Agentic coding at Airbnb

35 MIN · YOUTUBE · HTTPS://YOUTU.BE/O6OYC9-X9IW
https://youtu.be/o6oYc9-X9Iw

SUMMARY

Airbnb engineers Szczepan and Mike discuss the evolution of agentic coding within the company, highlighting its rapid adoption and the transformative impact of AI tools on developer productivity. They emphasize the importance of integrating AI seamlessly into existing workflows and the necessity of human oversight in the coding process, while also sharing insights on how to effectively implement these technologies.

- *Agentic coding allows developers to steer multiple AI sessions to produce code changes, enhancing productivity beyond traditional methods.*
- *The adoption of agentic coding tools at Airbnb has skyrocketed, with 64% of PRs now utilizing these technologies, surpassing initial predictions.*
- *A holistic approach to measuring developer productivity includes sentiment analysis, tool usage, and objective metrics like PR velocity.*
- *Engineers are encouraged to adopt agentic coding at their own pace, with a focus on maintaining high standards for code quality and thorough reviews.*
- *The development of a unified toolset, including Airchat and MCP servers, aims to provide consistent experiences across different coding environments (IDE and CLI).*
- *A community-driven approach, including hackathons and knowledge sharing, has been vital for fostering engagement and support for agentic coding.*
- *Key lessons learned include the importance of removing barriers to entry, standardizing tools, and not compromising on quality when integrating AI into workflows.*

We will tell you about agentic coding at Airbnb. So, not looking at everything you can do with AIs, but specifically agenting coding. And I'm Szczepan, joining with Mike. We're engineers at Developer Group Platform at Airbnb, and we build tools for engineers and we integrate those amazing AI technologies. And I've been building tools for engineers for close to two decades. I've done open source. Maybe you use some of that. And it's been, I'm amazed where this like GenAI evolution is taking us. It's like, I've not seen such growth of productivity so far.

So, it's amazing. So at start of 2025, we set that vision. We told us that this is how we explain to our leadership where we headed. The way we like to do it, especially when we are tackling some big things, big step function changes, is we set up like a bold vision and then figure out how to work backwards from there. This helps because, you know, if you only think about incremental improvements, it's kind of hard to make a big change.

Now, we thought that this vision is bold. Now it's probably normative, right? We've seen the progression of the models and the tooling. But that's how we started earlier this year. S: And to put it into some like images, we

started with those like, you know, old ways of developing engineers, like typing letters and numbers and producing lines of code through what we've done a few years ago with copy pasting from ChatGPT and, you know, your co-pilot, all good tools, we still use them. But that's not where the magic happens. This is not this like paradigm shifting change where an agent can, sorry, when a developer, not an agent, when a human engineer can steer multiple agentic sessions and produce and materialize code changes, right? S: That's our direction. The way we've explained that to our leadership at start of 2025, we've used pair programming analogy for this. I have a bias for this. I like pair programming.

I used to do a lot of that. I used to do pair programming like entire full day from nine to five, from Monday to Friday, pair programming all the way. I loved it. It formed me as an engineer, is especially rewarding if you are pairing with an expert because you learn so much. But at the same time, the classic pair programming has also disadvantages, right? Sometimes it feels slow when you're making like a small changes, back fixes. Sometimes it's also taxing, it's tiring if you're like, you know, pairing entire day.

And in classic pair programming, you have a driver, an engineer who takes the role of like holding the keyboard and like being tactical and navigator who is strategic, who's going to see, you know, the forest, not just the one tree, right? So using that analogy, agentic coding is where a developer becomes this permanent navigator and it's steering that agentic session that and an agent or your tool is materializing code changes. That's where we headed. I think we're going to stick to that analogy this year, like in the future. S: I don't know if we're going to stick, especially if you paralyze and you have many of those agents.

But that's how we wrapped our heads around that, like at start of 2025. And we also gave some predictions, our leadership. And we were sandbagging them a bit too much, I think, because we were not ambitious enough, I guess, with those. We thought that like at the end of 2025, like materializing code changes through an agentic session is going to be maybe at like, you know, 20 to 40 % of like your PRs or engineers that are using it on a daily basis. So I'm showing you those predictions, but they are not very accurate because we are like right now that we're like 64 %. So it's going faster.

It's amazing. It's interesting. Now, those are predictions and they are useful, especially if you're explaining your vision to your leadership, to your workforce. But those are not KRs. Those are not how we measure success. And that's to measure the success, you need to apply a different methodology. What works at Airbnb is we try to get this holistic view of developer productivity. Things that like Vic was talking in his keynote. So we look at developer sentiments. We look at the tool usage and adoption. And we also look at the objective metrics from your engineering system. S: You can look at PR velocity and number of other metrics. Right.

To give you a glimpse of what the data is showing us like today and currently. So when we look at developer sentiments, they tell us the AI is amazing. This is the best thing ever. Like give me more. So last four surveys, that was like the top voted engineering productivity improving tool. Another prediction for the next six surveys is going to be the same. They're going to keep telling us this is the best thing ever. Right. We also look at the adoption of those AI tools. I'm specifically showing you the adoption curve of agentic coding, and it amazes me.

I'm showing two curves: One is some of the tooling that we built that wasn't agentic, with IDE plugins that aid engineers through this journey but not yet agentic. Only around six, seven months ago, we started going further into agentic coding. Certain models improved, certain tools emerged on the market or in open source, so it was made possible for us, and this is where we see this almost vertical line of eager adoption. I've not seen that ever, building tools for engineers for close to 20 years, such a growth, such eager adoption. S: We don't force engineers to use it. It's a supplemental tool. You can use it to augment your engineering or developer, but you don't have to. We don't force you. We do all kinds of things to encourage, to go on the road, to explain, to teach. We also try to integrate it as seamlessly as possible so that you are one click away from starting an agentic session, so those are the things you can do.

But it's amazing how the engineering community is adopting this. Also, we look at the outputs of the engineering system, and there are a number of caveats with that. This is just one of the metrics you can look at. We don't use it as a measure of personal productivity. We look at aggregate. We want to understand if those AI tools have an effect on certain objective metrics that we can view our engineering system. That's PR velocity. And I have a couple of curves here. One is on AI-aided tools and also something that started six to seven months ago, agentic coding, and we can see that there is correlation of some metrics of the engineering system, the PR velocity, and adoption and, like, leveraging those tools by our engineers. S: Okay, so we talked about agentic coding. Let's define it.

We often talk about inner developer loop, so we have our developer loop. At some point from that developer loop, an engineer would spec or prompt and will start the agentic loop. An agentic loop is a loop because it can call LLMs multiple times. It can call different tools, your MCPs, multiple times. It will leverage guardrails. It will load the config. By config, I mean, you know, your internal configuration, like system prompts, but also things like, you know, your agents MD, cloud MD, your cursor rules. Those are also, like, configurations. And I guess the key thing that I want you to take away from it is that, like, an engineer who is, like, leveraging agentic tool, like, you don't know how many...

You start your agentic session, but you don't know exactly how... What are the prompts that the agentic tool will use when it calls LLMs. You don't know exactly how many times it's going to call LLMs. You don't know exactly what tools it's going to use. It can run builds or tests or access your internal knowledge. It has this level of, let's call it, autonomy. S: Okay, so that's why it's agentic. That's why we think about it as a loop. At the end of that, there is output from that loop. Typically, those are code changes that a human has to review.

Okay, that's important. It must be reviewed by a human engineer before the PR is submitted. We also like to think about this agentic coding through, you know, what is the model of learning and acquiring skill in agentic coding. It is useful because it helps us shape that. It helps us understand, you know, maybe some engineers are going too fast or too slow on that journey, so that helps us make certain changes and work with our engineering community at Airbnb. Just to go quickly through this, we've all started with our first steps, and I think most of you are familiar with, like, level two, using your agentic tool. And initially, when I remember using agentic tool, I would approve every operation it does, and that's fine.

I learned. I got better at prompting. I got better at contextualizing. At some point, I leveled up to where I feel more comfortable putting my tool on auto-approve. Just keep going. Keep going. Show me the diff. Okay? Now, I wouldn't say that I trust the AI. I would say that I trust my skill in prompting and using the tools so that I get value out of it without approving every operation. Okay? But it takes time. It's a journey. Okay? And finally, at level four, this is like Mike, this is me. S: We're integrating those tools. We're making that transformation at your workplace.

Some of you probably are working in your dev infra teams, building MCPs, integrating those tools, contributing to open source. That's beautiful. Okay. I guess the point I'm trying to make, humans are still needed. They need to review the outputs from the AI before the PR is submitted for another human to review. I think that you want to give time to engineers to adopt, and they will have different pace in acquiring this skill and getting to really high proficiency.

Getting too fast to putting your tool on auto-approve is not great. What happens is an engineer may create very large PRs that they don't review fully. They don't actually understand and review every single line of code to fully understand. They may not review test code as eagerly as the production code. This is what I keep telling at conferences, like make sure your test code is of high quality as much as your production code. Refactor your tests, keep them clean and maintainable.

Another thing, to me, what's also important in understanding those changes that are coming from your agentic tool is pushing back on the tool. S: You don't trust it. If something feels suspicious, because I don't think that it works this way. Why? You ask it, right? Like, why did you do this in line 120, right? And then your agentic tool will be very confident and will explain you why, right? But after a couple of rounds, and they're like, no, but like, tell me why, you know, you get to this very satisfying moment where the agentic tool gives up and say, oh, yeah, you're right. Okay, I'm sorry. I was wrong.

Let's fix it, right? So that's like really important. Yeah. And I think that this, at some point, we are able to paralyze those workloads, okay? And I remember like my journey. So I would first when I was able to, when I felt confident to put my tool on auto-approve, yeah, I didn't actually start another coding task, right? I would review the design docs, I would create a design doc on the side. But at some point, I got more confident and I would have like multiple of those workspaces. Each workspace would have a agentic session, it will be doing certain coding tasks for me.

There are some power users at Airbnb that will have like five parallel workspaces at the same time, each building S: Another technical prototype of some sort, like pandemonium a little bit, right? Like in the keynote. But this is where the magic happens and we can multiply productivity. So we'll see what the, maybe the future engineer will not be good at flow, he's gonna just be very good at context switching, that's gonna be the main thing we need to learn, right? One of the things that has worked for us really well is we built tech at Airbnb that helps us have those sandbox environments, workspaces. It's called AirDev workspace. We presented this a few years ago at this conference, so there's a YouTube video on this. Because look, if you're parallelizing those agentic sessions, I mean, there's a way to do it on your laptop, on one machine, that like you can use Git work

trees, there are ways to do it, there's some open source tech that you can use. But we don't recommend it. I think that like when you can have sandboxed environments, like separate machines in the cloud, that is gonna be much better, it's gonna be more consistent, more reliable, and we're happy that we can leverage that at Airbnb because we built some tech for that. S: And I think my one takeaway that I want you to take from it is that like your agentic tool is not enough, you will have to have a lot of tech around it. Like you need to have a tech for your remote IDs, for your sandbox environments, and you need to have a really good code review tools and culture around that because who knows, maybe software engineering of the future is like all, it's all code review, right? That's all we do, right? I'm okay with that. Yeah, and I think that was my last slide. I'm gonna give you Mike now, and Mike will go deeper on things, I promise, right? Take it from here. Mike Hey folks, wow, what an opening. So one, before we get to my actual slides, I just wanna say I can't believe any of this works. I can't believe that any of this made it easier, but AI allows people like me, I'm a college dropout, I did Android for 10 years, I've recently switched to dev platform, it allows me to keep up with people like Chapan that's been, like you said, doing this for 20 years and running circles around me previously. So game on, I'm keeping up now.

So first, I wanna echo something that my VP said. We need to meet developers where they are. It's hard enough for all of us to learn agentic coding and AI, it's even more difficult if someone tells you, hey, you have to do it this way in this new tool and really just kick your legs out from under you. So in Q1 2025, I think this is when that big change happened. We all were learning a new way to code. Coding agents started to appear in CLI and IDEs, and this was industry-wide. M: It didn't matter if you're a infra engineer, front-end, back-end, maybe you were not even an engineer, but this really affected our whole industry and changed how we all do our job day to day.

I wanted to now talk about what a typical engineer looks like at Airbnb. And plot twist, we don't have one typical engineer at Airbnb. We're a company in our 17th year, and we're across the stack. Even where our developer environments are are not consistent. 60% of our engineers like working in those remote workspaces that Chupan talked about. 40%, like me, enjoy working locally and love their local developer experience. Similar, IDE or CLI. Confession, I don't know how to do Git from the command line, so I'm squarely in the IDE camp. I love IntelliJ, I've always loved it, and it does the things that I need.

We have other engineers that love Vim, that love doing things from the command line, and we also support them and allow them to code in the way that they feel comfortable. Languages are across the board. We use TypeScript, Java, Kotlin, Python, Go, Swift. I wrote my first Go and TypeScript classes with the help of AI at Airbnb, and this is one of those moments where I can keep up and start transitioning to platforms where maybe I didn't have the comfort to do before.

M: Similar, as AI started proliferating through Airbnb, it happened in multiple ways, and I'll dive into each of them. For one thing, we had IDE plugins. Think of it as almost like a cursor plugin that we had for VS Code, IntelliJ, and Xcode. But now, there were all of these open source tools that started coming along. And an issue that my team was having was, people would report bugs, and we needed a lot of context. It would always be, where did you run the tool? What slash commands did you have? What config did you use? There was too much split in how we were doing things.

It was different, and we sort of had to analyze, take a step back, and figure out what our strategy is going to be going forward. Our first strategy that we analyzed was IDE first. Like I said, we had these IDE plugins. They had beautiful UX. They had tight integration with current contacts. They can see what tabs you had open, code that's been edited, your local changes. And we also had this really tight integration with Airbnb knowledge. The one thing we had over the market is we had a rack pipeline. M: We had it integrated tightly with our IDE plugins, and we have Airbnb knowledge. So it's able to search an index, rack pipeline, and for a certain term, give me sources, and give me some information that was Airbnb specific. And that was a great edge at first. But then we started seeing the other side of it as agentic tools started coming along, all of these CLI tools, the AIDER, OpenHands, Gemini, CloudCode, Codex. And they all had these agentic orchestrations that could do multiple steps at once. It wasn't just me asking a question and getting response back. It wasn't just chat. Now it was actually calling tools, calling multiple tools, making decisions, and being able to interface by itself without me constantly copying and pasting back from that kind of like level one slides that we saw.

Okay, let's try a new strategy. This time, let's try CLI first. It's powerful. It gives you the best answers. It has the most amount of tools. It can actually run anywhere. A CLI is a terminal app. It's easier to install in these workspaces or even environments you might not have UIs. But unfortunately, there's no Airbnb awareness. M: A lot of our engineers love these tools and love them on side projects and use them at home and kept coming in and saying, hey, why aren't we using X? Why aren't we using Y? It's great at home.

We analyzed it and it just didn't do as well as we expected compared to our tools. It was kind of a, some things were great, but it didn't know our repos. It didn't know our design patterns. It was great for the side projects, not really great for giant mono repos and where most of our engineers were coding. So we wanted those quality of answers that were good. We wanted to have our Airbnb knowledge and we took a step back and realized that we need to invest more heavily in the system.

Cat's out of the bag. We have momentum going and let's actually spin up a team. So before we talk about spinning up the team, I just want to say that we scaled agentic usage to 60% of PR authors in the last year. Again, I can't believe this worked. I can't believe how quickly folks adopted it. And I can't believe how many people like me would probably quit if we took these tools away from people. M: So we started a DevAI team. We created this core team and a cross-functional working group to bring parity to all of the source, to all of our services. We wanted the same great experience whether you're an IDE or you're in the CLI. We also started creating a vibrant community. This is everything from hackathons to giving talks to having this working group of 30 folks that can then go back to their teams and kind of have this champion program of this is spinning up a new environment. This is like when we spun up mobile and we just started over. It all needs to be built from the ground up. So we thought about it. We're a small team. We have four engineers. We have some product support. And what are we going to focus on? What is going to be most high impact? So we took a bet on three things. First, we wanted to bring agents to Airbnb.

Actually, to be honest, our leadership wanted us to bring agents to Airbnb. This was one of those Friday night emails of, hey, I'm using cursor externally. It's great. How come our internal stuff isn't great? M: When can we have agents here? So we made a team. We figured out a way and kind of had a path forward to bringing these

agents internally, these external tools that people like to use, whether open source or not. The other side of it is we had to bring Airbnb to agents. We not only had to bring the agents in, but if there was Airbnb awareness, we had to somehow bind this to these agents and figure out a way to not train models.

We're small. You saw it as a team of seven people. And finally, there was this new standard that came out called MCP. For those that aren't familiar with it, think of it as REST for LLMs. It's some protocol, some way for you to wrap your code and give it access so that LLMs can call those tools. So let's start from the top, bringing agents to Airbnb. There's all these CLI tools like Aider, OpenHands, CloudCode, Codex, Gemini, and they're all kind of, you know, some of them were great three months ago. Tomorrow may be a different one will be great. And it moves really fast.

The quality of answers here is top of market. M: They can work in remote environments. And the speed to find a solution was faster for me and many of my coworkers than what we had internally. So we wanted a safe and simple way to adopt new CLI tools. And we did this by creating our wrapper. We created an abstraction that we called Airchat. And Airchat was an abstraction that allowed us to have multiple engines for us to quickly deploy and distribute these agents to engineer machines.

If someone had Airchat installed, single command install, we'd be able to automatically update it as codex came out or Gemini came out, or if certain models started being used within some of these tools. So this gave us a single entry point that we can continue to invest in. Really small, but kind of like that glacier that you see underneath of us being able to build functionality into it. So now that we have this management deployment for agents, we moved on to MCP. We went all in very early on MCP.

By that, I mean, I think my third week of being on a new team in a new org, I said, let's go with this crazy standard MCP that came out yesterday. M: I got a lot of confused faces, but at the same time, as we started looking through the standards and what they're doing, it really aligned closely to what we were already doing at Airbnb. We already had tool calling. We were already using tool APIs from the LLM providers and their SDKs. So what we realized is that this standard is very close to what we're already doing.

But because it's a standard, we will be able to publish and use our tools with clients that weren't just developed internally. So we kind of went down this path of first, we love our IDE tools. I love IntelliJ. We exposed our favorite IDE tools, favorite functionality in an MCP server. So now we can use those same tools, even if we're working in a CLI. The other side of it is we created MCP clients, way to interface with tools inside of our IDE plugins. And finally, we created a configuration and syncing mechanism within Airchat, within our Airchat CLI seamless so that we can now distribute these MCP servers and configure them and make sure that they're working correctly. It grew and grew and grew.

And this is another one of those like it's snowballing. M: We can't stop it even if we want it to at this point. Teams started contributing all over. Some teams started contributing by pulling in MCP servers or Republican open source. Others started building MCP servers for analytics platforms and CI and all of these internal tools that we couldn't, you know, you wouldn't be able to find those servers publicly. We now have over a dozen

MCP servers internally that we're able to bind to all of our different clients.

Back to my team, back to the core team. We continue to invest now in areas that we can speed up this MCP ecosystem. One of them is auth. Auth, no one likes to deal with. So we found this automatic way, gave you a pattern, some kind of framework that you can create MCP hosted MCP servers and we will inject your user auth into it. We also made sure to go through security. So now we have a paved path. As long as you do it in this way, you don't have to have a two week review with our security or even though this is new, we now have a way that is approved and consistent that everyone can build. We also created project templates. M: This helped. We have a culture of experimentation. We do a lot of hackathons and this allowed teams to experiment and build very cheaply MCP servers or clients around these MCP servers so they can iterate on them. We made it simple to install, almost like a homebrew install, one command and boom, your MCP servers either deployed or locally as a standard IO tool.

And finally, this is something new that we're now doing. We're building SDKs for building clients. We have a ton of MCP servers. At first we thought we would only have CLI and IDEs, but now we're starting to see people want to build agentic apps for everything. They want to integrate them into their web apps. They want to be building other desktop apps for it. So now we're standardizing on an SDK, taking the work of what we've learned the last nine months and now making into a framework that is more declarative for folks to be able to build their own clients that support MCP and support agentic orchestrators.

Little preview of our Airchat. This is our IDE plugin and this is when it was naive. It was just a chat app. M: You can ask it a question. You can get a response back. Looks very similar to most of these IDE plugins. So we wanted to bring agents inside of the IDE. Now that we have Airchat CLI, now we've brought in open source, folks are like, hold on, I want this good experience in IntelliJ, but can you make the answers as good as they are in CLI? Sure. We did research. We did research. We looked around and there was this great blog post called Effective Agents by Anthropic and it had a lot of the patterns that we were trying to standardize on and kind of think about. It gave us a shape. The shape we went with is that you have some orchestrator. The orchestrator lets you register many agents. An agent is a very large term. It sounds like it's going to be something complex. Effectively, an agent is a prompt and a list of tools that the agent can call along with a way to actually call an LLM. So we built this structure where each agent can delegate to another agent. You know, you can have a planning, a coding, and a validation agent. You can chain them together.

M: And it was starting to come along. It's a new product. We don't have the muscle and it didn't go as planned. AI moves way too fast. We were still planning, figuring out what we were going to do that existed in the market and the market now is five steps ahead of us. It wasn't going to work. Our team is too small and we don't have the muscle to be building these agents. So instead, can't beat them, join them. We pivoted and instead delegated to our Airchat CLI, to this thing that can call other agents and have this as a unidirectional data flow and a single responsibility. Our agent now is a CLI agent, but we have a thin shim above it that allows us to interface from other applications or other services. So let's go to our philosophy. We don't want to reinvent the wheel. I'm a big proponent of open source. Again, I came from mobile. I came from Android community and everything. We had more open source than we had things that Google gave us. We also don't want to confuse our users. We

don't want to keep churn and switching from one tool to another and being like, this is better, this is better. M: Figure out what your requirements are.

We realize what our users use. We talk to them and we try to stick to something unless there's a step function improvement as something new comes out. But things that we do want to do, we want to have the ability to continue conversations from one thing to another. So it's nice to kind of decouple your clients from the engines, your session, persistence, and anything else where a user might want to move from one thing to another. So maybe I want to start a conversation in a CLI and it gets too complicated. I then want to be able to move to my IDE and see it within that Airchat panel as well. So decouple what you can. Good development practices still apply.

The AI is not making your practices for you. But what we did want to do is focus on having the same feature set across the board, and then doing additive improvements that didn't exist anywhere else. So things like diff viewers inside of IntelliJ, things like menus for interaction that users can click with shortcuts that can highlight and do things, and just generally nicer rendering components. M: Not everyone likes looking at a CLI that's scrolling past them.

You know, we don't all live in the Matrix. Some of us like UIs. So now I kind of want to talk about how we brought this Airbnb knowledge to agents. So now we have our CLI, we have our agents, it has our unidirectional data flow, but that agent doesn't know anything about Airbnb. And I mentioned before, we had this rack pipeline and a REST endpoint where I can try to predict and say, hey, this part of the user prompt, let me send this to the search, and it would try to guess what search results I need. It was one shot, it was linear. This is all pre-agentic, and it wasn't something that worked well in this agent mode. So while we could have just exposed this and figured out a way to hack it in and, you know, like do this at the network layer, we tried to reimagine how knowledge bases should work. The way I think about it is tool calling is all you need. The world has moved. M: If I was given this talk last year, I may have told you that we would be fine-tuning, we would be training our own model, and we'd be investing in having the model actually have the Airbnb knowledge.

Some of our friends at other companies tried going down that direction, and it's expensive, and it doesn't quite work out well. My take is tool calling is all you need. It's all about the tools that you give access to your agent. So we re-architected this pipeline that was a single shot to instead be wrapped in an MCP server, and MCP tools, and descriptions of how this endpoint works. So now the LLM is able to be the navigator.

It's able to say, hey, could you call your knowledge base and get me information about some repo? And if it didn't like that answer, it can search based on other terms. We sort of for free got agentic search and deep research by just giving it an endpoint where it can get information about what we have at Airbnb. So even things like how do I set up an S3 resource? It wasn't a generic how you set it up. It was how do you do this at Airbnb using our particular tools? This was huge for us. M: So the last thing I want to go over is lessons, kind of like these war stories of what do we learn? What do we forget? What do we wish we did different? I split them into six categories.

First, it takes a village. This is too huge. This is not bringing a new CI to your company. You get a few experts, you put them in a dark room, and they're just going to iterate on it for a year and be able to keep up. This is moving too fast. The scope is too large, and the surfaces are something that's going to touch most aspects of software development. And it's going to take everyone. Don't think that you're going to be able to solve this yourself. Empower your local experts. They're the ones that are going to spread the knowledge. We've had so many talks and hackathon presentations and documents written and slash commands made by the folks that are not on this core team. And now it feels more like an ecosystem that I'm contributing to and others are and not just something that our team owns. This one was hard for me.

Don't call CLI an underdog. M: Like I said, I can't even do Git from the command line. But I can do Git when I use agentic tools. And maybe that's a little overusing agentic tools, but that's where I am. And those kind of little helpers, that kind of pair programming with it, lets me keep up with people that can think in CLIs. But at the same time, the CLIs have gotten so good at this point that we need to consider them. And we need to at least show our coworkers that maybe we're not CLI first, that it is a surface that they would be comfortable with. Today, I can say that I haven't used an IDE in months. I've coded across multiple languages. And now I'm one of these super cool CLI developers that doesn't need an IDE. Caveat, I'm doing it through agents, but at the same time, I'm able to debug. I'm able to ask for information. I'm able to do all the things that I used to use my IDE as a crutch for. And we saw something really interesting. 80% of our engineers feel the way that I do. It's not that our adoption tanked in IDE usage. It's that our adoption of CLI agent tools skyrocketed. M: This was almost hockey stick growth. And more and more folks started preferring CLIs and preferring multiple tabs open. And having this experience that felt like it wasn't just good enough. It was perfect for what they were trying to do. The next lesson is remove barriers. Make it easy to start an agentic session with a single click. Put it where your developers are. This is the whole meet your developers where they are.

Don't do too much at once. We considered, maybe for a few days, gabbing all of our IntelliJ users move over to VS Code because there's all these agentic tools in VS Code and all these vendors that can do it. And that lasted for a few days before I had to fear for my life that someone would come after me. It doesn't work. People that love IntelliJ love IntelliJ. People that love VS Code love VS Code. People that want to use Emacs love that as well for some reason. And yeah, you just kind of remove those barriers and put the tools where they are. And if you're going to make a new surface, make it as easy as you can.

M: Single command. We run Airchat. It's pre-installed. And boom, you're in your flow. You're in your agentic coding session. But you do have to standardize. The market moves fast, but don't just hop on every new trend as it comes along. You'll notice it. You'll notice what the industry is starting to stabilize on. To be honest, most decisions we made I think would have been much easier six months down the line. So what you should think about is, is it worth investment for you to be an early adopter before the market starts to stabilize a little bit and pick which one of these tools is the one that you should be using. Obviously measure all the things. It's hard. Some of it is quantitative, some of it is qualitative. There's companies like DX that are starting to do integrations with even agentic tools. So that's something to consider of. How will you measure productivity? A lot of the ways we've always measured productivity still holds through for agents, but I think we have a little bit of work to do here of figuring out what the impact of all of this is and how well it's working. M: And finally, don't drop your standards for AI.

I'm one of those non-believers. This is all going to be slop. This is all going to be vibe coding. This is going to be worse than what I can write by hand. And when I was using GPT 3.5, that was true. I personally got lapped by agentic coding. Now, I can't speak for you. You may still be able to do better than some of these tools do. But if you're not and you can't, you now have something that can get you to parity with folks that were doing things that you may have not felt was feasible.

So I just want to wrap it up and say thank you. I'm Mike. This is Szczepan. And we've had an incredible time both being here and spinning out agentic coding at Airbnb.