honeycomb.io

What Handcrafted Servers Taught Us About Handcrafted Code

Charity Majors

CTO @ honeycomb.io

@mipsytipsy, @charity.wtf

Me, last year:

“Call me old fashioned, but there is no test suite on earth that could make me trust a new chunk of code more than I trust the code that’s been running in production for two years. That’s how trust works; it takes time.

The only way to build confidence in your code is by exposing it to reality in a controlled manner, then iterating on what is known and stable.”

✨ https://www.honeycomb.io/blog/disposable-code-is-here-to-stay ✨

This is not a law of software development.

These are defining characteristics of durable software.

The cost of software has always been defined by the cost of its maintenance.

✨ https://www.honeycomb.io/blog/disposable-code-is-here-to-stay ✨

Durable code: to **maintain**, apply a series of small diffs to a known good baseline

Disposable code: **cheap because** you don’t even try to maintain it

Until recently, we were obsessed with code readability and understandability

This made sense, because elegance was a reasonable proxy for maintainability.

And writing code was time-consuming and hard.

Now it is *not* time-consuming and hard.

A lot of people are arguing over whether or not we should be reading the code and spending all our time on code review.

I want to look a little further out.

The durable model of software development is agonizingly expensive.

Code in production is more than just code. It is the fossilized record of the system and its history — the only record that exists.

Our primary asset is not the codebase, but the system, and its ability to keep working.

The “system” is the unique intersection of code, infra, users, data, and time.

The most important parts of our system have never been written down.

We find them when we break them, mostly.

To bring down the cost of software, we are going to have to unbundle developer intent and commitments from the code, and put them somewhere else.

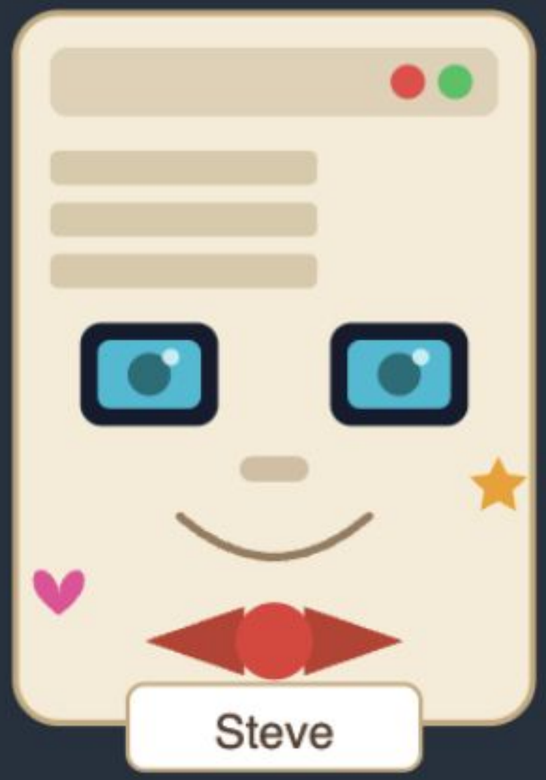
Does this all sound familiar? It should.

This has all happened before.

My first job title was

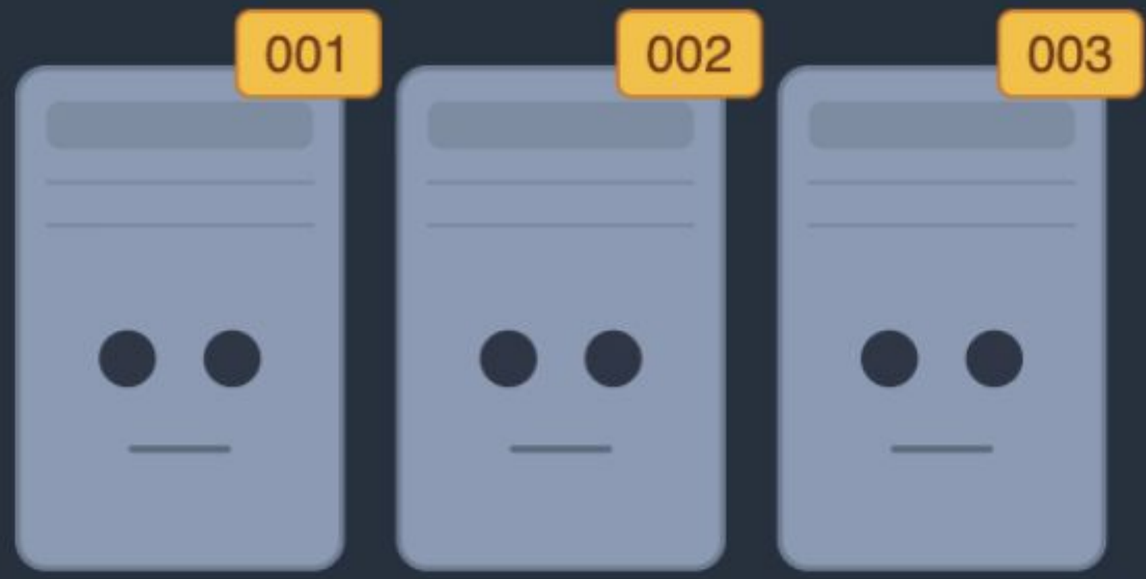
“System Administrator”

Server Pet



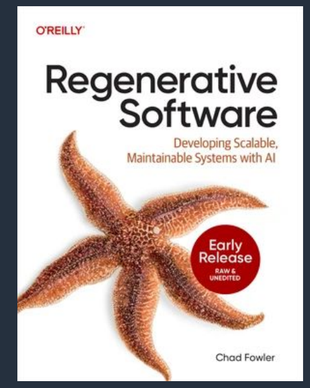
unique · named · beloved

Server Cattle



identical · numbered · replaceable

Chad Fowler, “Regenerative Software”



Infrastructure (n):

“The code you have to run in order to run the code you want to run”

What did we learn from handcrafted infrastructure?

1. Mutability creates drift, and is the sworn enemy of understanding.

What did we learn from handcrafted infrastructure?

2. Extract before you validate. You cannot regenerate what you have not yet defined.

What did we learn from handcrafted infrastructure?

3. When failure is rare, it is *terrifying*; when failure is ordinary, it is boring.

Hand-writing code isn’t that different from hand-installing packages

Editing Code Is Mutation

When you edit code in place, you're doing the software equivalent of SSHing into a production server and tweaking a config file.

You're assuming you understand the full state of the system. You're assuming the change is local, that history doesn't matter, that side effects are predictable.

Chad Fowler, “Regenerative Software”

If you can regenerate the code while keeping all your commitments to users, maintainability becomes a property of the **system**—not the code.

✨Code becomes cache✨

Chad Fowler, “The Deletion Test”,
https://aicoding.leaflet.pub/3md5ftetaes2e

“The Deletion Test”

Imagine deleting the entire implementation.

Not refactoring it.

Not archiving it.

Not putting it behind a feature flag.

Deleting it.

```
rm -rf src/
```

If that thought makes your stomach drop, pay attention. That reaction is telling you something important.

It’s not telling you that you’re reckless.

It’s not telling you that you lack discipline.

It’s telling you that you don’t know what would survive.

“We can’t just throw the code away!”

- We don’t know exactly what behavior is required.
- We don’t know which failures are unacceptable.
- We don’t know what invariants must always hold.
- We don’t know how to tell if a new version is correct.
- We don’t know which bugs are intentional fixes for forgotten edge cases.

From “How do we write safer code?” to “What must exist so that code can be replaced safely?”

> The question stops being “How do we write safer code?”
>
> It becomes “What must exist so that code can be replaced safely?”
>
> That question is harder. It requires deciding what truly matters and making it explicit, testable, and durable. It requires relocating rigor out of the implementation and into the system around it.

Chad Fowler, “The Deletion Test”

Software has a gradient of risk

- There are many use cases for disposable code, or durable code with a high tolerance for variance
- And many with low tolerance for variance.

The only structure we get in software is the structure we give ourselves.

- The higher in the stack, the more we are governed by logic, language and metaphor
- This DOES NOT mean structure does not matter.
- It means we must enforce it ourselves.

Pace layers

Fast layers experiment, slow layers stabilize. Tension between them is healthy, confusing them is destructive.

> When fast layers are over-constrained by slow ones, innovation dies. Every UI change requires a committee.
>
> When slow layers are eroded by fast ones, trust dies. The system becomes a house of cards that looks fine until it doesn't.

Chad Fowler, “Pace Layers and AI Integration”

No one misses the bad old days of handcrafted servers, perl scripts, and snowflake cron jobs.

No one is going to miss the bad old days of handcrafted code, capture/replay and strangler figs either.

In the end, the sysadmins mostly got on board.
They built the systems that replaced them.

Feelings are normal, and we should make space for them.

But.

The “manager” vs “practitioner” chasm...
Or the “pro-AI vs “hates-AI” chasm...
Please be careful.
It is rarely that simple.

People dragging their feet **often have cause**. Give them a chance. Listen. Before you force it down their throats: **Be sure you are right**.

See everything. Solve anything.

honeycomb.io

Palette

gray50 | Fog	sky50		red50	gold50	green50	
gray100	sky100		red100	gold100	green100	
gray200	sky200		red200	gold200	green200	
gray300	sky300		red300	gold300	green300	purple100
gray400	sky400		red400	gold400	green400	purple200
gray500	sky500	blue500	red50	gold500 | Honey	green500	purple500
gray600	sky600	blue600	red50	gold600	Green600 | Lime	purple600
gray700	sky700 | Pacific | Cobalt	blue700	red50	gold700	green700	purple700
gray800	sky800	blue800	red50	gold800	green800	purple800 | Indigo
gray900	sky900	blue900	red50	gold900 | Tango	green900	purple900
				gold1000	green1000	

Slate - Text/Bkgs	Denim

Eyebrow Subtitle

2-points layout dark background

200%

Add unlimited fields to your data. Add unlimited users.

$5.5m

Add unlimited fields to your data. Add unlimited users.

Eyebrow Subtitle

3-points layout dark background

200%

Add unlimited fields to your data. Add unlimited users. Access unlimited services. Don’t be penalized for your curiosity.

200%

Add unlimited fields to your data. Add unlimited users. Access unlimited services. Don’t be penalized for your curiosity.

200%

Add unlimited fields to your data. Add unlimited users. Access unlimited services. Don’t be penalized for your curiosity.

Eyebrow Subtitle

4-points layout dark background

200%

Add unlimited fields to your data. Add unlimited users. Access unlimited services. Don’t be penalized for your curiosity.

200%

Add unlimited fields to your data. Add unlimited users. Access unlimited services. Don’t be penalized for your curiosity.

200%

Add unlimited fields to your data. Add unlimited users. Access unlimited services. Don’t be penalized for your curiosity.

200%

Add unlimited fields to your data. Add unlimited users. Access unlimited services. Don’t be penalized for your curiosity.

Eyebrow Subtitle

6-points layout dark background

Point one

Add unlimited fields to your data. Add unlimited users.

Point two

Add unlimited fields to your data. Add unlimited users.

Point three

Add unlimited fields to your data. Add unlimited users.

Point four

Add unlimited fields to your data. Add unlimited users.

Point five

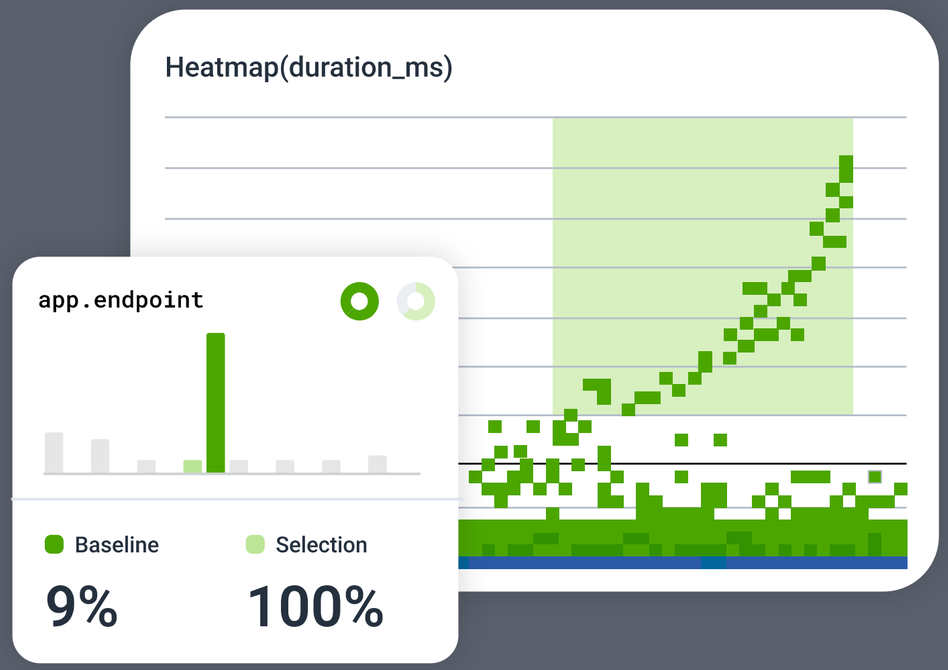
Add unlimited fields to your data. Add unlimited users.

Point six

Add unlimited fields to your data. Add unlimited users.

Server Pets

Server Cattle



Eyebrow Subtitle

Large card layout dark

Add unlimited fields to your data. Add unlimited users. Access unlimited services. Don’t be penalized for your curiosity.

Add unlimited fields to your data. Add unlimited users. Access unlimited services. Don’t be penalized for your curiosity.

Add unlimited fields to your data. Add unlimited users. Access unlimited services. Don’t be penalized for your curiosity.

honeycomb.io

Card Title Screen Dark

March 2025