wtf are loops?

12 MIN · YOUTUBE · HTTPS://YOUTU.BE/GI0LX6C84VE?SI=BXLMK5AIHNYW5COW
https://youtu.be/gi0Lx6c84VE?si=bXlmK5AiHNYw5cOW

SUMMARY

The discussion in the coding agent community centers around "loop engineering," a method to automate coding processes using agents. By setting up loops, developers can streamline their workflow, allowing agents to handle tasks like triaging issues, implementing features, and preparing pull requests, thereby reducing the need for constant human oversight.

- *Loop engineering automates the coding process, allowing agents to manage tasks with minimal human intervention.*
- *Traditional in-the-loop development requires constant human presence for task initiation, monitoring, and code reviews, leading to potential backlogs.*
- *The new approach involves agents triaging issues, implementing solutions, and opening pull requests based on well-defined tasks.*
- *Developers can set up automations to trigger agents to work on tasks while they are away, enhancing productivity.*
- *Clear task descriptions and acceptance criteria are essential for agents to effectively execute their roles.*
- *The process can be expanded to include multiple agents for planning, implementation, and code review, further reducing bottlenecks.*
- *Open-source tools and templates are available to help developers implement these automations in their own projects.*
- *Future developments may include agents that can self-improve and adapt based on feedback, further enhancing their capabilities.*

There's a new discussion in the coding agent community right now around loops, or loop engineering. Both Boris from the Claude Code team and Peter Steinberger from OpenCode have been sharing their approach here. I don't prompt Claude anymore. I have loops that are running. They're the ones that are prompting Claude and figuring out what to do. My job is to write loops. At first, it sounds like some revolution, but honestly, it's not much different than how you're using coding agents right now. So I want to walk through the basics of what a loop is, what they're talking about, and walk you through the simplest automations all the way to the most complex. Because I've been using this approach both on my own projects and also as a member of the Warp team maintaining an open source project. So let's start with what you're probably familiar with, which is in-the-loop development. You start by opening a new process in your terminal or the Codex app, whatever you're using. You send a prompt for what you want to build. It goes through its usual research, code, test, pass loop. Then ideally, you go through a code review step to take what works and make it good, simplifying and addressing edge cases. And then finally, you merge it in. But this whole time, you're monitoring that process. Maybe you're tabbed away scrolling through YouTube and Twitter and coming back. But in any case, you're the one who kicks off the prompt. You're the one who monitors the work, reviews the code, and merges it in

manually. There's nothing inherently wrong with this approach, especially if you're using worktrees to parallelize and run multiple agents without them stepping on each other's work. But it does mean that you need to be present for multiple steps in this process. First, you need to be present to go grab the linear ticket, GitHub issue, or whatever else you're sourcing the prompt from, and make sure you're present to hit that code review button and hit merge into your code. And in my experience, this tends to lead to a backlog that isn't getting resolved. It's easy enough to log what issues you run into when you're working on an app or writing down what your users run into. But if an agent isn't proactively checking this thing and kicking it off, you have to time block your day to make sure, oh, I'm going to sit down and I'm going to ramp through 10 worktrees on my machine, which, let's be real, we're not always in the place to do that. In a perfect world, an agent would come through this list of issues that I've resolved and just start working on it. Maybe it'll write out an implementation plan if it's a larger feature, or maybe it'll work all the way to a pull request if the goal is clearly defined in the issue. So that's where we get to loops. The most basic flow of this being you have a log of all of your issues stored on whatever task board you prefer. You have an agent come along and triage those issues, perhaps on a GitHub trigger or perhaps on an automation so that it's a manageable amount of work coming down the pipe. Then an agent kicks on to take down that issue, work on it, verify its work using an automated test suite or even computer use, and then open PR changes for you to look at. Before, you were involved in every step of this process, remembering to spin up some worktrees to go through GitHub issues, spinning up agents to act on those issues, having an agent review its code and open the PR, and then finally merging it into the codebase. But with this new setup, you're slowly removing bottlenecks at each step in the chain, where it's all automated right up until you get to the code review step to let things into the codebase. If you're brave enough to just let agents vibe-code every aspect of your app and do all of your QA, you can remove the human bottleneck entirely and just let this thing run into oblivion.

But I don't think we're at a point yet with model intelligence to let agents merge their own work into a codebase. So we still need this final human bottleneck at the end of the process to make sure code is good before it gets merged. But at the very least, this is removing a lot of bottlenecks in the process so that you just spend time on the initial issue writing and on the other side at code review. So how do you set up the simplest possible loop to write down everything that you want done and have an agent wake up and do it? First, in order for this loop to work, you need good issues for the agent to work on. And here's my basic process to write down a good task for an agent to pick up without me looking at it. I tend to start with the grill with docs skill. This is something Matt Pocock put together. It's kind of a planning mode on steroids, where we'll ask you as many questions as it can think of to drill into how the code should work.

So at the end of the session, you walk away with a really clear plan of what to do. This is a long discussion that I had working through a number of questions on a large feature that I was trying to build. At the end of it, I had the agent draft a couple issues using a issue template that I have. You can see here, the agent broke up this long discussion into a couple issues to work on. The first one being a scoped out issue on how our CLI should be built. And you can see I've already had an agent work on this in the past. You can see the template is fairly simple. We start with what to build. This was a fairly scoped out feature. We could just say this and describe the acceptance criteria for the agent to work towards. This is the first major aspect of a loop, which is describing the goal that the agent should walk towards. These days, agents have a goal skill. Both Claude Code and Codex have this, which will drive an agent to work towards a set of acceptance criteria before it returns back to you for control.

Now, once you have your issues set up with goals outlined, we need an agent to come along and work on this issue with an automation. There are a number of solutions out there to turn on agents when you're AFK, all the way from hosted cloud solutions to cron jobs that just wake up a task on your machine.

I'm going to talk about that second one. If you're a Codex app user, there's this tab called automations that you can go to. You can create new automations just by asking the Codex app what automation you want to do. Then it will create a new one inside of here. I have a couple that I've already set up. One of them for writing specifications for big features and another one to work on issues that already have that acceptance criteria that I showed you. If we click into this, you'll see it's a very basic automation. We just have it go to that repo, look for anything labeled ready for agent or do nothing if there aren't any issues open, pick up an unclaimed issue. The one bit of instruction that I added was to always use computer use for UI changes, which is a feature that Codex has to let the agent click around the app that it built. Otherwise, I just defer to the acceptance criteria in the issue to guide the agent to a working solution. This is a sample run that kicked off recently, you can see it picked up the first well-scoped issue it found, it opened a pull request. And I'm also free to review all of the code that the agent worked on. The beauty of this is I don't have to sit down and wait for a triage task to run, I can open whatever coding agent that I prefer to use. And I can look through all of the runs that completed and review the code on the other side. In other words, it lets me be as lazy as I want to be. This is the simplest automation you can set up just to implement tasks that are well defined. But you could apply this to a lot of aspects of your work. For example, taking an underspecified issue and turning it into one that's well researched, or going from a problem statement to a written out implementation plan. And then you can hand that implementation plan to another agent to work on. Maybe add a PR review agent to work on the code review so that it gets a second opinion. You can go crazy with it. As a preview, here is another agent that I've set up on this repository that goes from an underspecified issue to a specified plan. So here's an issue that I filed while I was working on another issue, where I realized there was a technical constraint in something that we were building, but I didn't really have time to work with the agent on what that should look like. All I knew is roughly, we should solve this problem at some point. So again, I use the same templates, just describing what to build with acceptance criteria.

I applied a different label to kick off sort of a different automation flow that I've called ready for spec. Here I have the automation that looks for those issues. It's the same as the previous one, just looking for ready for spec. You'll notice here it's applying a few rules or skills that I've written that describes how to write a good spec document. Here, instead of having the agent just implement this spike I documented, I want the agent to actually plan out what it would look like on the product side if we implemented this, and also what the technical architecture would be. You can see what planning documents the agent put together for this issue. This is going to be highly opinionated. It will really vary depending on your project. You can see here that I prefer my tech specs to include the affected apps and packages. Since it's a monorepo, I want to see what apps are we going to touch, what packages and shared logic are going to mean version bumps, and then an overview of what it's going to do and how it might define the acceptance criteria to hand off to an implementation agent. From here, I could go back and forth with the agent. If I feel good about that spec, I can merge it into the codebase. I prefer to merge all the plans that I write with an agent just to have the paper trail. Then I can flip this label from ready for spec to ready for agent.

That way, the other automation can pick up this plan as acceptance criteria and implement it for me. So now

I've 2x'd my laziness factor. Agents handle all the planning for me and agents handle the initial implementation. If you're curious about those spec documents I showed you and you want to invent your own skills to teach agents how to write specs, I would suggest pulling down the open source specs that Warp already has. That is the open-source terminal that you are seeing right here. And we have a repository called common skills where we've documented our tech spec and product spec documents.

Here's what I did for my own project. I just asked Codex to go look at those product specs and adapt it to my own project. You can see the fine details are very scoped to my project, detailing all the aspects of the monorepo and how I prefer these things to be written. So adjust to taste. But I do love this idea of having an agent take a rough idea to a solid idea and then another agent take that to implementation. To expand your brain a bit on what loops can do, we've been applying this process to manage the open-source repository for the Warp terminal. You can see it runs across thousands of issues and we use very similar labeling to what I just showed you, like a ready-to-implement label when an agent can come in and open a pull request. We've also added more automations for each step of the contribution flow. For example, when a user files an issue, we have a triage agent decide if it is well documented and do a bit of initial research so that a maintainer or another coding agent can understand what it would take to implement that request or bug report. This really reduces friction when you're working with a whole team of developers. On this issue, a triage agent was able to come through and define it as well scoped. We were able to have one of our maintainers come through and agree on the implementation, apply this label, and then another contributor stepped in and said, hey, thank you for all that information. I'm going to stick an agent on this and open a pull request. You could argue that's sort of a human in the loop, but you get the idea. We're trying to remove bottlenecks on your own team so that you can get features over the line more and more often. You'll also see another agent that we stitched on to the PR review process. Instead of a maintainer looking at a community contribution and deciding whether it's good enough to even merit a code review, we can have an agent do all that work for us and look for any critical or important bugs before we take a look. If that excites you, we open-sourced this stack. It is a bit more involved the setup because you will want cloud runners so that it doesn't have to run on a worktree on your machine. Once you get it set up, we have templates for that issue triage that I showed you, PR review, and anything else you want to add. I hope this got your brain turning on how you can flip in-the-loop work to out-of-the-loop work, bringing in simple automations to go through your backlog and start working, and then adding more and more automations over time to fill in the finer details. The last way to level up this journey is actually having agents self-improve over time, having them update their own skills based on your feedback. If you find that topic interesting, drop a like and subscribe because I'm going to be talking about it a lot more in the next video. See you around.