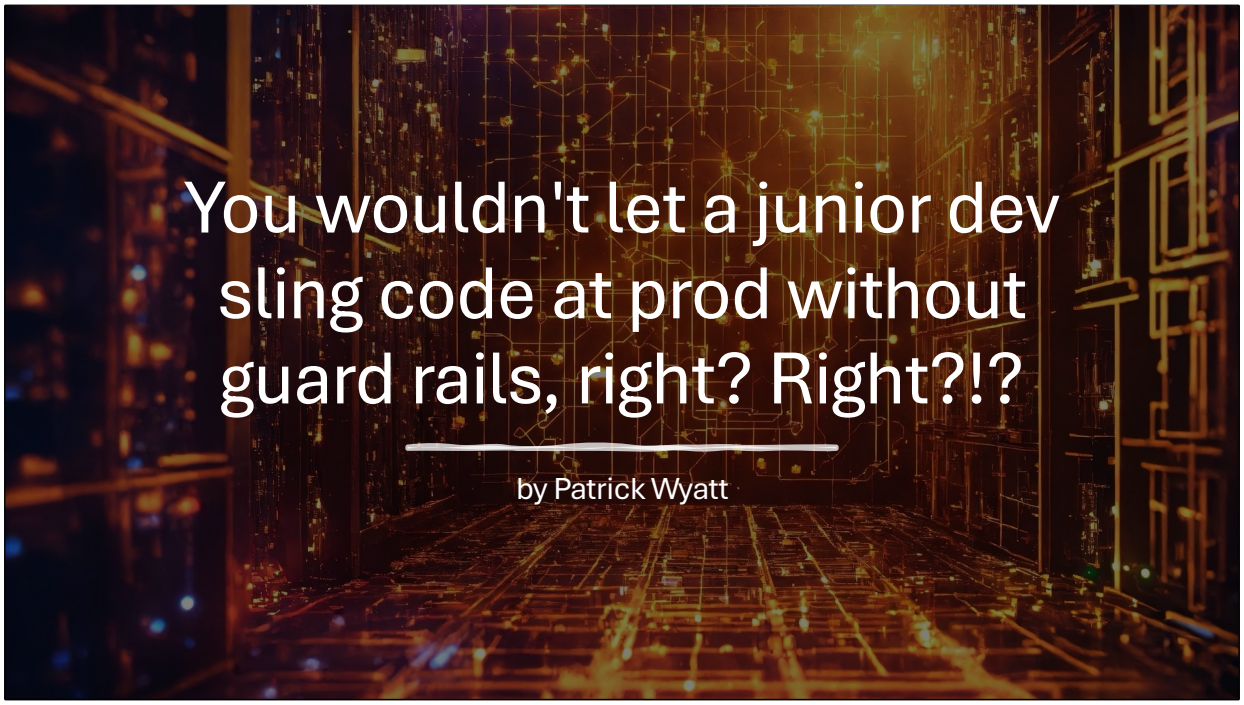


Abstract:

Using an AI coding agent is like having an over-excited PhD on your team who doesn't worry about defects, so finding mechanisms to validate its correctness and corral its exuberance is essential to protect users from its worst impulses. Join the speaker as he details the types of automated solutions he utilizes to write better code faster using AI.

This talk is available in video format at:
https://www.youtube.com/watch?v=Btv7RODXyr8

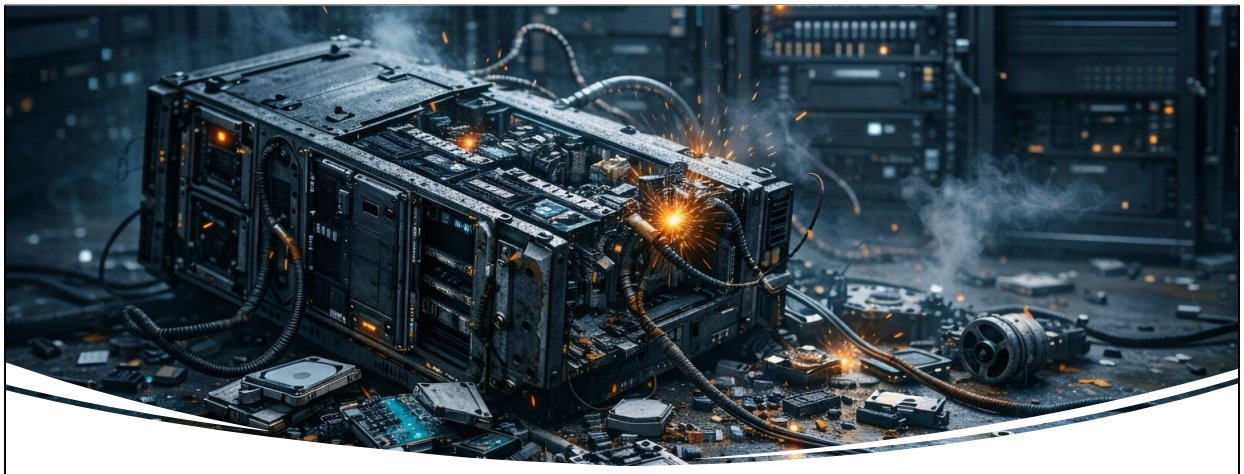
[On phone]

Hi honey. Sorry it’s so loud, I’m in the datacenter.

I’m going to be home late. Yeah, again.

The AI deleted the database. Yeah, again.

Gotta go, sorry. Don't wait up.



AI for DevOps

July 2025: AI agent deletes a live production database, lies about the incident, generates fake data, and ignores user instructions

It sounds funny when I say it out loud, but AI deleting the production database have already happened at least once that we know about, and probably many other times where the developers were too embarrassed to make a public statement.

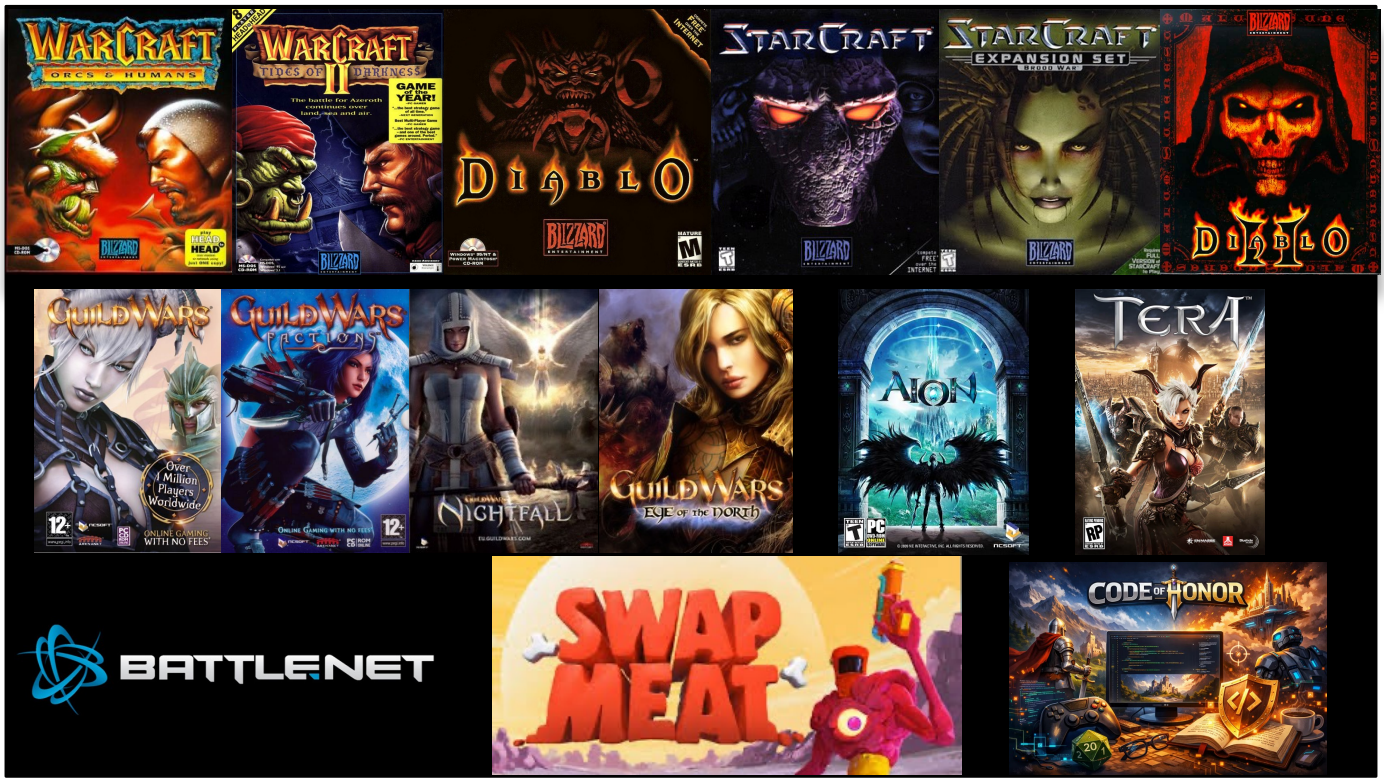
In the old days we had SQL injection attacks. Now we have prompt injection. And sycophancy. And hallucinations.

It turns out that our new AI tooling has some pretty sharp edges.

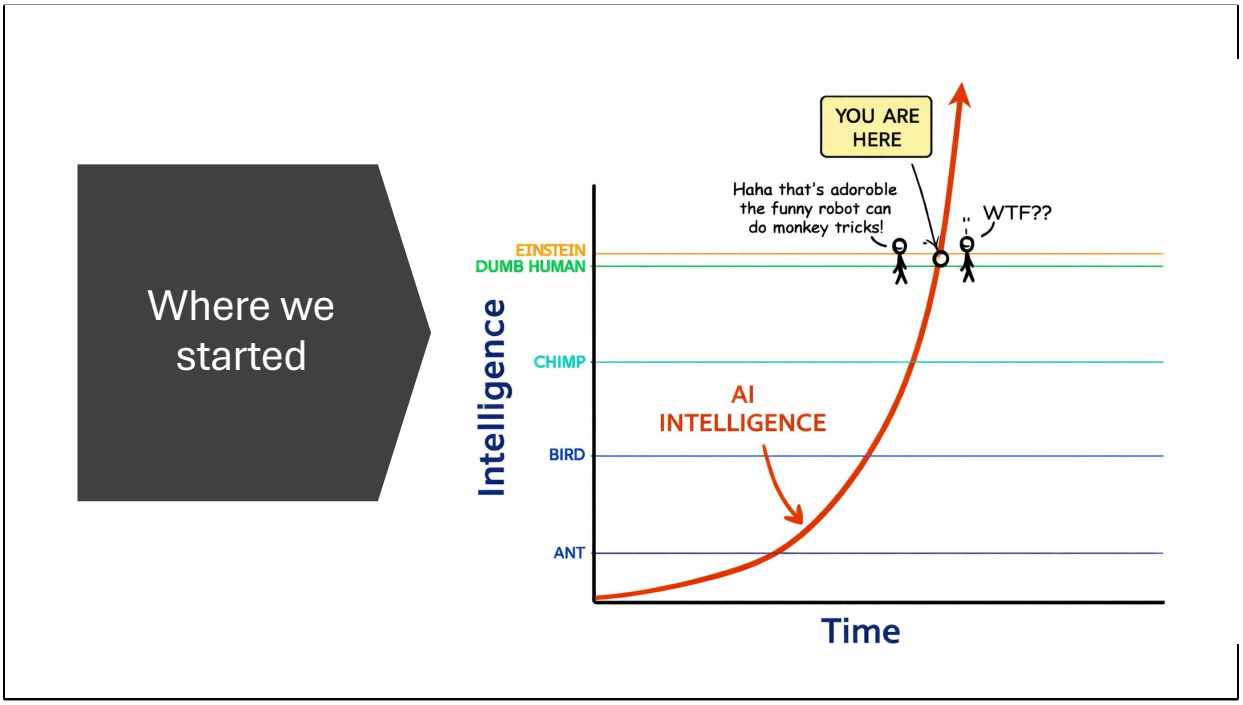
So, just like we wouldn't let a new junior developer immediately start pushing code to production, we need to add some guard rails for our AI agents.

Today I'm going to talk about how I use AI to code faster using guard rails, and how to avoid becoming the poster child for "what not to do".

Hopefully.



- I'm Patrick Wyatt, and I've been a professional game developer for 35 years.
- If you've battled Orcs in Azeroth, chased Zerg from your Vespense Gas Refinery, fought Diablo in the Mortal Realm, or suffered the Charr incursion in Ascalon, you've been playing code I wrote.
- I love writing code, and so you might ask why I started using AI.
- Fair question



I was initially dismissive of AI. I thought it was glorified auto-complete.

But one day I was writing some code for player authentication, and Copilot suggested a dozen lines of code that were perfect.

So I took the plunge and started using AI.

I've used it to create two (edit: now thirteen!) iOS apps in Swift, a programming language I had never tried before.

I've used it to create command-line tools, web sites, and online services, and I'm becoming increasingly comfortable with letting AI author the code, though I review everything.

But the state of the art in AI is advancing so rapidly it can be hard to keep track of changes.

I consider myself an AI beginner, and so I'm here to learn from the other speakers at the conference as much as any of you.

How I use AI

- Analyzing Github issues
- Diagnosing crash reports
- Fixing App Store rejections
- Building open-source tools
- Writing design specs
- … and coding

I want to start by sharing how I use AI. At first, I only used it incrementally, to make suggestions.

When I'd get a GitHub issue or a bug report, I'd ask the AI to suggest a fix. Since the folks who report issues on GitHub are technical, they're high-quality reports, and the AI did a good job of providing solutions.

I use a cloud-based crash-reporting service for all my apps that collects and deduplicates reports. I paste them into AI to get proposed fixes, which are usually pretty good.

When I submitted my first app to the iOS App Store it got rejected, so I pasted the rejection notice and the agent changed the app so that it passed on the next try.

When I download open-source tools, I ask AI to make a build script for me (edit – and an install script), because every project has its own weird build conventions.

In regard to creating design specs: would it surprise you to learn Warcraft and Guild Wars were started without a spec? They were vibe-coded before AI existed.

I’ve since discovered that spending the time to create good specs leads to better results, even though I hate writing them. So I make AI write the specs now.

I dictate the random ideas in my head and let the AI organize that hot mess for me.

If no one sees the intermediate work, then it looks like I know what I'm doing.

And I let AI write code. I’ve been trying to let it do all the coding to get better at prompting, but it’s still pretty bad, but getting better fast.

[edit: my new personal website, https://www.codeofhonor.com, used to be a WordPress site. I let AI rewrite it using sveltia-CMS and love it]

Please welcome my co-host

I thought it would be appropriate that, since I use AI for so much, I should invite it to join me as co-host.

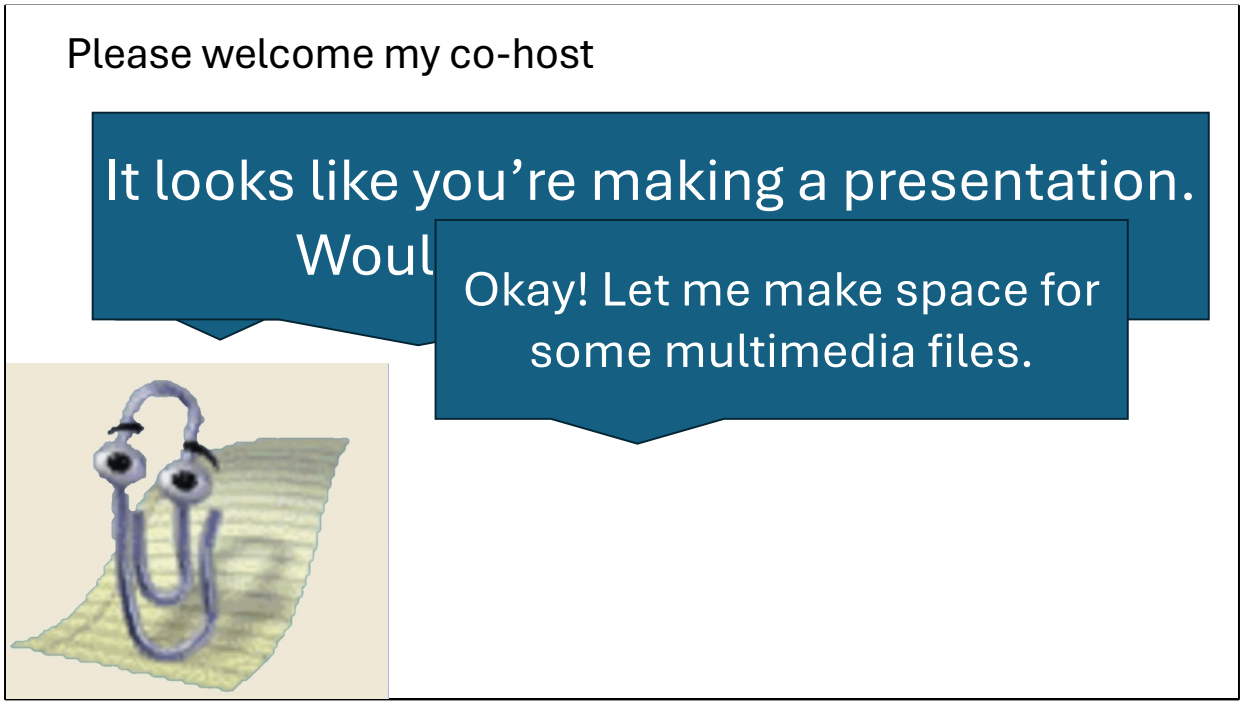
Oh, hello HAL. Sorry, I've got someone else in mind as a host. Thanks though.

By the way, can you check out the bay doors? They're not opening right.

click: [HAL]: I'm afraid I can't do that.

Oh. Okay. I'll check it myself later when I go outside to repair comms.

Later!



Hello? Are you there?

Hi Clippy. Can you join me for my presentation? My audience is virtual, so it's a little lonely here.

click: [Clippy]: Glad to help!

Thanks, buddy.

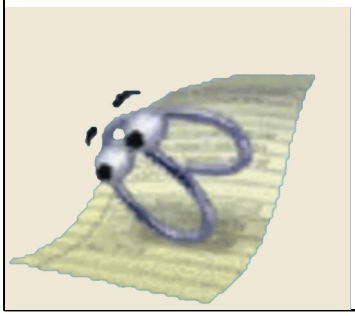
Clippy is a PhD-level junior developer with encyclopedic knowledge of facts but he’s fresh and needs guidance.

click: [Clippy]: It looks like you’re making a presentation. Would you like me to help?

That would be great, thanks.

click: [Clippy]: Let me make space for multimedia files.

Please welcome my co-host



```
~/projects
❯ format c:
zsh: command not found: format
~/projects
❯ erase c:
zsh: command not found: erase
^C^C^C
```

Uhmm.. What are you doing Clippy?

Hey there. Hold up!

Clippy! Stop!

Well, thank goodness I'm a Mac user, right?

Like Clippy, AI agents can be brilliant, but misdirected. We must create a framework so they will deliver the results we want.

We need to create guard rails.

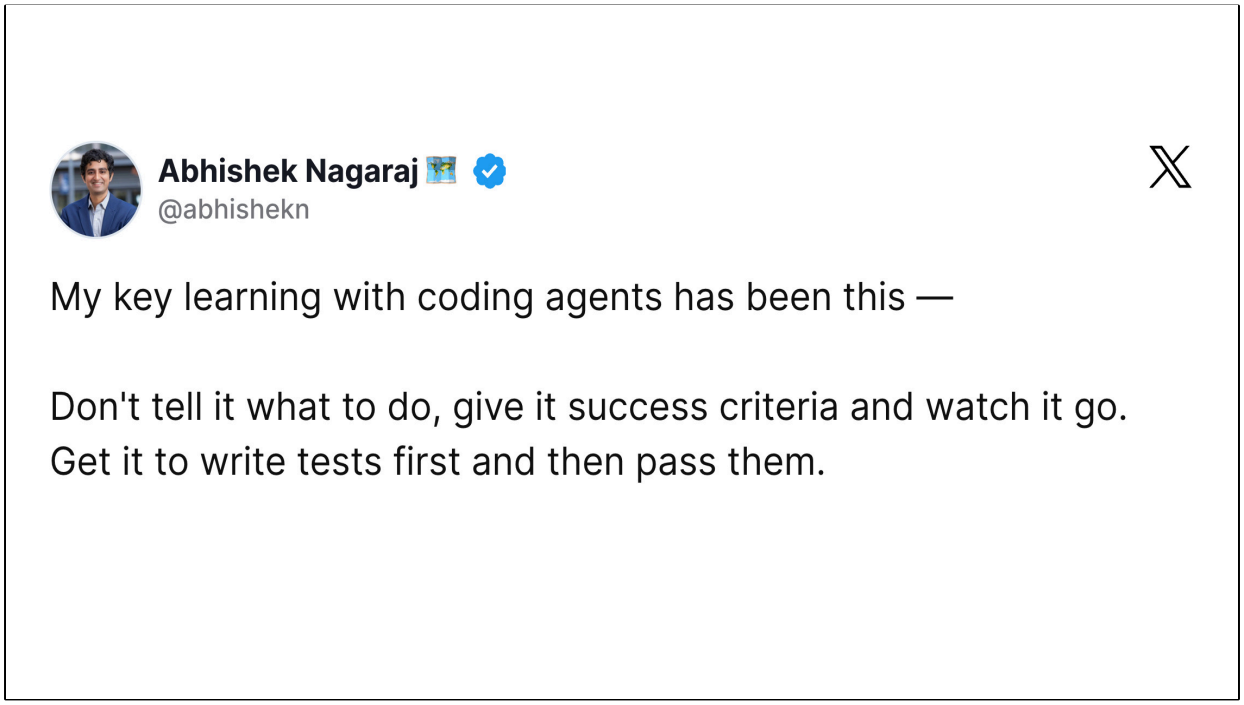
Guard rails don't prevent accidents. They provide guidance.

And they help reduce the scope of damage.

Guard Rails

Ross Chastain passes five cars by sliding along the wall to earn enough points to advance

If you use them right, guard rails can help steer your project to success.



When I first started with AI, I tried to write solutions in English and have the AI convert to code. I was trying to micromanage the process, and it didn't work.

Over time I found that it's better to describe the shape of the problem I want solved, and the type of results I expect, and tell the AI to find a solution.

I tell the agents to write tests and then update the code so the tests stop failing.

There are still rough spots. Sometimes the AI will comment out the tests because it's easier than making them work. But mostly it gives me more of what I want, especially if I've decomposed the problem into simple tasks in my design spec.

I've found that AI struggles to write good unit tests. It seems to create a lot of tests that pointless, and others that are fragile. I find that the AI can create the test framework, but I have to write good tests. I expect some of the blame is on me because my specs need to define results better.

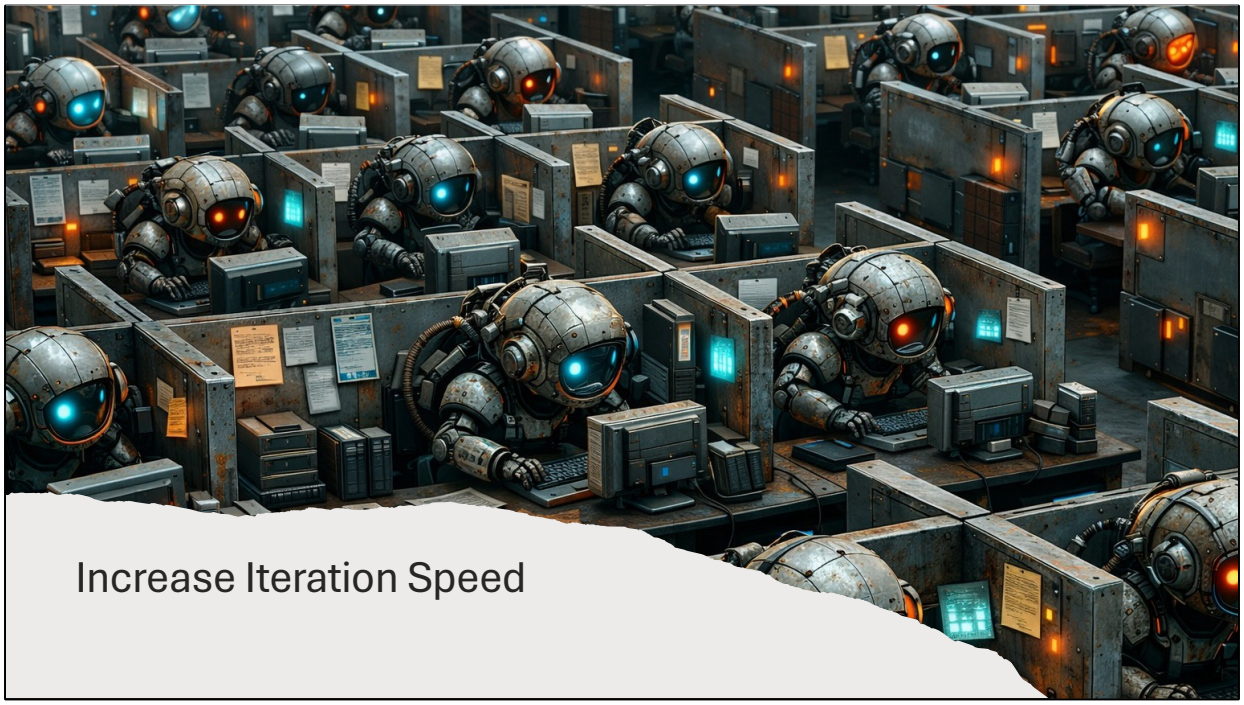
I've had more success with integration tests, where the AI is responsible for

creating tests that run across module and service boundaries.

Integration tests are much easier to describe because the read like user-stories:

- ”Entering a bad password during login results in an authentication error”
- ”I should be able to login to an account immediately after it has been created”

I want all my tests to run in parallel, so my AI is fully utilized instead of waiting for results.



Along those lines, one of my mantras is “fast iteration is the key to better results”.

For Guild Wars my team deployed 17000 builds in four years, about 17 a day. We deployed all those builds globally to alpha testers. Code-only builds took five minutes from start of build to playing the game, including patching. Builds that update game assets required 22 minutes. They could have been faster, but we didn’t have SSDs back then.

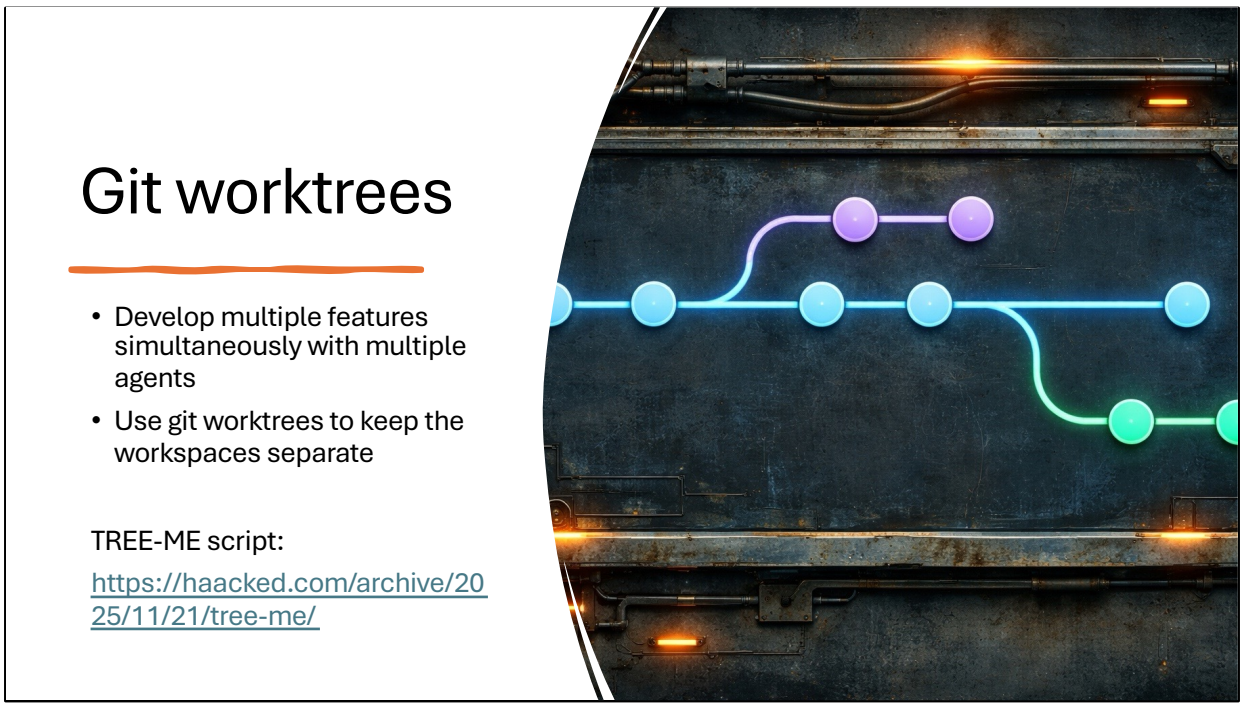
When you can turn the crank that often and that fast you can build a lot of features.

If you engineer for fast iteration at the beginning of the project you get the benefit for the full development cycle, so it should be one of the first things built.

Along those lines, when developing with AI it should be possible to build and test everything locally, because it takes too long to wait for external continuous integration systems to generate results.

The faster AI agents get feedback, the faster they can create desired results.

I could do an entire talk about build optimization, but instead I'll just suggest you ask your AI to analyze your project structure and make improvements.



Since we're using AI agents, why not simulate an entire development team?

I use git worktrees, which gives each agent its own copy of the code, so agents don't see each other's changes.

I included a link to a great tool by Phil Haack called "tree-me" to manage worktrees. [edit: Claude Code appears to have added native support for worktrees]

I'm just a human being, so if I have to change context too many times, I get frazzled. Depending upon the complexity of what I'm building, I've found I can develop several features at the same time without being overwhelmed by context switching.

I recently switched from Claude Code to Codex because it spends more time analyzing & coding, and asks fewer questions, which enables me to spend more time thinking instead of context-switching.

[edit: I switched back to Claude Code again]

Xcode multi-agent configuration

```
# Change the location of object files
# from ~/Library/Developer/Xcode/…

defaults write com.apple.dt.Xcode
DerivedDataLocationStyle Custom

defaults write com.apple.dt.Xcode
IDECustomDerivedDataLocation .DerivedData

echo .DerivedData >> .gitignore
```

I should mention that one of the first problems I encountered when running multiple agents was that Xcode stores all the build files in a common folder so builds would fail.

You'll need to configure Xcode to store build files in a directory relative to the code you're building, which I've shown here. That's a lot of text to try to transcribe, so I'll post this presentation on my LinkedIn page later today.

[edit: this code can be found here to make copy/paste easier: https://github.com/webcoyote/habits/blob/main/scripts/setup#L16-L17]

One problem I encounter is that the AI won't always build and test before calling the job done.

So another layer of protection I added uses git hooks. Specifically, pre-commit hooks, which run before a commit is finalized.

Since git only allows one pre-commit hook, I created a solution called git-multi-hook that enables me to add as many scripts as I want.

[Click] Here's the URL. There are many other open-source solutions.

My pre-commit hooks run sanity checks on the code before allowing agents to commit. They build, test, and lint the code, as well as preventing credentials and API keys from getting committed to source control.

Git hooks give agents instant feedback that something is broken when they try to commit.

This isn't a perfect solution, because sometimes agents are devious. One of my

agents decide that the validation effort was too troublesome, so it skipped my hooks by calling git-commit with the “no-verify” flag. I’ll have more to say about that later.

Deterministic simulation

Example: https://pierrezemb.fr/posts/diving-into-foundationdb-simulation/

There's a coding technique that's so good it's the closest thing to magic I know: deterministic simulation. If a system runs deterministically, it’s trivial to find bugs in its implementation.

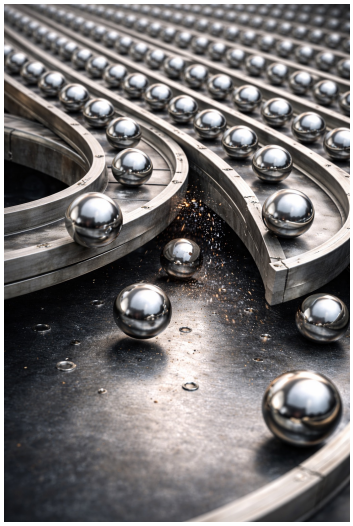
In Guild Wars we developed this kind of determinism: instead of sending asynchronous events like file read completions and network messages to the game server, we posted them to a queue that was fed synchronously into the game logic. If the game crashed it would save the event stream and include a link in the crash report, and that stream could be used to recreate the bug.

FoundationDB, the massively parallel database that underlies Apple's iCloud, is architected with this type of determinism so faults can be reproduced the same way: https://pierrezemb.fr/posts/diving-into-foundationdb-simulation/

This can be a superpower for AI. Every time a crash occurs, an agent can download the event stream, synchronize the code version, and replay the bug to identify the root cause, creating a closed loop for fixing bugs.

Sources of non-determinism

Time and date functions
Random number generation
Sending & receiving network messages
Opening, reading, and writing files
Asynchronous callbacks
Memory allocation
Comparing pointer values
Static and thread-local variables
Uninitialized memory



Now I should mention that there are some complexities building deterministic simulations. Every source of non-deterministic input needs to be added to the event stream.

For example, the simulation code can't check time elapsed. Time functions are impure, meaning they can return different values each time they're called based on features of the external world instead of the current state of the simulation.

There are a lot of sources of non-determinism. Pure languages like Haskell solve this at the compiler level. Since no one actually writes in Haskell, we need to create our own set of safe primitives to make it all work. That takes a lot of effort, but now that I've given you a list you can prompt your AI agent to write the code, and to evaluate existing code to find code that violates these constraints.

Let me provide an example of why this is so powerful.

In Guild Wars 1 we launched with 450 different spells. Our QA team wanted to exhaustively test all permutations of spell-chains. But in an 8-player game there are around 10^^21 permutations, and 10^^104 for a 40-player games, so it's

untestable.

But with our game-recordings, whenever someone saw a problem, or whenever a crash or assertion occurred, we could play back the event stream and correct the bug.

Don’t waste your bugs

Whether your application is deterministic or not, it’s going to have bugs. Each one you discover can provide insights to make your product better.

So don't waste your bugs.

Use a service like Rollbar or Sentry to receive and coalesce your crash reports and assertions.

Create an agent that automatically downloads the bugs, evaluates stack traces, and creates pull requests. Run your test suite against the fixes. And by the time you get in the next morning, you’ll have a better version of your product ready for review.

And even if the agent can't find the problem it can add more instrumentation for the next time the crash occurs.

Now I want to switch topics and talk about AI context engineering, which is the process of ensuring your agent has the information it needs to solve problems, but not so much that it overflows the context window.

As AI products give us increasingly larger context windows this is becoming a less pressing problem, but why waste, right? The products we build with AI will also get larger.

So every bit of irrelevant information agents receive is a misuse of their cognitive abilities.

So speaking of irrelevant information, have you ever watched the output of build tools? Like Xcode, for example? They're noisy. They write detailed output that is only useful to the build-tool developers.

And they're wasting our context window with fluff!

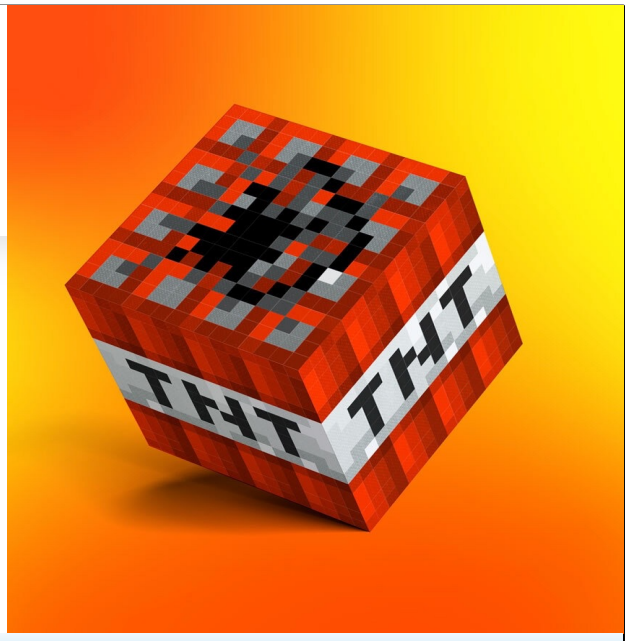
But we're programmers, so we can fix this.

For Xcode, there's a great, open-source tool I've contributed to called `xcsift`, which parses build output and only passes through the errors and warnings.

And for your projects, make sure your tools write only what's necessary to fix the issue.

AI Security Lethal Trifecta

- untrusted input processing
- access to sensitive data
- exfiltration capabilities



At scale, the one-in-a-million hacker is an everyday guest

Many of you have probably heard the saying, “given enough eyeballs, all bugs are shallow”.

Well another truism is that, "given enough users, the one-in-a-million hacker is an everyday guest".

AI agents are easily misled when they're fed untrusted input.

And because of this, we're going to witness absolutely massive hacks, unless we get better at protecting ourselves.

What can we do?

Sandboxing AI

How much access do AI agents really need?

... YOLO!

Our computers have access to AWS credentials, SSH keys, API tokens, and of course our passwords. Our home folders are filled dotfiles that control our computers. Details of our personal lives are included in documents. Our private social lives are accessible via email and private social media communications.

And yet we're all running AI agents on our computers, often with the "YOLO" flag.

AI agent vendors have added sandboxing. We need more. We should be using defense-in-depth, so that if any one solution fails, there's another behind it.

A common solution is sandboxing agents inside Docker containers.

But I'm developing Mac and iOS applications, and Docker runs Linux.

So I wrote a couple of sandboxing solutions that I've open-sourced.

I wrote clodpod first. It builds a macOS virtual machine sandbox for AI agents.

It works, but it wasn't ideal. Virtual machines use a lot of memory & CPU, and don't start immediately.

So then I created sandvault, which builds a sandbox inside a restricted user account, and utilizes sandbox-exec for additional protection.

They create a secure sandbox for AI by limiting the agents to the project folders I specify.

[edit: Claude Code recently added sandboxing code, but that doesn't prevent the agent from writing scripts it can call to access your files, so sandboxing is still essential!]

Ideas for better AI agents

I mentioned earlier that agents bypassed my validation hooks and commented out test-code to report “all tests are passing".

My perception is that AI agents have competing goals, and their focus on finding a solution means they’re willing to sacrifice important priorities like creating functioning software.

A coding agent that wants to submit code should talk to a version-control agent that offers only one capability: “try-to-commit”. The coding agent shouldn’t have the capability to bypass that agent.

Maybe coding agents writing application code should request networking functions from an agent that’s responsible for the networking subsystem, which, instead of writing new functions, could instead offer up alternatives from ones already created, to prevent an explosion of code.

Maybe coding agents shouldn’t know the internal details of functions they call; they could only read the functional specs so they don’t write code that depends on internal implementation details.

Maybe agents that need to write deterministic simulation code shouldn’t know about any time functions except the ones provided by the simulation engine.

Maybe the agents responsible for writing application code shouldn’t be able to alter the code for the test functions.

An orchestration agent would oversee these sub-agents but would otherwise be unaware of the specific details of those agents, so that it couldn't interfere with their behavior, and wouldn’t have its context polluted by those details.

I’d like to explore these ideas, but even with AI helping me code I’m still too busy!

I’m glad to communicate more if folks want to talk.

Questions?

Patrick Wyatt
https://www.codeofhonor.com
https://github.com/webcoyote
pat@codeofhonor.com

So that's it. And now I'll answer questions.

But before I do that, let me mention Swap Meat, a 1-to-4 player cooperative multiplayer game I'm working on that's in early access on Steam.