The future of agentic coding with Claude Code

20 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=IF9IV4XPONK
https://www.youtube.com/watch?v=iF9iV4xponk

SUMMARY

Alex and Boris discuss the evolution of coding with AI, particularly focusing on Claude Code, a tool developed by Anthropic. They highlight how AI has shifted from simple code autocomplete features to more complex agent-based coding that can autonomously handle significant portions of the coding process.

- *The landscape of coding has dramatically changed in the past year, moving from basic IDE tools to AI agents that actively participate in coding workflows.*
- *Claude Code allows users to interact with AI in a way that enables it to manipulate text and write code autonomously, reducing the need for direct text manipulation by developers.*
- *The development of Claude Code has benefited from a feedback loop where engineers use the tool daily, allowing for organic improvements based on real-world usage.*
- *Key features of Claude Code include a sophisticated settings and permission system, user-defined slash commands, and hooks for extensibility, making it highly customizable.*
- *Future developments aim to enhance Claude Code's ability to understand higher-level goals, allowing it to manage more complex tasks autonomously.*
- *For engineers, adapting to this new landscape means focusing on creativity and high-level problem-solving rather than just coding mechanics.*
- *Best practices for using Claude Code include starting with codebase exploration before attempting to write code and categorizing tasks into easy, medium, and hard to determine the appropriate level of AI assistance.*
- *The importance of learning foundational coding skills remains, but the focus is shifting towards leveraging AI to enhance productivity and creativity in software development.*

- I think back to when I
first started learning coding, I was the kid that sat in the back of math class in middle school, and I had my little TI-83 Plus calculator. And we just program it with BASIC, 'cause at some point I realized that I can actually program the answers for the math test into the calculator . Hey, I'm Alex. I lead Claude Relations here at Anthropic. Today we're gonna be
talking about Claude Code and the future of software engineering.

And I'm joined by my colleague Boris. - I'm Boris. I'm a member of technical
staff here at Anthropic and creator of Claude Code. - A lot has happened
in the past 12 months, and things are moving very, very fast, especially in the coding domain. For folks that, you know, maybe aren't following
the news every single day or even staying on top of the latest, and I have trouble myself sometimes, can you kind of catch us
up here on what's happened, and where are we standing currently?

- Yeah, a year ago coding
was totally different than what it is today. A year ago, if you want to write code, you have a IDE, you have some sort of
autocomplete in the IDE, and then there's some sort of chat app, and you might like copy and paste code back and forth a little bit. And that was the state of the
art, that was AI in coding. And I think maybe
sometime around a year ago we started to see agents appear as a thing that people
earnestly use in coding.

It's like a part of the workflow. It's not like a gimmick or a prototype. It's actually part of the inner
loop when you're doing dev. And I think this is the thing that's changed the most in the last year, is now when you code, you use an agent, you don't directly manipulate
text in an IDE anymore. It's not just about tab; it's about the model writing code for you. And I think what we've started to see is the shift from directly
manipulating texts to having the model do the
text manipulation for you.

And I think projecting it out, this is sort of the
trajectory that we're on, is this continuing into the future. - I see, so we've gone from it all being within a web app where you're copy and pasting the code out and you're making like very
targeted edits, almost, to just being a lot more hands-off and telling an agent
what you want it to do, and then trusting it to
go make tons of edits and create whole apps
sometimes even by itself.

- Yeah, exactly. And this was something that I think the reason we
couldn't do it a year ago, and, you know, like, people
have tried to make AI do coding for the longest time and to, you know, just like automate more and
more of coding in various ways. And it hasn't really worked, I think, probably for a couple of reasons: one is the models weren't
really good enough, and the second one is
that like the scaffolding, the thing on top of the
model, wasn't good enough.

And when we initially
launched Claude Code, the very, very first
versions late last year, I think it was still using Sonnet 3.5. This wasn't even 3.6, or whatever we call this thing, the new Sonnet 3.5. - Yeah, upgraded Sonnet. - Yeah. It wasn't even this. And it like sort of worked, you know? Like, I used it for maybe 10% of my code or something like that. But even then, I remember
when we launched it, we gave it to the core team.

And it was just me and like a few other people
on the team at the time. And I remember walking in one morning, and kind of on the way to to my desk, there was a few engineers sitting there; and one of them was Robert, and there was a couple of other engineers. And I just walked in, and I saw Claude Code on
their screen the first time. And like I just gave this
to them the day before and they're already using it.

And it was just the craziest thing. And the model wasn't very good. The harness wasn't very good. But even in this early version, it was already a little bit useful. And I think that over the last year what's happened is the model has gotten way better at agentic coding, and that's happened with like
3.7 and now 4.0 and Opus 4.1. And the harness has also
gotten a lot better. And, you know, obviously
the harness is Claude Code, because the way you
interact with the model, you can't just like
directly use the model: you have to use a harness.

It's sort of like, you know, like if you're riding a horse,
you need some sort of saddle. And like that saddle
makes a giant difference when you're riding a horse. I'm not a horse rider. - I like that analogy, though. I mean, it is kind of
like Claude is the horse, and as the engineer you're trying to get it to go in a certain direction and you're trying to guide it, and like you need some sort
of scaffolding around it to be able to steer it correctly.

And the harness in this case, just so we're on the same page, is everything from like
the tools we're giving it to how we handle like the
context and everything for the model. - Exactly, exactly. It's like all of Claude Code. Like, the model is the thing behind the API, and then Claude Code,
it's the system prompt, it's context management, it's tools, it's the
ability for, you know, to plug in MCP servers, settings, permissions,
all this kind of stuff.

All of this interfaces with the model. And the model sees all the context, all the output from this stuff, and it makes a giant difference
in the way that it performs. And I think over the last year we've learned how exactly
we build for the model. And the model has kind of coevolved with not just Claude Code but all these different products that are using Anthropic models to build agentic coding tools. - Maybe let's speak more on that.

When you say coevolve, is that because it's
like a deliberate thing in which we're doing with the training, or how is the model also getting better at these sorts of things? as we make the product

features itself better. - It's pretty organic, honestly. Like, you know, at Anthropic, everyone uses Claude Code. And that includes the researchers. And so every day the people building the models are using the model in order to do their job. And I think as part of that you kind of see these natural limits that you hit with a model.

So, you know, as an example, maybe the model's really bad at doing certain kinds of edits. And sometimes when you use Claude Code, you see like, oh, failed to replace string, failed to replace string. Like, this is a model capability, and we can improve this if we learn from it. Or another example, maybe something like higher level is if you just let the model cook for like 30 minutes, with 3.5, it could kind of do it for a little bit, maybe for like a minute or something it would stay on track.

And then with newer models it kind of gets longer and longer this amount of time the model can operate autonomously. And I think this is really based on experience, because you use the model, you kind of see where as a human you have to course correct and steer it. And then we've learn from that, and we can kind of incorporate that into the model and teach it better to do this itself. - When you're evaluating a new model, do you kind of have a vibe check set of tests that you run?

Or if it's like a new feature that we're rolling out to make something better in the harness, how do you personally evaluate if the performance is getting better? - I just do my work that day. - Interesting. - Yeah. Like, my perfect day is I'm just coding all day. And, you know, whatever the model is, whatever is the new thing we're testing, I'll just code using that and see what the pipe is. There isn't like a specific thing I do.

- Right, you just see how does it actually work for me in my day to day? - Yeah. Exactly, exactly. And, you know, like in day-to-day work you do all sorts of stuff. Like, you're writing new code, you're maybe like fixing bugs, you're maybe reading Slack messages or GitHub issues to respond to feedback. And I think more and more the model is able to do more and more of this. So actually, in a way, if you had maybe one thing that you always use the model for, you would miss out on some of these newer capabilities, like pulling in context through MCP, like reading your Slack messages.

Or, you know, automatically debugging stuff, 'cause you can pull in Sentry logs automatically. - Yeah, so the best eval in some sense is the one that most looks like real life. And in that case, just using it gives you the best result. - We tried really hard,

when building Claude Code, to build a product evals. - Yeah. - You know, just like to have some sort of benchmark; like, when we change a system prompt or whatever, is the model getting better?

And we have a little bit of this, but honestly it's just like so hard to build evals. And by far the biggest signal is just the vibes. Like, does it feel smarter? 'Cause there's such a broad range of tasks they use it for. - Yeah, that's actually a question I hear from developers all the time, is they would appreciate more guidance on how we go about prompt testing and iterating. I know for different products we have like various sorts of evals that we've tried to create, but for Claude Code it really is just kind of this tight feedback loop that almost gives us like more immediate signal than any hardcoded set of evals.

- I wonder if people kind of want to hear a better answer from an AI. But yeah, man, it's all vibes. I think at this point we're, you know, the models are doing so good on evals, like SWE-bench. You know, we're just trying to find these harder evals. And now there's like T-bench, which is like a little bit less kind of saturated. But I think it's just really hard to find synthetic evals that capture all the complexity in software engineering.

- Right, right. Do you think there's something we did uniquely to set up that feedback loop internally? 'Cause I feel like Claude Code has like the best dogfooding cycle I've seen of like any type of product. - Initially, I built it the way that I do any other product, which is just listen to users and make it as easy as possible to listen to users. And I think one part of it is when we built Claude Code, there was just like a single feedback channel in Slack.

And anytime anyone had feedback, I would just direct them to that, just be like, "Yeah, post there." And I feel like people hesitated sometimes a little bit. 'Cause sometimes when you give feedback, you expect that no one listens and it kind of goes into this black hole, like into a void. And I think one of the things that we did really right was, from the beginning, whenever someone gave feedback, I would try to fix it as fast as I can.

And sometimes I would kind of go into the office and then just spend like three hours or two hours or whatever, just go through as many bugs as I can and fix them as fast as I can, and then every time comment back and tell people it's fixed. And this kind of encourages

them to keep giving feedback. And to this day the Claude Code
feedback channel internally is just this fire hose, just nonstop. - Oh, totally.

I remember, on those
early days, and still do, dropping in there, posting something, and immediately your emoji reacting. Or you're asking for more
clarification and more questions, and you do feel like, oh, okay,
my feedback's being heard. And then you're able to like actually be, you know, incentivized to go post more feedback in the future. - Yeah, 'cause, you know, honestly, like, I don't
know what I'm doing. Like, no one really knows
what they're doing with AI.

Like, we're kind of discovering
this thing as we build it. And the best indicator
is what the users want. So you gotta listen. - Right, switching gears slightly, what is like the current state
of Claude Code as a product? What are the latest features?
What are you excited about? Some things that you're seeing
folks do with it right now? - Claude Code, from the start, was built to be the simplest thing it can and to be as hackable as possible.

And I think the hackability is something that we've been developing a lot, and that's something I'm
really excited about. So originally, the way to hack Claude Code is adding to its CLAUDE.md. That was the original extension point. And CLAUDE.md, as you
know, is like this file. You can put it in the root directory, you can put it in child directories. There's kind of different
places you can put it. And it's just additional
context to give Claude Code, and it kind of goes with your repo.

You often check it into your code base. So it's kind of, you know, a little bit more
information about the code. But over time we've added a
lot more extension points. So now there's a very
sophisticated setting system and permission system. There's hooks now which Dixon built. Dixon's an engineer on our team, and he just kind of saw all
these different user asks coming in for: "I want
to extend it this way. I want to hook into this, hook into this."

And so he built a super
extensive hook system. MCP, obviously, this is a
really great extension point. and now there's slash
commands and subagents. And user-defined slash commands is something we've invested in a lot. And the idea is it's just a workflow: it's like a markdown file. You put it in your code, and it's something that

you can reuse a lot. So for example, I have a slash command for making commits. And I have some instructions in there: here's how you write a good git commit.

I pre-allow the git commit Bash command so I don't have to accept it every time, and the model can just do it. So I think slash commands
are really interesting, and agents are kind of a
different view of slash commands. Like, it's like a slash command, but it has a forked context window. And so you can kind of think
of agents and slash commands as two sides of the same thing. And this is also very exciting.

It's just another way
to extend Claude Code. And so when I look at the future, I think a lot of it is just about like how do we extend Claude Code more? How do we make it easier for
other people to build on top? How do we make the SDK
more useful for people? So it's useful for code if you
want to build a coding agent, but also you can use it for other stuff. Like, anything that you need an agent for, you can just use the SDK for.

And I think these are the things that I'm the most excited about. And obviously all of this benefits from all the other work we're doing to make the model more autonomous, to make it work for
longer periods of time, to make it better adhere to instructions, to make it remember things better. And so everything along
the way it benefits. - So I'm using Claude Code, or whatever form of it,
in six to 12 months; what does my work actually look like?

Am I reviewing PRs all day, or what does it day to day break down to? - Yeah, I think there's gonna be a mix of more hands-on coding. I don't think that's going away. And maybe it'll look different, though. So maybe hands-on coding today is directly manipulating text, but in the future it might be using Claude to manipulate the text for you. And then I think there's
gonna be this other bucket of maybe less direct coding where Claude proactively does something, and maybe Claude even reviewed it.

And it's your job to
decide if this is a change that you want or not. And I think maybe 12 or 24 months from now we're gonna start seeing
Claude that's more about goals and more about these higher
level things that it needs to do and less about the specific
tasks that go into it. The same way that, as an engineer, I think about what is it that I want to do over the next month. And I kind of make small
changes to work towards that.

Maybe Claude will go
through the same thing. - Right, sort moving up and
up the stack, to some degree, of these like abstraction
levels of getting Claude to make individual changes to files, to getting Claude to make
changes to a whole PR, to getting Claude to think
about a goal of building an app or whatever else it is. - Yeah.
- Okay. That's interesting. If I'm an engineer and I'm hearing that, it seems like there's
gonna be a lot changing in a very short amount of time, especially with my role
and what I should be doing.

What's your advice for folks out there that are looking to prepare themselves and adapt to this world? about what they should be learning or what skills they should be developing. - I think back to when I
first started learning coding; I was the kid that sat in the back of math class in middle school, and I had my little TI-83 Plus calculator. It was like a transparent gray one; you can kind of see the circuit. And we just program it with BASIC, because at some point I realized that I can actually program the answers for the math test into the calculator .

And you can get better grades that way. And there's just something about
kind of this visceral feeling of being able to hack,
and having this idea of maybe there's this
one program I can make; and just I go into my
calculator and I code it, and then I can just restart
and use it really quick: this kind of feedback cycle
that was really amazing. And it made it possible
for me to build stuff that I never could have before.

And it was just so easy to get started. And I think about the difference between that world and the
world before agentic coding, where stacks just got
way, way too complicated. You know, if I wanted
to make a JavaScript, you know, like website, I had to learn about
React and maybe Next.js, and then three different build
systems and a deploy system. And it was just so complicated. And I think one really
cool thing about agents is that they're changing this.

So with coding agents it makes it really easy to get started. And if you have an idea
you can just build it. And it's a lot more about the idea now than it is about the details, because just like Claude Code, you can rewrite the code over and over. And, you know, Claude Code
itself, we rewrite all the time. And I think this is just something that coding agents enable. The code itself is no longer precious.

And there's still an art to writing it, and, you know, all stone

code by hand sometimes. And one of the engineers
on the team, Lena, she was talking about how on the weekends she still sometimes writes C++ by hand, just 'cause it's fun. And, you know, as a coder, it can be a really
joyous thing to do this. But I think more and more it's gonna be about the thing you make and not about the process
of making it as much.

And I think my advice for
people learning to code today is you still have to learn the craft. So you still have to learn
to code, learn languages, learn compilers, runtimes,
how to build web apps, how to build programs, system design. You still have to know all the stuff, but also just start to get more creative. And, you know, if you
have an idea for a startup or an idea for a product,
you can just build it now in a way that you just couldn't before.

And we don't really
understand what this means, but there's just so much potential that's about to be unlocked because of it. - Yeah, I love that. I think that's great advice too. Ideas suddenly become
something you can action on in, you know, a span of a few minutes almost; whereas before it could be
just in your backlog forever. Before we wrap, I want to ask you, as the creator of Claude Code, what are your best practices
for using Claude Code?

And any tips or tricks. - Yeah, I think the biggest
thing that I recommend, okay, maybe two tricks. So one thing I recommend is that if you're brand new to Claude Code and you haven't used it before, don't use it to write code. I know it sounds crazy.
- Yeah, explain, explain. - But you gotta stop yourself. Like, don't use it to write code yet. The thing to start with is use it to ask questions about the code base.

So you can ask, you
know, if I want to make, if I want to add a new
logger, how do I do that? And then ask Claude Code
to explore the code base and figure it out for you. Or why is this function
designed the way that it is? Claude Code can go in and it
can look through Git history and it can answer this stuff for you. So I think ask Claude Code
questions about the code base and just don't code yet.

And then once you feel comfortable with using Claude Code this way and you get comfortable
with this idea of an agent that's doing this research for you, then start to use it to code. I think the second thing is when you are using Claude
Code to write code, think about what kind of

work do you want to do and like how big is the task? So for something that's really easy, I kind of, in my mind, I have these three categories: easy, medium, and hard, very roughly.

And so easy tasks are something that Claude can write in one shot; like one prompt, it'll
get it pretty much right. And nowadays I'll just go to GitHub and I'll tag @Claude on an issue and just have Claude write the PR for me. And this is how I do easy tasks, 'cause that frees up my terminal. I don't have to kind of spend it on this. Medium tasks, I'll start
it in the terminal, and I'll start in plan mode.

So just Shift + Tab into plan, and I'll align on a
plan with Claude first. And then once I feel good about the plan, I'll go into auto-accept and
I'll have it implemented. And then for really hard tasks, I'm still the one driving, and Claude is more of a tool. And I'm kind of pairing with it. But really I'm the one
in the driver's seat, not Claude for this. And so I'll use Claude maybe
to do some code-based research, maybe prototype a few ideas, maybe I'll just like
vibe code a few options to understand the boundaries of the system and what works well.

But I'll still mostly implemented myself. And maybe Claude will
write the unit tests, but it's still mostly me doing the coding. So I think that'll be the second advice, is just think about what's
the task that you're doing and what's the right way to
use Claude Code to do it. - Those are great tips. Really, really appreciate the time, Boris. This has been awesome. Thank you. - Yeah, thanks, Alex.