >> You then go into a kind of exploratory phase. This blue sort of where the coding agent is going out and exploring the code base. Then you go into implementation. And then you go into testing. And sort of making sure that it works, running your feedback loops and things like this. Raise your hand if that feels familiar based on what you've done. Yeah. Sort of the like the the main cornerstones of any session.

And when you clear the context, you go right back to the system prompt. Oof, you go right back there. So, you delete everything that's come before. And raise your hand if you've heard of compacting, as well. Yeah, okay. There are some people who've not heard of compacting. So, let's just quickly show what that means. For instance, I've just been having a little chat with my LLM. Uh I want to make sure we sort of, you know, just cover the basics so we're all sort of on the same wavelength here.

I've just been having a chat with my LLM. I've been talking about a thing that I want to build. How's the font size? Should I bump it up? Folks in the back? Bump. Bump. Bump. Bump. Bump. Oh. I'm using Claude Code for this session, but you don't need to use Claude Code. Uh in fact, it's often nice not to use Claude Code. Um so, I've been having a chat with the LLM, just sort of planning out what I'm going to do next. It's asking me a bunch of questions, and I can I highly recommend you do this.

There's this tiny little status line here that tells me how many tokens I'm using, the exact number of tokens I'm using. Um I have a article on my website AI Hero if you want to copy this. This is Oh, wow, that is that shakes, doesn't it? Um this is essential information on every coding session cuz you need to know exactly how many tokens you're using so that you know how close you are to the dumb zone.

Absolutely essential. And so let's watch it. So I've got two options. I can either clear wrong and go back to nothing or I can compact. And when I compact then it's going to squeeze all of that conversation, which admittedly isn't very much, into a much smaller space. And this in diagram terms kind of looks like this. Where you take all of the information from the session and you essentially create a history out of it, a written record of what happened.

And devs love compacting for some reason, but I hate it. I much prefer my AI to behave like uh the guy from Memento because this state is always the same. Always the same every time you do it. You clear and you go back to the beginning. And so if you're able to do that and you're able to optimize for that then you're in a great spot. So that's kind of the two things I want you to think about with LLMs, the two constraints that we're working with.

They have a smart zone and a dumb zone and they're like the guy from Memento. So let's take a look at the first exercise. And I'm while I'm doing this, the way I want this to work is I'm going to sort of show you how um I'm going to be sort of walking through it up here and I want you folks to be kind of like tapping away and doing things as well. So that was just a little lecture bit. Let's now actually get and do some coding.

For anyone who arrived late or anyone in the Gilgud room uh go to this link this link up here to see the exercises and clone the repo. You absolutely do not have to, you can just watch me do it if you fancy it. But let's go there myself and let's see what exercises await us. So essentially I've built a um this is from my course. This is a uh a course management platform essentially, a kind of CMS for instructors, for students, and this is what we're going to be building a feature in. So I'm going to take you from essentially the idea for the feature all the way up to building a PRD for the feature, all the way up to implementing the feature.

And hopefully you can take inspiration from this process and use it in your own work. So uh let's kick off. So we're going to start by using a a skill which is very close to my heart. It's the grill me skill. And this grill me skill is wonderfully small wonderfully tiny and it helps prevent one of I think the main issues when you're working with an AI, which is misalignments. The uh the sort of silent idea that I'm talking against here, that I'm arguing against, is the specs to code movement. Has anyone heard of the specs to code movement? Raise your hand. It's not really a movement I suppose, it's just sort of people saying specs to code.

Um what it is is people say, "Okay, you can write a program or you want to build an app the best way to build that app is to take some specifications so to write some sort of like document and then turn that document into code." So they just turn it into code. How do you do that? You pass it to AI. If there's something wrong with the resulting code, you don't look at the code, you look back at the specs. You change the specs and you sort of just keep going like this. This is kind of like vibe coding by another name where you're essentially ignoring the code.

You don't need to worry about the code. You just sort of keep editing the specs and eventually you just keep going. And I tried this. I really tried it. And it sucks. It doesn't work. Because you need to keep a handle on the code. You need to understand what's in it. You need to shape it because the code is your battleground. And so this is again is where we're going. Let's let's get some exercises. So what I'd like you to do is go to this page, the the grill me skill.

And inside the repo here we have a slack message from our pal. Uh where is it? It's in the root of the repo and it's under bur bur bur bur Oh, where is it? Mhm mhm client brief.md. It's a slack message from Sarah Chen. For some reason the Claude always chooses Sarah Chen as the name. I don't know why. Um it's saying that in cadence, our um course platform, our retention numbers are not great. Students sign up to a few lessons then they drop off. I'd love to add some gamification to the platform.

And so when you're presented with an idea like this, you need to find some way of turning it into reality. Let's say Sarah Chen is your client, you're on a tight budget, you need to get this done fast. How do you go and do it? Um raise your hand if you would um enter plan mode when you're doing this. Anyone a big user of plan mode? Yep. Um let's actually shout out quickly any other ideas about what you would do with this or any Raise your hand if you what what would be your first port of call?

Yep. Ask for more info. Sorry? Ask for more info to verify what is the purpose and where our current standing is. Yes, exactly. Let's imagine that Sarah Chen's gone on holiday, you have no idea, right? Uh she's just posted this thing, you need to action it before you go. Well, my first port of call is I go for this particular skill. I'm going to clear my context. I'm going to uh get rid of you, you don't need to be there.

And I'm going to say um I'm going to invoke a skill which is the grill me skill. Let's quickly check. Raise your hands if you don't know what this is. Cool. Oh, sorry sorry. Let me be more specific. Raise your hands if you don't know what I'm doing here when I uh do a forward slash and then type something. Anyone Everyone kind of understand what that is? I'm invoking a skill. I'm invoking the grill me skill.

And what I'm going to do is I'm going to say grill me and I'm going to pass in the client brief. So now the LLM really has only a couple of things here. It just has the skill and it has the description of what I want to do. And this is virtually how I start every piece of work with AI. And while it's exploring the code base I'm just going to show you what the grill me skill does.

So this is inside the repo so you can check it out. It's extremely short. "Interview me relentlessly about every aspect of this plan until we reach a shared understanding. Walk down each branch of the decision tree resolving dependencies one by one. For each question provide your recommended answer. Ask the questions one at a time uh blah blah blah." What this does and what I noticed when I was working with AI, especially in plan mode actually is it would really eagerly try to produce a plan for me.

It would say, "Okay, I think I've got enough. I'm just going to poof plan plan." And what I found was that I was really trying to find the words for this, for for what I wanted instead of that. And Frederick P. Brooks in The Design of Design, he has a great quote uh talking about the design concept. When you're working on something new with someone when you're uh all trying to build something together then there's this shared idea that's shared between all participants and that is the design concept. And that's what I realized I needed with Claude. I needed I needed to reach a shared understanding. need an asset, I didn't need a plan, I needed to be on the same wavelength as the AI, as my agent. And this is an extremely effective way of doing it. So hopefully Here we go. Nice. It has done its exploration first of all.

It's invoked a sub agent which spent 97 93.7k tokens on Opus. Um and it's asked me the first question. Cool. We can see that even though the sub agent burned a a ton of tokens I haven't actually um uh increased my token usage that much. Raise your hand if you don't know what sub agents are. It's important question. Everyone kind of clear what sub agents are? Okay, I'll give a brief definition. Which is that this this sub agents thing here, this explore sub agent it has essentially gone and called another LLM which has an isolated context window.

And then that LLM has reported a summary back. So a sub agent is kind of like a delegation. You're delegating a task to a sub agent. It goes eagerly does all the thing, explores a ton of stuff and then just drip feeds the important stuff back up to the orchestrator agent. To the parent agent. So okay. So hopefully you guys have seen the same thing. It's done an explore. And we now have our first question. Points economy. What actions earn points and how much? Ooh, okay.

At this point you can ask it by the way questions to um deepen your understanding of the repo. I obviously know this repo really well cuz I wrote it, but you might not um know what's going on. So let's say my recommendation, keep it simple, two point sources to start. What's so nice about this is that not only does it give us a question that kind of aligns us here, we get a recommendation too. And often what I'll find is the AI's recommendations are really good.

And so I'll just say skip video watch events, they're noisy and gameable. I agree. Sarah's asked we'll keep the lessons in the bread and butter. Yeah. Looks good, pal. >> [snorts] >> Now what I usually do is I usually dictate to the AI. I'm usually actually chatting to the AI instead of uh typing here, but uh this is a relatively new laptop

and I couldn't get my dictation software working on it um because Windows is crap. Um So, should points be retroactive? There are existing lesson progress records with completion at timestamps. This is a really nasty question, right? Should we actually go back and backfill all of the lesson progress events? This is a kind of question that you need to be aligned on if you're going to fulfill the feature properly. This is not something I considered and Sarah Chen certainly didn't consider.

Do I want it to be retroactive? Hmm. Let's actually do a vote inside here. Should we go back and backfill all the records? Raise your hand if you think we should backfill all the records. Raise your hand if you think we shouldn't backfill all the records. There are a lot of fence-sitters in the room. I'm going to say you know, this is the kind of discussion you're sort of having with the AI. You're getting further aligned. Yes, I'm just going to go with his recommendation cuz I'm lazy.

Notice too how I'm able to keep in the loop here with AI. I'm not you know, it's it's pinging me these questions pretty quickly. I'm not having to go off and check Twitter or something. Levels. What's the progression curve? Yeah, that looks about right. For instance, yes, okay. So hopefully you should be able to go and um kind of work through this with the AI. >> [clears throat] >> And essentially try to reach an alignment. And this grill me skill, this can last a long time. This can I've had it ask me 40 questions. I've had it ask me 80 questions. I've had some people that asks 100 questions too. Literally you're sat there for an hour chatting to the AI.

And what you end up with is essentially this conversation history that works really nicely and works really nicely as an asset of the design concept that you're creating. This can also function like this. You can have a meeting with someone who's a maybe a domain expert. Maybe I have a meeting with Sarah. I feed that meeting transcript into I don't know, Gemini meetings or whatever you guys are using. You take that, you feed it into a grilling session and you grill through the assumptions that you didn't have.

So this ends up being a really nice kind of um a really nice way of just taking inputs from the world and then just turning and validating them. So okay. Let's see. I really want to get to the end of this, but I also don't want to just like be sat here talking to the AI in front of you for uh a thousand days. So I'm just going to say yes. Let's see what happens. So I'll tell you what, um while you guys sort of have a little fiddle with this locally, let's start a little Q&A session now.

And let's see. How's this going to work? Can we keep the door closed or turn up the microphone? It's quite noisy. Uh let's see. Mike, can we uh door closed. Oh it has been closed. Mark has answered. Beautiful. So what I'd like you to do is there any air con? Yeah, there is some air con, I think. There is some air con. You guys aren't being lit here. I'm being fro I'm being fried alive here. Uh so what I'd like you to do is go on to the Slido, which you can join here.

Have a if if you're not taking the exercise, go on to the Slido, have a little fiddle and vote on some good questions. I'm just going to chat to the AI for a second uh until we reach a stopping point. So do streaks earn points? Um streaks are standalone. Let's see what else it comes up with.

Where does gamification UI live? Let's have it in the dashboard. I'm just going to scan these and blast through them basically. So how are we doing with our Slido? Okay. Have I tried Spec Kit, Open Spec or Taskmaster instead of the Grill Me skill? Do I find them more verbose or a structured alternative? This is a great question. So there are a ton of different frameworks out there that allow you to um sort of build up this planning process for you. I personally believe you at at this stage, when there's no clear winner, when there's no kind of like one true way and when things are changing all the time, you need to own as much of your planning stack as you possibly can.

What I've noticed and a lot of my students is they tend to overuse a certain stack. They get into trouble and they because they don't own the stack and they don't have observability over the whole thing, they just go this isn't working. This sucks. Whereas if um if you have control over the whole thing, then at least you know how to fix it or potentially know how to fix it. So I'm even though I'm sort of giving you uh a stack basically, I believe in inversion of control and you should be in control of the stack.

So bur bur bur. Can I press zero, please? Sorry? Sorry, that was a lot of sort of mumbling. Can I Thank you. I'm so sorry. >> [laughter] >> What you didn't want to give Claude good feedback? What is what is wrong with you? Uh okay, cool. Uh many of the questions asked by the Grill Me skill are not necessarily appropriate for a developer, rather a PO. In larger teams, who should use it?

Yeah. Um Raise your hand if um you've ever done pair programming. Anyone ever done pair programming? Right. I keep Put your hands down and raise your hand again if you've ever done a pair programming session with an AI. Right. How did it go? Was it good? You enjoy it? I think pair programming sessions with AI is a great idea because you've got a third person in the room who will relentlessly quiz you and ask you questions. It should If you don't know the answer, it should be you, the domain expert and the AI in the same room. If you're have a question about implementation, it should be you, a fellow developer and the AI in the same room, you know. You can be sort of working through these questions in your team. And I think actually we're going to look at implementation in a bit and we're going to see how you can make implementation so much faster.

And but I think the really crucial decisions, the ones you need humans for you actually need a lot of humans and it doesn't really matter how many humans are in there. You can actually throw a bunch like a kind of like mob programming with AI essentially. Uh what's my favorite meta prompting tool? I think I kind of answered that. Uh there's no air con. Let's just live with it. Uh how do I use the conversation as an asset after the Grill Me session? Well, we're going to get there.

Um okay, so I really want to I want to speed this up sort of artificially. Just what I This is the thing. So someone just said okay, Ralph loop this. But this is crucial because I can't loop over this, right? I can't um I think of there is being two types of tasks in the AI age. Where you have human in the loop tasks, where a human needs to sit there and do it. Which is this.

We are the human in the loop, with multiple humans in the loop. And there are AFK tasks. There are tasks where the human can be away from the keyboard and it doesn't matter. Implementation, as we'll see, can be

turned into an AFK task. But planning, this alignment phase, has to be human in the loop. Has to be. So I've got to do it, unfortunately. Um I don't know. Uh give me a long list of all your recommendations.

I'm running a workshop right now. So I artificially need you to pull more weight. So let's see what it does. Uh let's answer a couple more questions while it's doing its thing. What is my opinion on PMs or other non-dev roles vibe coding task? Hmm. Um I'm going to return to this later, I think. I'm going to leave this unanswered.

A bit of mystery. I notice I'm not using the ask user questions UI for Grill Me. Why? Um there's a specific uh UI that you can bring up in Claude Code. I'll answer this just quickly. Uh ask me a question using the ask user question tool. >> [snorts] >> And this UI um is just sort of broken in Claude and I really hate it. You notice I'm using Claude, but I don't like Claude very much. Like you you really are free with this method to choose any um system you like. And this is what the UI looks like.

It's very pleasing when you first encounter it, but then you realize it is actually broken in a ton of different ways. All right, what did it come back with? Oh blimey. Oh no. So while this is doing its thing, let me do some teaching in the meantime. The plan here is that we take our Grill Me skill and we need to essentially find some way of turning it into a destination. We need to go down to the uh We essentially need to we're figuring out the shape of this.

That's what we're doing. We're figuring out the shape of the tasks during the grilling session. And in order to turn it into a bunch of actionable actions for the AI we essentially need to figure out the destination. We need to know where we're going. We need to know the shape of this entire thing. So I think of there is being two essential documents that we need. We need a document that documents the destination. Oh no. It's so not bright enough. There we go.

Still not brighter. There we go. We need something to document the destination. And we need something to document the journey. In other words, we need something a document that's going to figure out what this even looks like in all of its user stories and figure out a definition of done and then we need to figure out what the split looks like. So, that's where we're going to go to next. So, once we finish with the grilling session, yeah, it looks great. Fantastic. I love it. It answered it answered 22 of its own questions.

There you go. That's quite representative of what a grilling session looks like. So, at this point now, I have used 25k tokens and all of that or loads of that stuff is gold. I want to keep that around. I've I've got 25k great tokens there. And what I want to do is kind of summarize it in some kind of destination documents. So, this is um the next exercise where we're going to uh we're going to write a product requirements document.

And the the product requirements documents or the PRD is essentially that's its function. It's the destination documents. And it's sort of doesn't matter what shape it is. I've got a shape that I prefer and I quite like. But, you can just choose your own shape or whatever your company uses. And all we're really doing is I'm not too worried about that. All we're really doing is summarizing the design concept that we have so far.

And the So, let let's try this. So, I'm going to initiate this. I'm going to say zoom all the way to the bottom. All I'm going to do is just say write a PRD. And we can take a look at that skill now. Write a PRD. So, this skill it does a few things. It first asks the user for a long detailed description of the problem. You can use write a PRD without grilling first, but I just like to grill first and then write the PRD afterwards.

Then you can um get it to install the repo which we've kind of already done. Then we get it to interview the user relentlessly so we have a kind of grilling session again and then we start um putting together a PRD template. So, this is available in the repo if you want to check it out. And essentially this is what it looks like. We've got some problem statements, the problem the user is facing, the solution to the problem and a set of user stories. And these user stories sort of define what this is. You know, as you you guys have probably seen things like this if you've been a developer at all. Um you know, there are cucumber is a language you can use to write these in or we just sort of um uh write them ourselves essentially.

Then we have a list of implementation decisions that were made and list of crucially testing decisions, too. So, I'm going to run this. Okay. And so, it's finished its thing. Ah! Windows, let me close the thing. Thank you. I don't know why I bought a Windows laptop. I think I just I like the challenge. Um >> [clears throat] >> So, the first thing that it's going to give me are a set of proposed modules it wants to modify.

Now, there's a deep reason why I'm thinking about this. So, this is at this stage we have an idea, we have sort of specked out the idea, we've reached a sort of understanding of what we're trying to do and then we need to start thinking about the code because at this point we need to this is not specs to code. This is not where we're ignoring the code. We actually keep the code in mind throughout the whole process.

And the way I like to do this is I like to just sort of think about a set of proposed modules to modify. We're going to return to this this idea of continually designing your system and keeping your system in mind. So, it's it's saying recommend tests for the gamification service is the only deep module with meaningful logic. These modules look right. Yeah. Looks good. And it's going to hang out a PRD. Now, for ease of setup I've got it so that it creates a set of issues locally.

So, it's just going to create essentially a PRD inside this issues directory. But, the way I usually do it and you can check this out yourself is you can go to my um essentially what I consider my work repo which is GitHub um dot com forward slash Matt Pocock forward slash course video manager up here. And in here, this is essentially a app that I create um that I use all the time to record my videos and things like this. I think I've recorded like I pulled out the stats. I think I've recorded like a thousand videos in here or something nuts.

Um and you can see here that it's got 744 closed issues. And this is essentially all of the uh PRDs and all of the implementation issues that I've put into here. So, this is how I usually like to do it. >> [clears throat] >> So, that's what I'm doing with the There we go. Yeah, I'm just going to say yes and uh and get that issue out. Let's see. It is inside here. So, we've got the problem statements.

People signing up for courses. Uh the solution, the user stories, uh 18 user stories looks nice, some

implementation decisions, level thresholds, etc. This is enough information. We've kind of clarified where we're going and what we're doing. So, that's what we do. We essentially have a grilling session and we've created an asset out of it. Now, raise your hand. Should I be reviewing this document? Raise your hand if you think I should be reviewing the documents. Yeah, I don't I don't look at these.

I don't look at these. The reason I don't look at these is because what am I testing at this point? What am I Like when I read it, what am I testing? What am I What are the failure modes I'm trying to test for? I know that LLMs are great at summarization cuz they are. They're really good at summarization. I have reached the same wavelength as the LLM, right? Using the grill me skill, we have a shared design concept.

So, if I have a shared design concept, all I'm doing is I'm just essentially checking the LLM's ability to summarize. So, I don't tend to read these. Let's have Let's have a Q&A cuz I can feel you guys are itching for it. And I think we might have like I don't know, just a 5-minute comfort break just to uh rest my voice and so you can catch up with the exercises for a minute if that's all right. So, let's have a little Q&A sesh.

Uh If I don't like Claude Code, which one do I actually like? Um uh Have you ever heard the phrase um uh democracy is the worst way to run a country apart from all the other ways? That's how I feel about Claude Code. Uh we've answered that one. Uh What's your thoughts on developers needing to very deeply understand TypeScript now that fix the TS make no mistakes exist? I don't understand the phrasing of this, but I think I understand meaning, which is that I believe that code is very important and this is kind of going to feed through the whole session and that bad code bases make bad agents. If you have a garbage code base, you're going to get garbage out of the agent that's working in that code base. We'll talk more about that in a bit.

And so, I think understanding these tools very deeply, understanding code deeply is going to make you a much much better developer and get more out of AI. Uh and that answers that question, too. Sweet. Uh Get out of there. There you are. Now that we have 1 million tokens available, do we ever actually want to take advantage of that? I've noticed that the dumb zone has become less dumb lately. Okay, great question. This goes back to our kind of initial idea on the dumb zone.

Uh I am I recorded my Claude Code course using a 200k context window and on the day that I launched the course they announced the 1 million context window. My take on this is that what Claude Code did is they essentially just did this. Wee! They shipped a lot more dumb zone to you essentially. Now, this is good for tasks where you want to retrieve things from a large context window. If you want to pass five copies of War and Peace or something to it and you want to find out all the things that uh uh I can't remember a character from War and Peace. Uh Why did I start with that?

It's good for retrieval. It's less good for coding. So, I consider that it is about 100k at the moment is the smart zone. The smart zone will get bigger and that will be a really nice improvement. So, folks, we're going to take it like a 5-minute comfort break if that's all right just for my voice and to maybe you can have a little move around or something or grab a drink. I can just notice some sleepy eyes and I want to make sure that we're awake for the next bit if that's all right. So, we'll take 5 minutes and I will see you back here then. All right?

So, we have our PRD which I'm not going to read, our kind of destination document. Let's quickly scan for any good questions before we zoom ahead. And Rediscovering the role of software engineering today's world, top three disciplines you recommend. Um Taekwondo is good, I've heard. I've no I've no idea how to answer this question. Um thank you for asking it though. Um Top three disciplines I recommend. I mean Sorry? Plumbing. Plumbing is a good one.

Yeah, yeah, yeah. I don't know if that's a discipline. I the plumbers I've hired are not usually very disciplined. Um Right. So, okay. We now have our destination, okay? Um Perfect. So, how do we actually get to our destination? How do we We have a sort of vague PRD? How do we split it so that we don't put things into the dumb zone? In other words, we have our number four, how do we split it into this kind of multi-phase plan? Well, probably what you would do at this point is you would say, "Okay, Claude, give me a multi-phase plan that gets me to this destination, right?" That sort of makes sense. This is what we've been doing before.

But I have um a sort of better way of doing it now, which is that I like creating a Kanban board out of this. Raise your hand if you don't know what a Kanban board is. Mm, cool. Okay. A Kanban board is essentially just a set of tickets that you put on the wall that have blocking relationships to each other. So, we're going to see what it kind of looks like here. This is how we've worked um as developers for a long time, really since Agile came around. And what it does, we can see it here, it has proposed that we split this setup into um five different tasks here.

We have the first one, which is the schema and the gamification service. Yeah, well, that looks pretty good. This is blocked by nothing. And we can even see here that it's a it's given it a type of AFK, too. You remember I talked about human in the loop and AFK earlier? This is an AFK task. This is something we can just pass off to an agent to do its thing. Streak tracking, okay, that looks good. Uh then wire points and streaks into lessons quiz completion. This is blocked by one and two.

Retroactive backfill. This is blocked only by one. And then this one here is blocked by all of the tasks. Cool. Hmm. Now, I consider this you could say, "Why don't we just make this sort of generation of the issues, why don't we just hand that over to the AI? Why do I need to be involved here, right?" Cuz it's given us quite a good selection of tools here. Why do I need to review this and sort of figure out what's next?

Now, my take here is that this is really cheap to do, like very quick to do once I've done the PR, and I can immediately see some issues here. There's a really, really important technique when you're kind of figuring out what the shape of this journey should look like. And it sort of comes to this very classic idea, uh which comes from the Pragmatic Programmer called traceable bullets or vertical slices. And traceable bullets really transformed the way I think about actually getting AI to pick its own tasks.

Systems have layers, right? There are layers in your system. These might be different deployable units. You might have a database that lives somewhere. You might have an API that lives maybe close to the database but in a separate bit. You might have a front end that lives somewhere totally different like a CDN. Or within these deployable units, you might have different layers within those. In for instance, the code base that we're working

in, we have a ton of different services. Service. We have a quiz service, a team service, a user service, coupon service, core service.

And these services have dependencies on each other. So, they're kind of like individual layers. Well, what I noticed is that AI loves to code horizontally. So, it loves to code layer by layer. So, in other words, in phase one, it will do all of the database stuff, all of the schema, all of the you know, all the stuff related to that unit. Then it will go into phase two and do all of the API stuff. Then it will add the front end on top of that.

Does Can anyone tell me what's wrong with that picture? Why is that not a good thing to do? Raise your hand if you have an answer. Yeah. >> have that whole feedback loop. Exactly. You don't get feedback on your work until you've really started or completed phase three. So, what you really need to do is you you're not until you get to phase three, you're not actually testing that all the layers work together. You haven't got an integrated system that you can test against.

And so, instead you need to think about vertical layers. You need to think about thin slices of functionality that cross all of the layers that you need to. And this is a much better way to work, much better way for the AI to work, too, because it means at the end of phase one or during phase one it can get feedback on its entire flow. So, what this means to me is inside the PRD to issues skill up here, I have got break a PRD into independently grabbable issues using vertical slices traceable bullets written as local markdown files.

[snorts] We first locate the PRD. Uh again, explore the code base if this is a fresh session. We draft vertical slices. So, we break the PRD into traceable issues. A traceable bullet, by the way, is uh essentially when you're like an anti-aircraft gunner. It's quite a violent idea, actually. Uh and you're looking up in the sky and it's night. If you're just shooting normal bullets, you have no idea what you're firing at, right? You could just be you know, you you see the plane but you don't see where your bullets are going.

Traceable bullets is they attach a tiny bit of phosphorescence or phosphor or something to make it glow as it goes. So, this means that every sixth bullet or something you actually see a line in the sky. So, you have feedback on where you're aiming. So, this is what this is the idea here is that we increase our level of feedback and we get near instant feedback on what we're building. Cuz without that the AI is kind of coding blind until it reaches the later phases.

We got some vertical slice rules. We quiz the user. And then we create the issue files. So, what I see here is that even though I've I've told it to do vertical slices, it's proposing to create the gamification service first on its own. That's just one slice there. And that to me feels like a horizontal slice. What I want to see in the first vertical slice especially is I want to see the schema changes or some schema changes. I want to see some new service being created and I want a minimal representation of that on the front end. So, I want it to go through the vertical slices, not just the horizontal. Does that make sense?

Okay. So, I'm going to give the AI a rollicking. Uh bad boy. No, I'm not. I'm not going to waste tokens just being just naming. Um So, the first slice is too horizontal. I'll just start with that and see if it picks it up. Does that make

sense as a concept? And I think having that um what I really like about going back to those old books is that we're really trying to in this day and age like get uh verbalize best software practices in English.

And these books, 20-year-old books, have already done that. And it's an absolute gold mine if you want to throw that into prompts. But even with that, it's not going to um not going to do a perfect job each time. So, award points for lesson completion visible on dashboard. Yes, that's a beautiful vertical slice because it's definitely a big chunk of stuff. It's doing a lot of stories there, but we're going to see something visible at the end and the AI will then just be able to add to that. You see why that's preferable to the first one. Cool.

Uh looks great. So, we're getting closer now. Anyone following at home as well, you know, not at home but you get the idea. Um will hopefully see the same thing, too, and start developing the same instincts. Let's open up for questions just while I'm still creating these GitHub issues. Uh ba ba ba ba Oh, not GitHub issues. Uh local issues. When will I stop using Windows? Never. What is your Okay, we'll get to that later. How does AI um decide when to stop grilling? Cuz AI can ask incessantly, can we have a smarter way to decide the stop point? Yeah, it does tend to really um those grilling sessions can be super intense. And the thing about these skills is you can tune them if you want to. If you feel like the AI is just absolutely hammering you, hammering you, hammering you, then you can just tell it to just pull back a little bit or get it to do, you know, stop points and that kind of thing. So, if that's a failure mode that you run into a lot, then you just, you know, change the skill.

Uh do I still use uh be extremely concise, sacrifice grammar for the sake of concision? Um there was a tip that I gave folks um 5 months ago, which is that to basically increase the readability of your plans. So, when you're using plan mode, then you can put it in your Claude.md and you can say, "Okay, yeah, approve that." Let's open up Claude.md. Uh do I have a Claude.md? Maybe I don't. I I really don't use Claude.md very much. I'm just going to put a dummy inside here.

Um when No. When talking to me, uh sacrifice grammar for the sake of concision. And this um prompt was uh really useful to me when I was reading the plans because it meant that the plans would come out and they would be very concise, really nice, easy to read, often very concise. But I've since dropped this idea in preference to a grilling session because what I noticed with it just I didn't want to read the plans. I wanted to get on the same wavelength as the LLM. I wanted it to ask aggressive questions to me. And when I stopped reading the plans, I stopped needing them to be concise.

So, I think of the plans really in the destination document as uh the end state. And I don't need that end state to be concise. Hopefully that answers your question. Uh What do I think will be the outcome of the Mexican standoff of future roles of PMs and other roles converging? Uh I've no idea. I'm not a pundit. I've no idea. Uh okay. So, we should uh after a couple of approvals, uh end up with a set of issues.

Now, these issues that we're creating, they're designed to be independently grabbable, which means that this Kanban board ends up looking kind of like this. Where you have essentially a set of tickets with a whole load of independent relationships. So, this one needs to be done before this one. This one needs to be done before this one. And this one, let's say we got another one over here. This one needs to be done before this one. This means

that you can start to parallelize.

You can start to get agents working at the same time on these tasks. Because yeah, this one needs to be done first. And then these two can be grabbed at the same time by independent agents. Raise your hand if you've done any kind of parallelization work with agents. Okay, cool. So, this allows you um to turn those plans into to optimally kind of like into a directed acyclic graphs essentially, where you just are able to um essentially have three phases here.

Where you have phase one. Uh let me grab move that. Uh above this line here, you do this one. Then phase two, you do the two below it. And then phase three, you do this third one and add it onto that. And when you think about there could be This could This is a relatively simple plan, but you could have many different plans operating all at once. It means that you can do really nice parallelization. And we'll talk more about that in a bit. But that's why I prefer a Kanban board set up like this to a sequential plan. Because a sequential plan can really only be picked up by one agent.

So, this Where did it go? Over here. Yeah, this plan here This is really only one loop, right? Only one agent can work on these because we have numbered phases and they're not parallelizable. Does that make sense? Cool. So, we've got our issues. Ah, come on. Stop asking me for I know it's creating them on GitHub. I really don't want that. Oh, no. You fool. Create them in issues instead.

No. That's not precise enough. Uh you fool. Create them in local markdown files instead, referencing the local version. Sorry about this. So, once we get to this point, we [clears throat] have a bunch of issues locally that we can start um looping over and implementing. And it's at this point that the human leaves the loop.

So, so far Let me pull up a a proper overview of this kind of flow that we're exploring here. So far we have taken an idea. I'll zoom this in a bit for the folks at the back. And we've grilled ourselves about the idea. We can skip over research and prototype, but we turn that into a PRD, into a destination document. We then turn that PRD into a Kanban board. And all of those steps are human reviewed.

And now the implementation stage, we step back. And we let an agent um work through that Kanban board or multiple agents work through the Kanban board. Now, what this means is that yeah, we spent a lot of time planning here, but it means that we've queued up a lot of work for the agent. We can think of this as kind of like the day shift and the night shift. This is the day shift for the human, right? Planning everything, getting all the all the stuff ready. And then once we kick it over to the night shift, the AI can just work AFK. But what does that look like?

Well, so I'm just going to Oh, yeah. Just allow it. It's perfect. So, this looks like if we head to the next exercise, which is uh in fact, the last exercise here, running your AFK agent. Now, I've called this uh Ralph really cuz it is a it is essentially a Ralph loop. And this prompt here, I want to walk through this really closely. The first thing it's doing here is we're essentially going to run Claude and we're going to basically try to encourage it to work um completely AFK.

I'll show you what the sort of script for this looks like in a minute. But you say, "Okay, local issue files from issues are provided at the start of context." The way we do that is if you look inside once.sh here inside the repo, we have uh it's essentially just a bash script, where we grab all of the issues, um [clears throat] which are inside markdown files, and we cat them into a local variable. So, that issues variable contains all of the issues that are in our entire backlog.

Then we grab the last five commits. I'll explain why in a minute. And then we grab the prompt and we just run Claude code with permission mode accept edits. And then just essentially just pass it all of the information. This is what the implementer looks like. So, that's what a very very simple version of this sort of loop looks like. And of course, this is not a loop. This is just running it once. The loop is in the AFK version up here, which is uh a fair bit more complicated.

And the crucial part here is we're running it in Docker sandbox as well. So, I I don't want you to install Docker on your laptops because we're just going to be like, "You need to download a special image and we're going to tank the conference Wi-Fi if we do that." So, I'm I am going to demo this to you, but you um won't need to run this yourself, but I'll talk through this in a minute. But essentially, this once loop here, and ba ba ba ba boom.

We're just essentially running one version of the thing that we're going to loop again and again and again. So, this is kind of like the human in the loop version. And this is essential. Running this again and again is essential because you're going to see what the agent does and see how it ends up working. And any tuning that you need to add to the prompt, then you can do that. Let's go to the prompt. Um So, local issue files are being passed in.

You're going to work on the AFK issues only. That makes sense. If all AFK tasks are complete, output this no more tasks thing. And then the next thing, pick the next task. So, what we're doing here is we're essentially running a backlog or curating a backlog that our AFK agent is going to pick up. That's the purpose of all of these um setups in the beginning. In this uh all the way to this Kanban board here, we're just essentially creating a backlog of tasks for the night shift to pick up.

And the night shift, this sort of Ralph prompt here, it's got its own idea about what a good task looks like to next pick up. I'm I did talk about parallelization. I will show you this later, but this is essentially a sequential loop here. We're just going to run one coding agent at a time. This is a good way to just sort of um get your feet wet essentially. So, it's prioritizing critical bug fixes, development infrastructure, then trace bullets, then polishing quick wins and refactors.

And then we just have a very simple kind of instruction on how to complete the task. So, we explore the repo. Use TDD to complete the task. I'll get to that later. And we then run some feedback loops. So, let's let's just try this and let's just see what happens. So, good. It's created the issue files. We should be good to go. I'm going to cancel out of this. I'll clear and I'm going to run uh Where is it? Ralph once.sh. And you can feel free if you're following along to do the same thing.

So, we can see it's just running Claude inside here with the prompt and with all of the issues that have been

passed in. And while it's doing its thing, you probably have some questions about this setup and about the decisions that I've made to essentially delegate all of my coding to AI, right? So, let's let's do a quick Q&A while it's getting its feet under it. Uh okay. Ba ba ba ba ba. I'm going to just remove those.

How do you retain negative decisions, things that you decided against, and rationales when persisting the results from the grill me session? Uh great question. There's a very simple answer, which is the in the PRD uh write a PRD section, there is a stuff at the bottom, a section of the things that are out of scope. So, the things we're not going to tackle in this PRD, which is very important for giving a definition of done. Feel free to ping on the Slido if you've got any more questions.

Uh what's my front end workflow? Okay, it's a great question. I'm going to I'm going to answer that in a minute, I think. How to deal with agents producing more code that we can review? How to properly parallelize and use multiple agents separate way. Okay, that's That's two questions there. Um Raise your hand if you feel like you're doing more code review now than you used to. Yeah, definitely. Um I don't think there's a way to avoid this.

If we delegate all of our coding to agents, you notice that the implementation here is really the only AFK bit. We then also need to QA the work and code review the work, right? And if we are running these loops where it's essentially going to implement four issues in one, it's hard to pair that with the dictum that you should keep pull requests small and self-contained, right? Like small self-contained pull requests means you're needing to do fewer loops or shorter loops or something.

Or maybe you do like a big stack of PRs, but that seems horrible as well. That's still just more separated code to review. I don't honestly know what the answer to this yet. I think we just need to be ready to be doing more code review, essentially. Which is not fun. That's not fun thing to say. That's not like I don't know. I don't feel good saying that, but I do think it's probably the the way things are going.

It's a great question. Uh Can we grab a couple of questions from the room as well? Let's not We won't do the mic, but uh raise your hand if you've got a question for me immediately. Yeah. So, the approach is very linear from an idea to uh QA code review. Of course, the real world is a lot more messy. So, you have all these ideas that are in parallel and nobody has the full picture. And uh while you're working on something, something else comes in as a bug. Yeah. How do you deal with the messiness? How do you tighten that feedback loop? Great question. So, the question was if this all looks great if you're a solo developer, but actually how do you implement this in a team? How do you gather team feedback on this?

And my answer to that is that if you have an idea up there and essentially the sort of journey from the idea to the destination is something you need to figure out with the team, right? So, all of this stuff up here, this is kind of like team stuff, you know what I mean? This So, if you have an idea and you do a grilling session on it and you have a question that you don't know how to answer, then you need to loop in your team as we described before. Then you might need to go, "Okay, like we just need to build a prototype of this. We need to actually hash this out. We need something that the domain experts can fiddle with."

Or okay, we might need to integrate a a third-party library into this. We might need to do some research. We might need to actually kind of like um ping this back and forth and find a third-party service that we can get the most out of. We might need to go back with the information that we gathered there to the idea phase. So, all the way up to the sort of PRD in the journey, that's something you need to involve your team with. That's something where these assets are going to be shared over and you're going to have requests for comments on them and that that loop is going to just keep grinding and grinding until you figure out where you're going.

Once you figure out where you're going, then you can start doing the Kanban board implementation. But this is essentially super arguable and the you'll be bouncing back and forth between the phases. Does that make sense? Yeah. Would you not need a PRD for your prototype? Say again, sorry. Would you not want to have a PRD for your prototype? The question was, do you want to go through this whole session just to sort of create a prototype? You don't need a PRD for your prototype as well. Let's just quickly talk about prototypes for a second.

Um there was a question about how do you make this work for front end? Like how do you cuz front end is like really sensitive to human eyes. You need human eyes looking at the front end all the time to make sure that it looks good. AI doesn't really have any eyes. It can look at code, but it front end is multimodal. And so my experiences with trying to plug AI into um let's say agent browser or Playwright MCP to give it You can give it tools to allow it to look through a front end and sort of look at images, but in my experience the um it's not very good at that yet and it can't create a nice front end in a mature code base. It can sort of spit one out. But what it can do is you say, "Okay, uh I want some ideas on how uh this front end might look. Give me three prototypes um that I can click between in a throwaway uh throwaway route that I can decide which one looks best." And you take the asset of that prototype and you then feed it back into the grilling session or you get feedback on it, blah blah blah blah blah.

Answer your question kind of thing? The prototype is just, you know, it's messy. It's supposed to give you feedback earlier on the process. So, that's a great way of working with front end code, great way of looking at software architecture in general. Let's go one more question here. Yes. >> [clears throat] >> In your system, how do you integrate respecting an architecture and design with API contracts and fitting with your larger system? Uh security constraints, all kinds of constraints like that.

Yeah. There's a lot in that question. The question was, how do you conform with existing architecture? How do you do um how do you make it conform to the code standards like of your code base or Yeah, the architecture design APIs, Yeah. security rules that constrain your design. Yeah. I'm going to answer that in a bit. That's okay. So, hopefully we have started to get some stuff cook cooking. Uh it's just pinging on the explore phase here.

Hmm, tempted to just start running it AFK. Maybe I will, maybe I won't. Um What it's essentially doing is it's exploring the repo. It's going to then start implementing based on what we wanted. Let's actually have one more question just while it's running. Yeah. Why not AI QA everything Yeah. So, the question was, why do you not get AI to QA? AI to QA.

I just got uh jargon overload for a second. Um why do you not get AI to uh test its own code? Now, of course, you absolutely can. And I think while it's doing while it's cooking here, okay, it's got a clear picture of the code base. It's assessing the issues. It's doing issue 02 as the next task. I'm again going to show you that in a bit, I think. The sort of uh cuz you definitely should do an automated review step as part of implementation.

So, you have your implementation, you should then, because tokens are pretty cheap and AI is actually really good at reviewing stuff, you should get it to review its own code before you then QA it. I found that that catches a ton of different bugs and the way that works is I will just do a little diagram is if you have, let's say, an implementation that sort of like used up a bunch of tokens in the smart zone, if you get it to sort of try to do its reviewing, it's going to be doing the reviewing in the dumb zone.

And so, the reviewer will be dumber than the thing that actually implemented it. If we imagine this is the uh let's be consistent. That's the review. That's the implementation. Whereas if you clear the context, then you're essentially going to be able to just review in the smart zone, which is where you want to be. Let's see how our implementation is doing. Okay, good. It's generating a migration. That looks pretty nice. We're getting some code spitting out.

And while I'm sort of like Aha, here we go. TDD. Let's talk about TDD and then I think we'll have a little another little break. TDD I found is absolutely essential for getting the most out of agents. Uh raise your hand if uh you know what TDD is. Cool. Okay. TDD is test-driven development. What it's essentially doing is it's doing a something called red green refactor. And if you look in the code base, you'll be able to find a um a skill which really describes how to do red green refactor and teaches the AI how to do it.

So, what it's doing is it's writing a failing test first. So, it's saying, "Okay, I've broken down the idea of what I'm doing and I'm just going to write a single test that fails and then I need to make the implementation pass." I have found that first of all, this adds tests to the code base and these this tends to add good tests to the code base. And so, we've got this kind of gamification service. It looks like it's using some existing stuff to create a test database. Test fails because the module doesn't exist yet. Okay, we've confirmed red. And then it goes and hopefully runs it and it passes.

I found that uh raise your hand if you've ever had AI write bad tests. Yeah. It tends to try to cheat at the tests because it's sort of doing it in layers. It will do the entire implementation and then it will do the entire test layer just below it. Uh I'm just going to say yes, you're allowed to use NPX V test. And using this technique, it generally is a lot harder to cheat because it's sort of instrumenting the code before it's then writing the code. So, I find that TDD is so so good for places where you can pull it off. In fact, it's so good that I sort of warped my whole uh technique around getting TDD to work better.

I can see some dripping eyes. It is so hot in here. You can't imagine how hot it is up here. Let's take another 5-minute comfort break. Let's come back at quarter to, I think. Have a nice generous one. And we'll be back in about 6 7 minutes and I'll talk about how uh I think about modules, think about constructing a code base to make this possible. I've just been sort of fiddling with the AI here and we have ended up with some with a

commit.

So, we have something to test. Issue number two is complete. Here's what was done. This is kind of what it looks like when a Ralph loop completes is you end up with a little summary. Um and we have now something we can QA. Because we did the feedback loops because we did the trace bullets because we were uh said, "Okay, give us something reviewable at the end of this." We can immediately go and QA it. Now, there's nothing uh less exciting than watching someone else QA something.

But, hopefully we can have a little play. Let's just check that it uh works at all. In fact, before I go there, I just want to sort of work through what just happened. Which is we see that it's created some stuff on the dashboard. And it then ran the feedback loops. So, it then ran the tests and the types. Now, TDD is obviously really important. And it's really important because these feedback loops are essential to AI, essential to get AI to produce anything reasonable.

Because without this, AI is totally coding blind, right? You have to have to um If if your code base doesn't have feedback loops, you're never ever ever going to get decent AI decent output out of AI. And often what you'll find is that the quality of your feedback loops influences how good your AI can code, essentially. That is the ceiling. So, if you're getting bad outputs from your AI, you often need to increase the quality of your feedback loops.

We'll talk about how to do that in a minute. Now, so it ran NPM run test, NPM run type check. It got one type error, and it needed to fix it with a nice bit of TypeScript magic. Very good. Yeah, type of level threshold number. Okay. Uh you see why I stopped teaching TypeScript cuz just AI knows everything now. Um So, and it ran the tests, and it passed, and it's looking good. So, we now end up with 284 tests in this repo. Pretty good.

I I do find uh front end really hard to test here. We're essentially just testing the service. So, we've created a gamification service, if we look up here. And then we have a test for that service. You can see that the service and the test itself. Now, if I was doing code review here, I would then go to I would first go to review the tests, make sure the tests were testing reasonable things, and then go and kind of review the code itself just to make sure that it's it's not doing anything too crazy, right?

The essential thing is I need to actually um look at the dashboard. I'm going to log in as a student. Oh, if it'll let me. Maybe it won't let me. Come on, son. There we go. Let's log in as Emma Wilson. Head into courses. Uh let's say I've got an introduction to TypeScript. Continue learning. Uh yes, I completed this lesson. And something went wrong. I imagine it's because I don't have Uh SQLite error. I don't have the right table. So, I need a table point events.

Point events is a strange table name. I'm not sure quite what it was thinking there. Uh let's suspend. Let's run uh NPM DB migrate. Push, I think. I can't remember which one it was. But, you kind of get the idea, right? I I'm not going to subject you to uh watching me do QA because it's so dull. Um but at this point, I would essentially go back in. I would um Let me open the project back up.

Uh and I would This This is a crucial moment, um and it's so important to um QA it manually here because QA Oh, dear, oh dear. What's going wrong? There we go. QA is how I then um impose my uh opinions back onto the code base, how I impose my taste. What you'll often find is that um there are teams out there who are trying to automate everything, like every part of this process. And they will tend to uh if you try to like automate the sort of creation of the idea, automate uh the QA, automate the research, automate the prototype, you end up with uh apps that I feel just lack taste and are bad.

Maybe they just don't work, or they they don't even work as intended, or there's just no You need a human touch when you're building this stuff because without that, you just end up with slop. And we are not producing slop here. We're trying to produce high-quality stuff, and so that's what the QA is for. Mhm. So, I'm going to do two things in this final section. Which is I'm going to first tell you how to There's probably a question in your mind here, which is let's say I have a code base that I'm working on.

And it's a bad code base. It's a code base that's like really complicated, uh that AI just never does good work in, and maybe actually most humans that go into that code base don't do good work. How what How do I improve that code base? And the second thing is I'll show you my setup for parallelization. So, let's go with um bad code first. Now, where is it? Where's the diagram? Here it is. In his book, um The Philosophy of Software Design, John Ousterhout talks about the ideal type of module.

And let's imagine that you have a code base that looks like this. Each of these uh blocks here are individual files. And these files export things from them. You know, they have um things that you pull from the files that you then use in other things. And so, you might have these weird dependencies where this file over here might rely on this file, or might rely on that file, for instance. Now, if these files are small and they don't kind of ex- like export many things, then John Ousterhout would call these shallow modules, essentially. Where they're not very um They kind of look like uh this, if I No, actually no. I can't can't make a good diagram of it.

They're essentially lots and lots of small chunks. Now, this is hard for the AI to navigate cuz it doesn't really understand the dependencies between everything. It can't work out where everything is. You know, it has to sort of manually track through the entire graph and go, "Okay, this relies on this. This one relies on this one. This one relies on this one." And it's then also hard to test this, as well, because where do you draw your test boundaries here?

Do you test each module individually? Like just literally draw a test boundary No, don't do that. Around this one? And then maybe another test boundary around the next one, and then the next one? Or should you sort of do big groups of it? Should you say, "Okay, we're going to test all of these related modules together, and just sort of, you know, hope and pray that they work." Now, >> [sighs] >> this means that if I think that bad tests mostly look like that, where the AI essentially tries to sort of wrap every tiny function in its own test boundary, and then just sort of test that those individually work. But, what that does is it means that when, let's say, this module over here calls those two, so it depends on both of these, then this module might miss order the functions, or there might be sort of stuff inside that poor module that's worth testing on its own. And if you then wrap this in a test boundary, what do you do? Do you mock the other two modules? How does that work?

So, actually figuring out how to um build a code base that is easy to test is essential here. Because if our code base is easy to test, then our code our feedback loops are going to be better, and the AI is going to do better work in our code base. Does that make sense? So, what does a good code base looks like look like? Well, not like that. It looks like this. Where you have what John Ousterhout calls deep modules.

Modules that have a little interface on there that expose a small, simple interface that have a lot of functionality inside them. Now, what this means is that these are easy to test cuz you just Let's say that there's a dependency between this one and this one. My arrow working? Yeah, there we go. Then, what you do is you just wrap a big test boundary around that one module, around this one up here, and you're going to catch a lot of good stuff.

Because there's lots of functionality that you're testing, and really the caller, the person calling the module, is going to have a simple interface to work from. So, it's not not too tricky. That makes sense? Deep modules versus shallow modules. This is good. This shallow version is bad. And what I find is that unaided um or if you don't uh if you don't watch AI carefully, it's going to produce a code base that looks like this. So, you need to be really, really careful when you're directing it.

And that's why, too, is that if we look inside the PRD, uh where is the PRD gone? It's inside the issues. It's inside the gamification system. Uh not found. Of course, it's not. Here it is. Then I have uh inside here data model the modules. So, it's specifically saying, "Okay, this gamification service is a new deep module, which we're going to test around. It's going to have this particular interface.

And it's going to have um Okay, we're modifying the progress service, too. We're modifying the lesson route. We're modifying the dashboard route, etc. So, it's I'm being really specific about the modules that I'm editing, and I'm making sure that I keep that module map in my mind at all times, throughout the planning, and then throughout the implementation. Does that make sense? Very, very useful. It's useful for one other reason, too. Not only does it make your app more testable, but you get to do a little mental trick.

And I'm going to refill my water while you wait for what that is. Uh let me Let me get a question from you guys. So, raise your hands if you feel like Uh if you feel like you're working harder than ever before with AI. Yeah. Uh raise your hands if you feel like you know your code base less well than you used to. Yeah.

This is a real thing. Um because we're moving fast, because we're delegating more things, we end up losing a sense of our code base. And if we lose the sense of our code base, we're not going to be able to improve it, and we're essentially delegating the shape of it to AI. I [snorts] don't think that's good. But then how do we how do we make it so that we can move fast while still keeping enough space in our brains?

I think that this is a way to do it. Because what you're doing here is not only are you thinking about creating big shapes in your code base, big services. What I think you should do is design the interface for these modules, but then delegate the implementation. In other words, these modules can become like gray boxes, where you just need to know the shape of them, you need to know what they do, and it's sort of how they behave, but you can delegate the implementation of those modules. I found this is really nice. I don't necessarily need to code review

everything inside that module. I don't necessarily need to know everything of what it's doing. I just need to know that it behaves a certain way under certain conditions, and that it does its thing. So, it's kind of like okay, I've got a big overview of my code base, and I understand kind of the shapes inside it, understand what the interfaces all do, but I can delegate what's inside.

I found that has been a really nice way to retain my sense of the code base while preserving my sanity. Make sense? And so, you might ask, how do I take a code base that looks like this and then turn it into a code base that looks like this? How do I deepen the modules? Well, we have Hopefully, it's in here. Pretty sure it is. We have a skill. And that skill is called improve code base architecture.

Nice and direct. Uh let's run it. What this skill is going to do is it's essentially just going to do it a scan of our code base and looking for what's available here. And feel free to run this yourself if you're um uh running the exercises. And it's exploring the architecture, exploring um essentially how to work within this code base, and it's going to attempt to uh find places to deepen the modules. Pretty simple. One really cool um thing that it found here is part of my uh part of my course video manager app is a video editor. A video editor built in the browser, which is really hardcore.

Uh it's a decent bit of engineering. And I wanted a way that I could wrap the entire front end all the way to the back end in like a single big module, so that I could test the fact that I press something on the front end and it goes all the way to the back end. And so, I found a way essentially by using a kind of discriminated union between the two types here by sort of I was able to use this uh skill to essentially have a huge great big module that just tested from the outside, it was testable from the outside, this video editor infrastructure. And it meant that AI could see the entire flow, could act on the entire flow, and test on the entire flow. And honestly, it was just night and day in terms of the uh ability of AI to actually make changes, cuz AI working on a video editor is pretty brutal if you don't give it good tests. So, that is Honestly, I If you take one thing away from today, just try running this skill on your repo and see what happens.

Let's go to Slido. Let's ask a check a couple of questions as well this is running. So, let's see. Have you tried Claude's auto mode with Claude enable auto mode? That way you can avoid many of the obvious permission checks. We'll talk about permission checks in a second. Do I keep the markdown plans and issues for later reference? Okay. This is a great question. So, let's say that you uh have a great idea, you turn it into a PRD, raise and you then implement that PRD, and the PRD is essentially done.

Raise your hand if you keep that information in the repo, so you turn it into a markdown file. Raise your hand if you want to keep that around. Cool. Okay. And raise your hand if you if you don't want to keep it around. If you want to get rid of it as soon as possible. Yeah, this is I think an a question that doesn't have a clear answer. What I'm really scared of with any documentation decision is that let's say that we have a PRD for this gamification system, we keep it in the repo.

We go on, go on, go on. Let's say a month later, we want some edits to the gamification system. And we go in with Claude, and it finds this old PRD and says, yes, I found the original documentation for the PRD system.

Well, it turns out that the actual code has changed so much from the original PRD that it's almost unrecognizable. The names of things have changed, the um file structure has changed, even the requirements may have changed. We might have actually tested it with users. This is doc rot, where the documentation for something is rotting away in your repo and influencing Claude badly. Or Claude, agents badly.

So, I tend to not keep it around. I tend to get rid of it. And for me, because my setup uses GitHub issues, I just mark it as closed. It can fetch it if it wants to, but it's got a visual indicator that it's done. So, I tend to prefer ditching these. Thoughts on the BEADS framework from Steve. Uh I've not tested it, but it seems like sort of um another way to manage Kanban boards and issues. Seems uh very good, but I've not tried it.

Um >> [clears throat] >> Uh let me just quickly check the uh setup here. Let's take a couple of questions from the room. Anybody got any questions at this point about anything that we've covered so far, especially this last bit? Yes. I thought it was interesting your answer about like the markdown files that you delete because they create like doc rot. How about migrations? Like with migration files, would you also squash them after that? Like database migrations? Yeah.

I don't know. I hope that answers your question. I'm so sorry. No, no. I think database migrations are a different thing because you have a sort of running record of exactly what changed, and it's more deterministic. And I think Yeah, it's an interesting analogy. I'm not sure. Let's talk about it afterwards. That's a good way of saying I've no idea. Yeah. Yeah. So, you mentioned that you don't delete the PRD. You mentioned you don't review the PRD once it's done.

Sorry, guys. Um I'm just trying to listen to this guy's question. Have you considered uh using a deep think like ChatGPT or something to tell it, "Look at this PRD and tell me if it It takes about an hour. Yeah, the question The question here is um should I um in the sort of early planning stage be trying to optimize the plan? This is something I actually see a lot of people doing, and it's a really good um idea. So, when you Let's go back to the phases.

So, let's say that you have all of these phases here. And you uh you get to the point where you've sort of figured out everything with the LLM, you understand where you're going, you've created this sort of uh journey destination documents here. How do you then uh Like should you then try to optimize and optimize and optimize that PRD until it's the perfect PRD you can possibly imagine? I don't think there's a lot of value in that.

Because I think the journey is really just sort of a hint of where you want to go, and the place that you need to be putting the work is in QA. And you can sort of do that AFK, I suppose, but in my experience, you're not going to get a lot of juice out of it. Like it's the The thing that really matters is getting alignment with the AI, which is you do in the grilling session initially.

Let's have one more question. Anyone got any more? Yeah. How do you get in in your workflow to get it to code the way you want it to code it so by the time you get to code review, it's at least familiar, it uses the libraries you wanted to use, Yeah. Um we had this question before, actually, which was like uh how do you uh enforce

your coding standards on the agents, essentially? How do you get it to code how you want it to code?

Now, there's essentially two different ways of doing it. Um you've got I don't know. Come on. Push. And you've got pull. What do I mean mean by push and pull? Um Push is where you push instructions to the LLM. So, you say, okay, if you put something in Claude.md, uh talk like a pirate, that instruction is always going to be sent to the agent, right? So, that is a push, actually. You're pushing tokens to it.

Pull is where you give the agent an opportunity to pull more information. And that's for instance like skills. So, a skill is something that can sit in the repo, and it has a little description header that says, okay, agent, you may pull this when you want to. My thinking, my current thinking about code review and about coding standards looks like this. When you have an implementer, What's going on? There we go.

Implementer. I'm going to make this less red in a second. Um then you want the coding standards to be available via pull. If it has a question, you want it to be able to sort of answer it. But if you then have an automated reviewer afterwards, then you want it to push. You want to push that information to the reviewer. You want to say, "These are our coding standards. Um make sure that this code um follows them."

So if you have skills for instance, then you want to push that stuff to the reviewer so the reviewer has both the code that's written and the coding standards to compare to. Hopefully that answers your question. I can show you an automated version of this as well actually. Um Yeah, let's do that now just while it's fresh in my mind. I recently um spent uh maybe a week or so uh building this thing called Sandcastle. And Sandcastle is a I was sort of unhappy with the options out there for um running agents AFK.

And what this does is it's essentially a TypeScript library for running these loops. So you have uh a run function that creates a work tree, um sandboxes it in a Docker container, and then allows you to run a prompt inside that. And in that work tree then, it's just a Git branch and you have that code and you can then merge it later. If I open up um there are some really really nice ways of viewing this and it essentially allows you to run these kind of automated loops and allows you to parallelize across multiple different agents really simply.

So I'll go into my Sandcastle file, go into main.ts here. And let's just walk through this. So this is kind of like I showed you um a sort of version of the Ralph loop earlier. This is where we take it from sequential into parallel. We have here first of all a planner that takes in it's has a plan prompt here that looks at the backlog and chooses a certain number of issues to work on in parallel. Remember I showed you that Kanban board where it had all the blocking relationships? It works out all the phases. So this one will say okay, uh let's say we have uh you can ignore all this glue code here. This is essentially just a set of issues, GitHub issues with a title and with a a branch for you to work on.

And then for each issue, we create a sandbox and then we run an implementer in that sandbox passing in the issue number, issue title, and the branch. This is like the loop that we ran just before. Then if it created some commits, we then review those commits. This is essentially the loop. What do we do with those commits? We pass those into a merger agent. Which takes in a merge prompt, takes in the branches that were created, takes in

the issues, and it just merges them in.

If there are any issues with the merge, you know, with the types and tests and that kind of thing, it solves them. And this has been my uh flow for quite a while now for working on most projects. It works super super well. And uh yeah, I recommend you check out Sandcastle if you want to sort of learn more. And to answer your question properly is that in the reviewer uh I would push the coding standards.

In the implementer, I would allow it to pull. And I'm actually using uh Sonnet for implementation and Opus for um reviewing cuz I consider reviewing sort of I need I need the smarts then. Any question Actually, let me uh before we do more questions, let's go back here. Okay, where are we at? Okay. We sort of zooming everywhere in this uh talk because I'm kind of having to run things in parallel. So let's go back to the improve code base architecture. It has finally finished running and it's found a bunch of architectural improvement candidates.

So it's got essentially a cluster of different modules that are all kind of related that could probably be tested as a unit. Got number one, the quiz scoring service. There's some reordering logic extraction as well. It has arguments for why they're coupled and it has a dependency category as well. So local substitutable in SQL light within memory test DB. Quiz scoring service just currently has zero tests. This is the biggest gap. So this is what it looks like when we come back of uh improve code base architecture.

Okay. So we have nominally kind of 17 minutes left. I don't know about you guys, but I'm knackered. >> [laughter] >> Um I want to >> [clears throat] >> Let me let me kind of sum up for you. Cuz I think we're sort of reaching the end of our stamina. I'm going to be available for the full time if you want to um come and ask me questions. Um I might do one more check of the slide over, but let's kind of sum up where we've got to.

So this is essentially the flow. Where throughout this whole process, we're bearing in mind the shape of our code base. This is not a spec to code compiler. This is not an AI that's sort of just like churning out code. We are being very intentional with the kind of modules and the shape of the code base that we want. We are making sure that we are as aligned as possible by using the grilling session, by really hammering out our idea. We're not over indexing into the PRD, we're not trying to read every part of it. We're not thinking too much about it even. We're then just turning that into a set of parallelizable issues which can be worked on by agents in parallel.

We implement it and we QA and code review the hell out of it and then keep going back to that implementation. One thing I didn't really mention is that in the QA phase what the QA phase is for is creating more issues for that Kanban board. So while it's implementing even, you can be QAing the stuff and going back, adding more issues. And the Kanban board just allows you to add blocking issues kind of um sort of infinitely really.

And then once that's all done, once you've got code that you're happy with, once you've got work that you're happy with, then you can share it with your team and you can get a full review. So this is kind of like once you get here, this is kind of one developer or maybe a couple of developers sort of um managing this and then it's kind of up to you to figure out how to merge it back in.

>> [sighs] >> Of course all of this can be customized by you. This is just something that I have found works. I'm not trying to like sell you on a kind of approach here. What I recommend if you take one thing away from this session is that you should head back, you should head to Amazon and just buy a ton of those old books because I mean, I just found it so enlightening reading them. Uh you know, pre-AI writing is always like a a really fun to read anyway.

And I just on every single page I found that there was something useful and something interesting to to read. So thank you so much. Thank you for putting up with the heat. Um hopefully your body temperatures will reset soon. Uh thank you very much. >> [applause] [music]