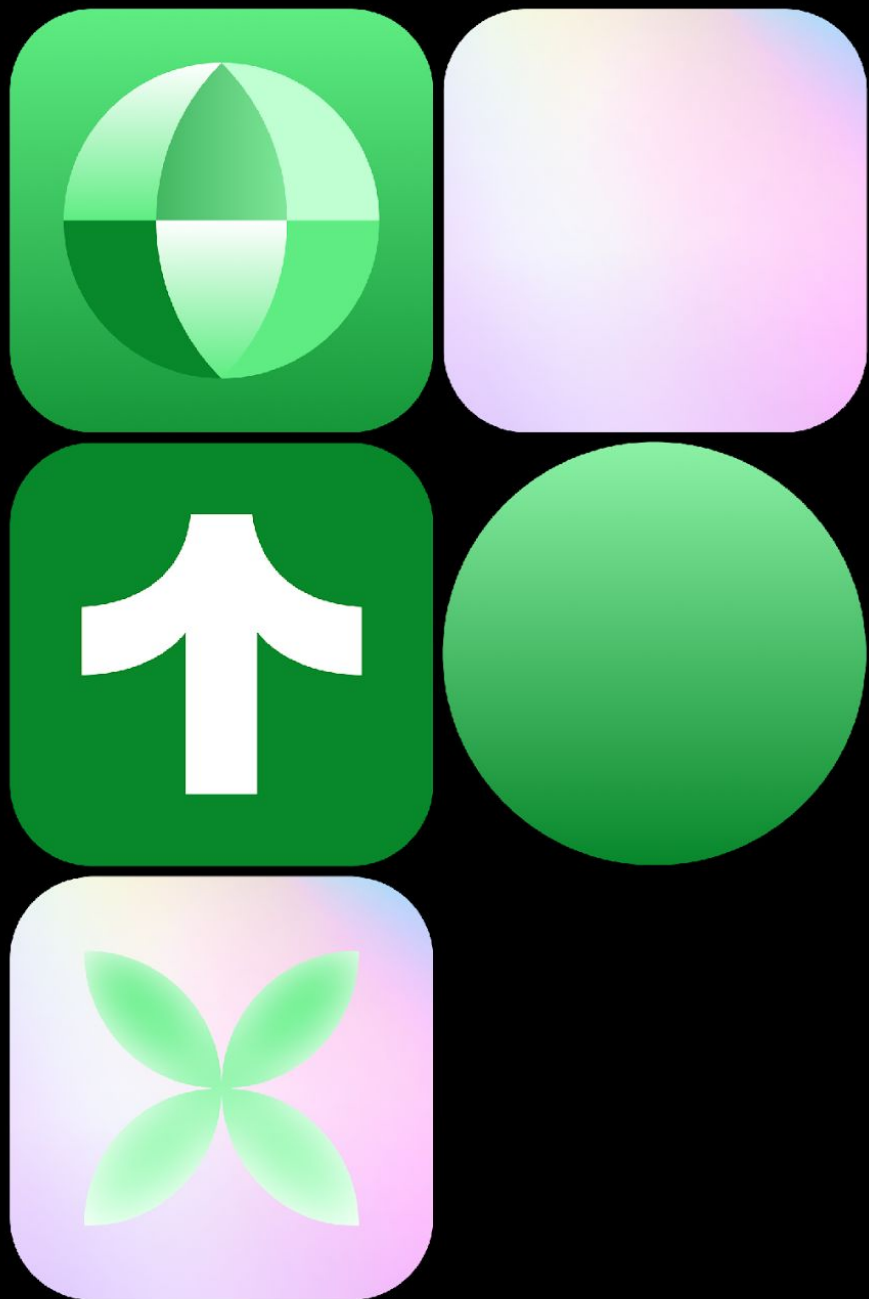




Building GitHub with Copilot

How our engineers multiply
their impact and reliably
ship features



Bassem Dghaidi

Senior Software Engineer / GitHub

Agenda

- 01 **GitHub's architecture** (an overview)
- 02 **GitHub's SDLC^[1]** (how we ship changes)
- 03 **Where AI fits into our SDLC**
- 04 **How I get the most out of Copilot**
- 05 **Q&A**

^[1] SDLC: Software Development Lifecycle



01. GitHub's architecture

(a quick overview)

Inbound
traffic



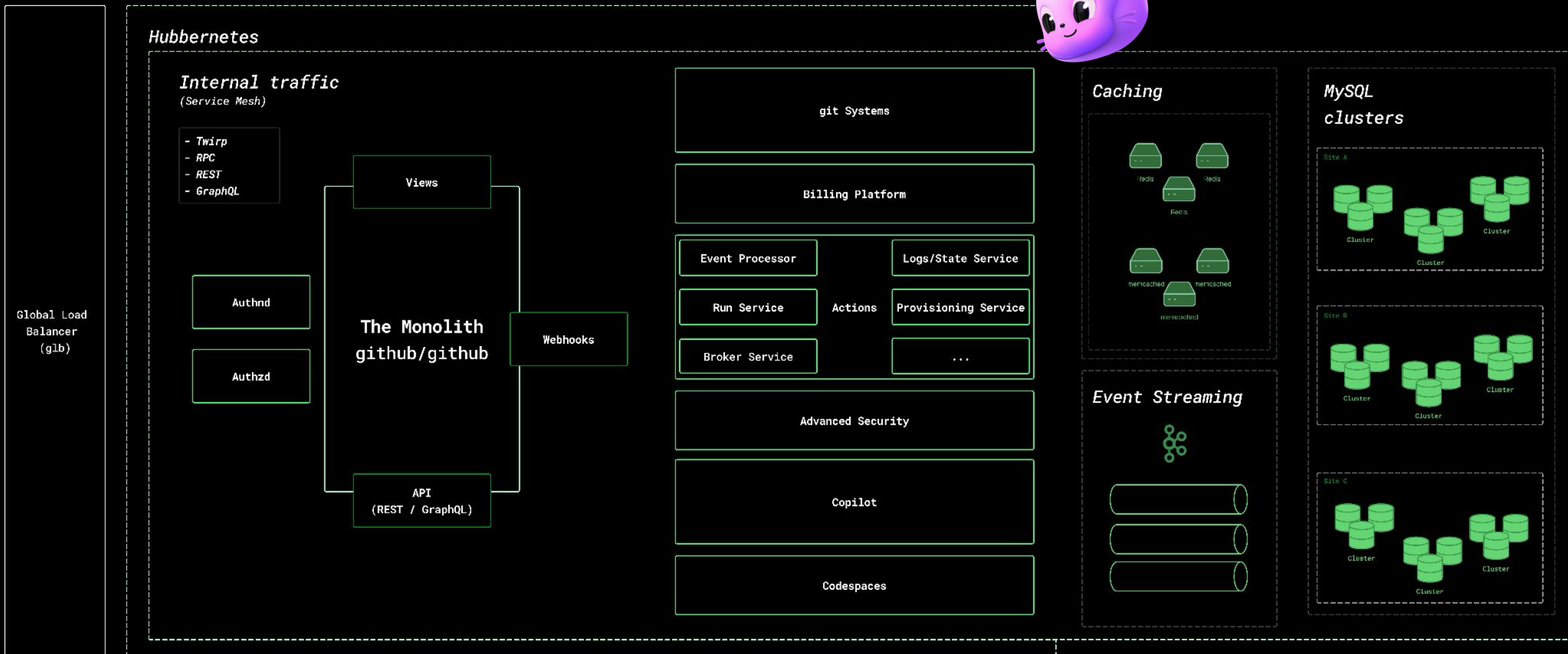
Outbound
traffic



The high level map of GitHub

Block diagram of some of the the main services powering GitHub.

* This is not an accurate representation of the integrations and traffic flows.



The high level map of GitHub

Block diagram of some of the the main services powering GitHub.

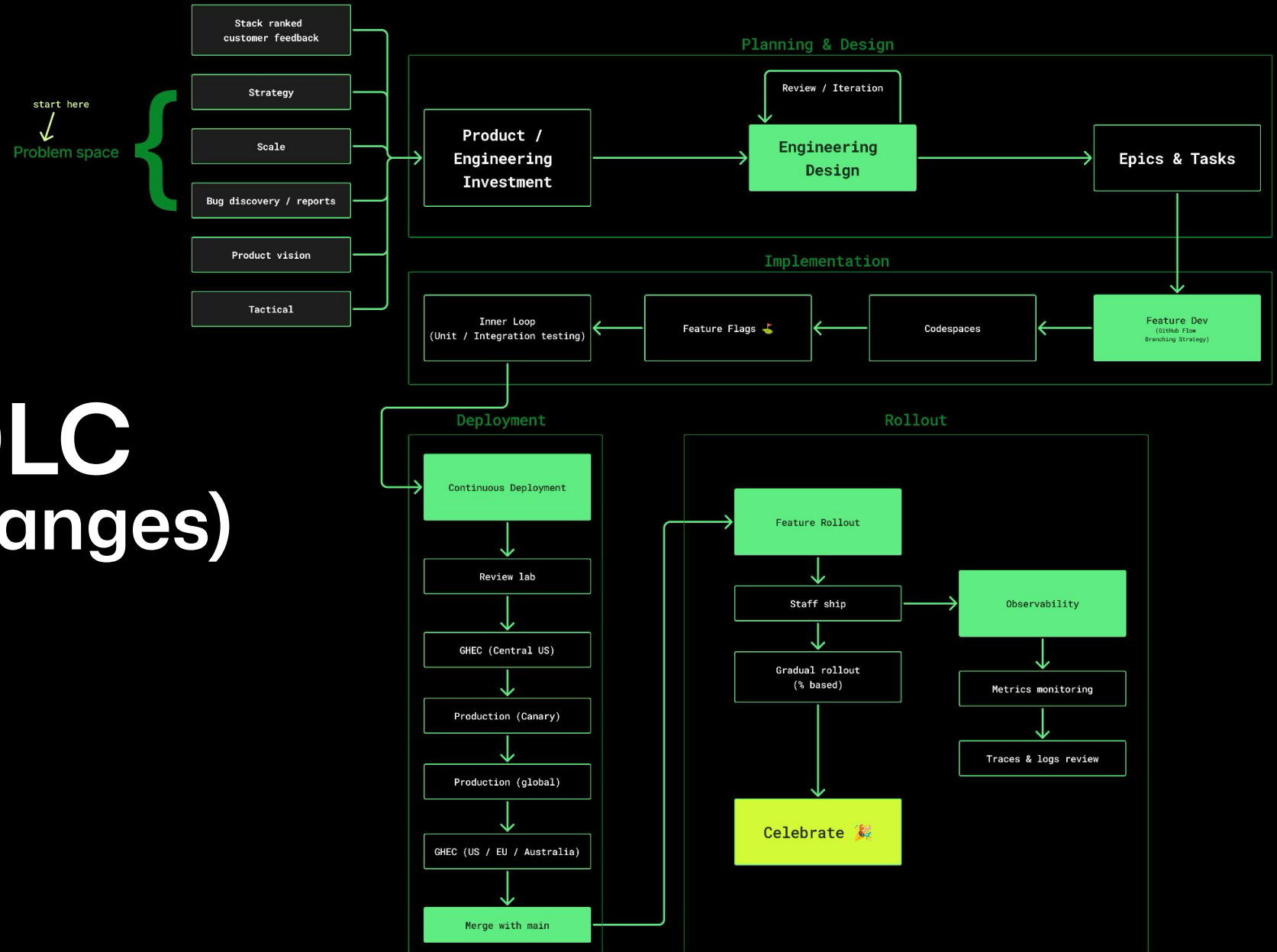
** This is not an accurate representation of the integrations and traffic flows.*





02. GitHub's SDLC

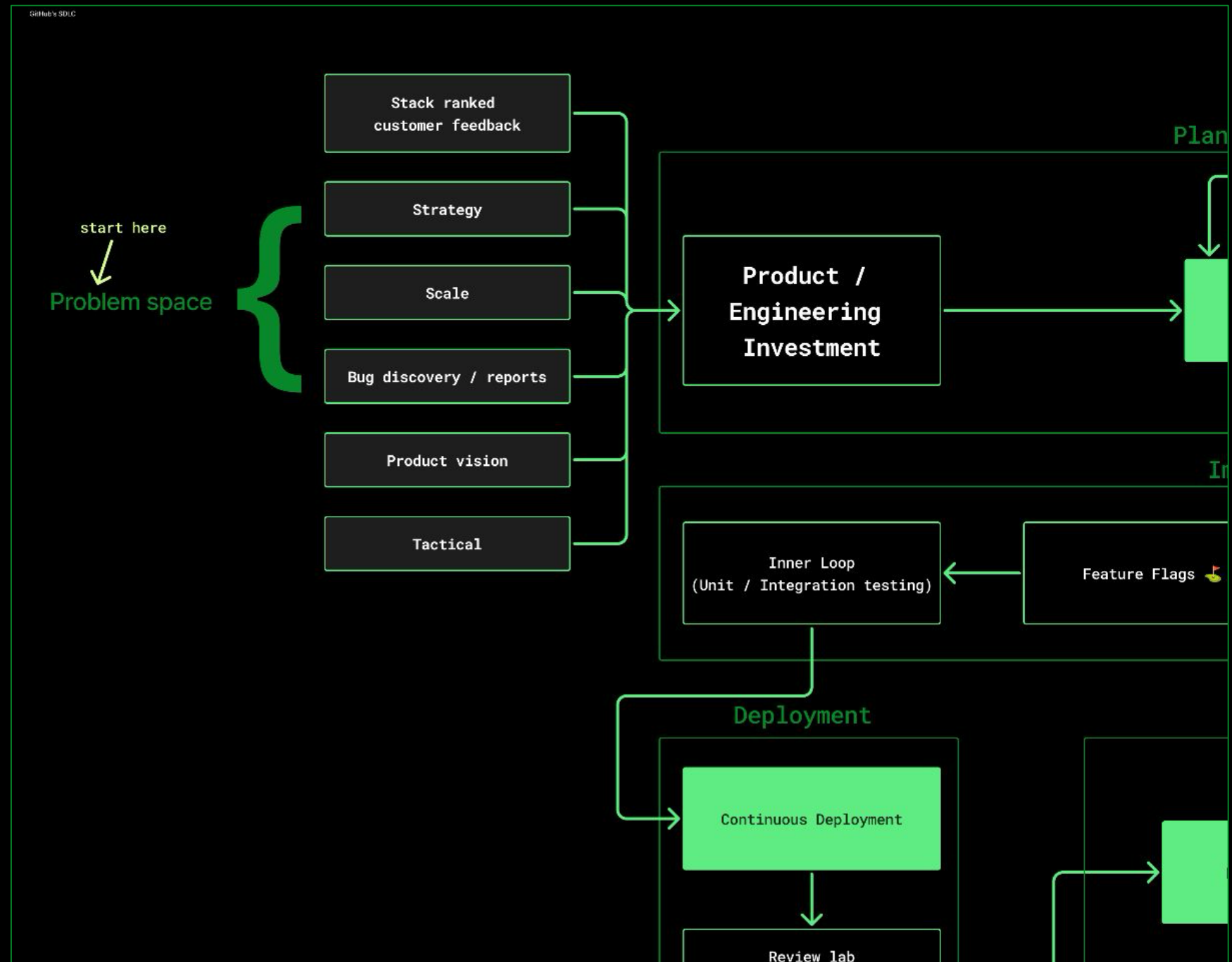
(how we ship changes)



GitHub's SDLC (how we ship changes)

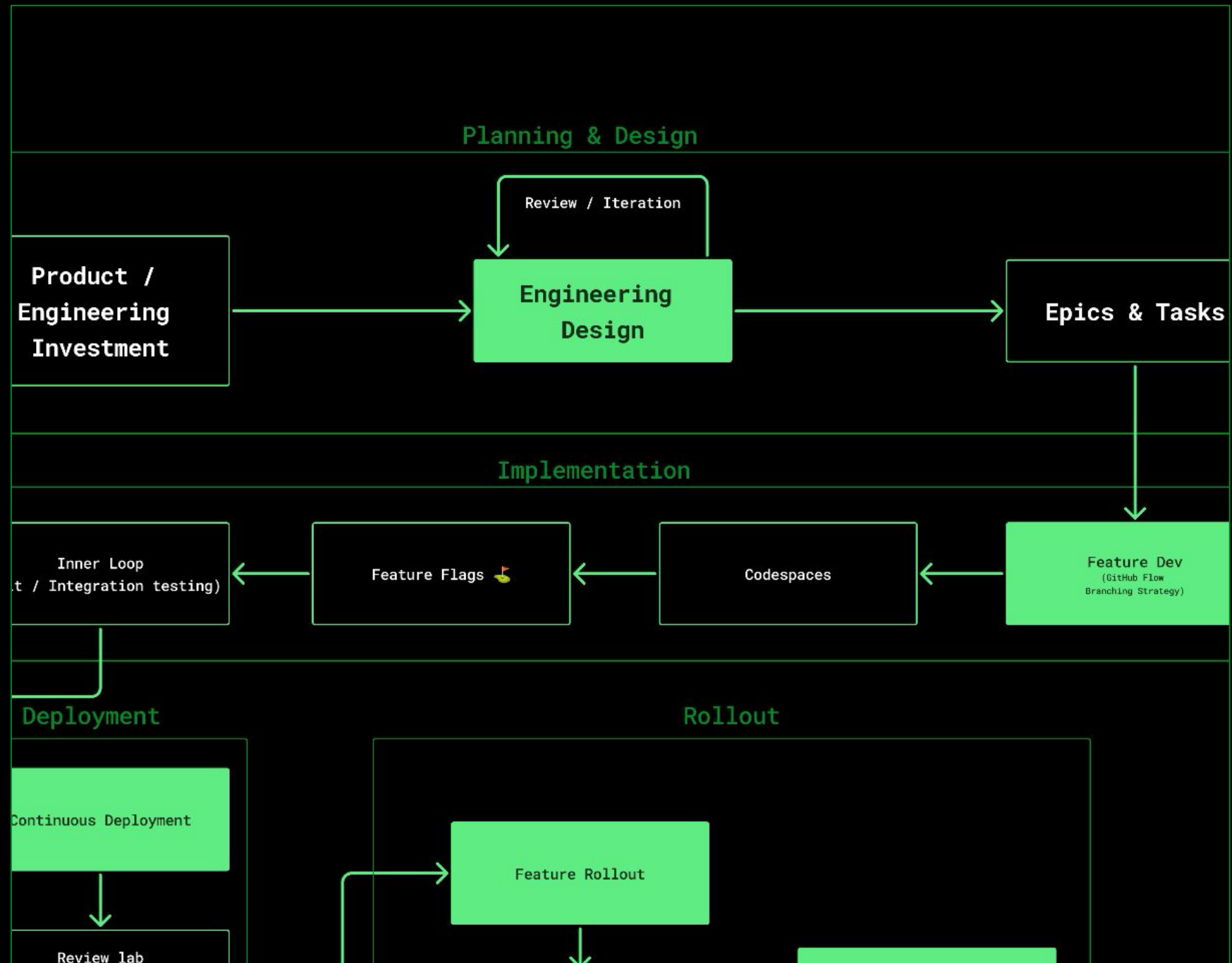
Pre-design phase

The problem space is not stack ranked. Priorities vary.



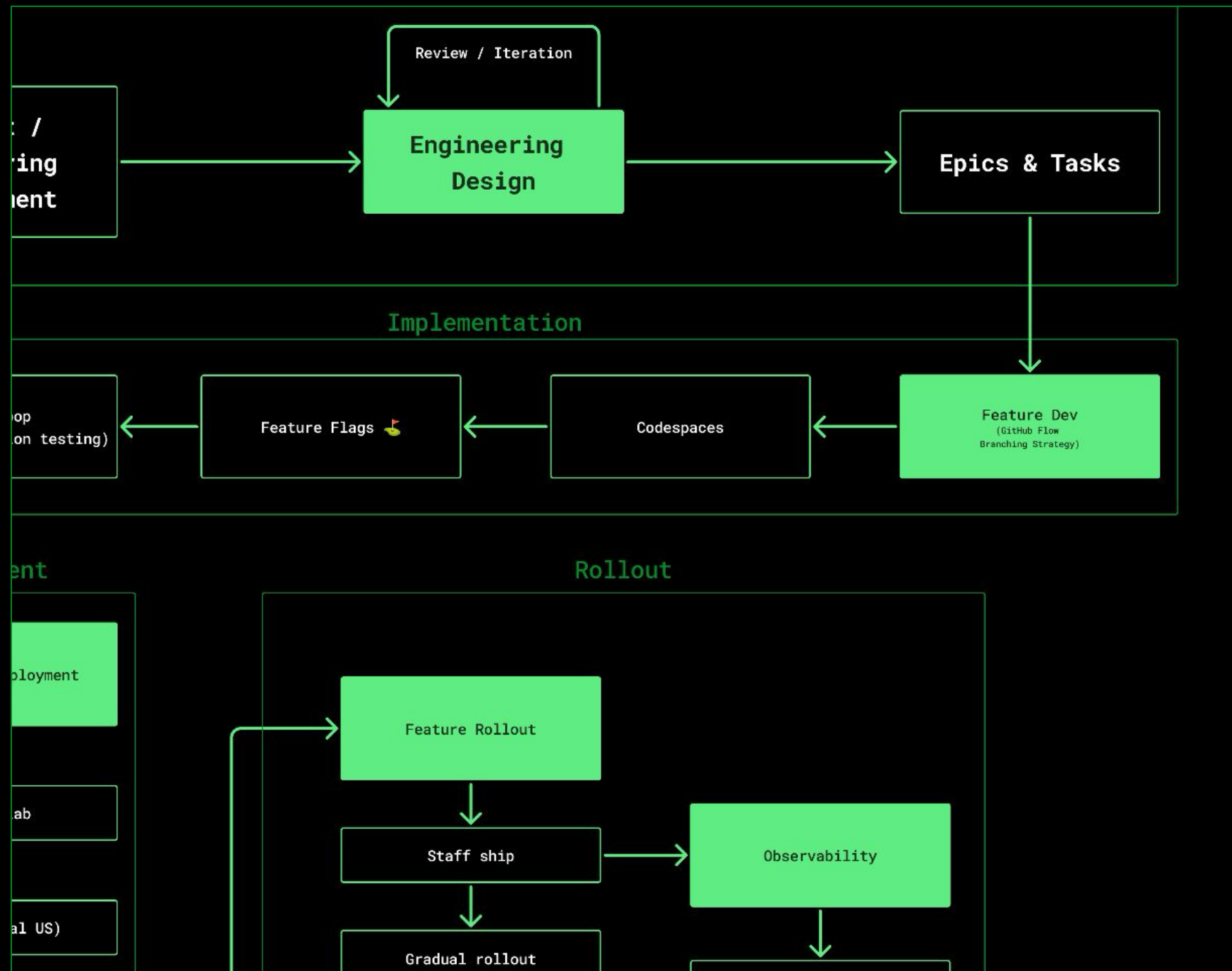
Design phase

Choice of EDR vs ADR depends on the scale of the problem



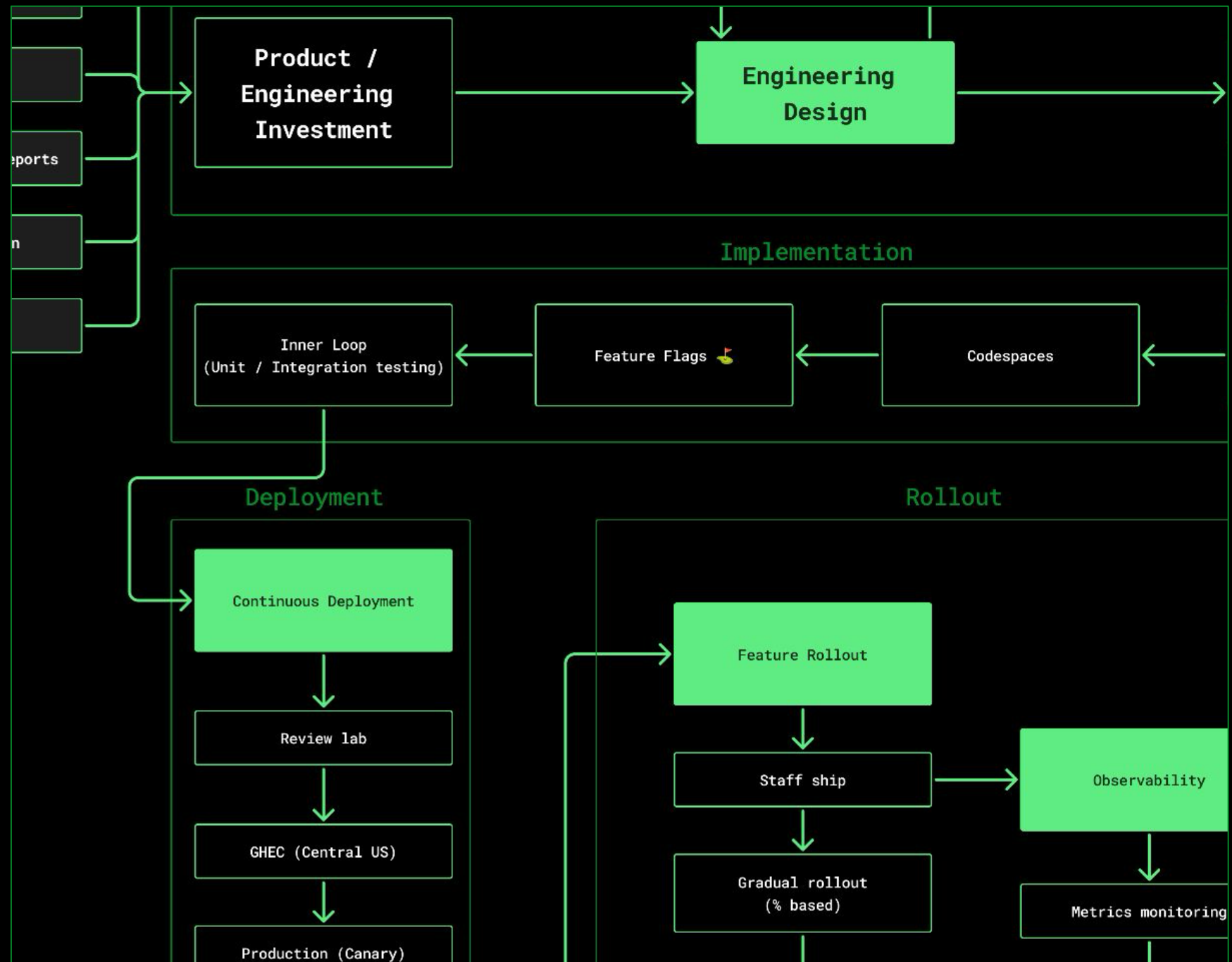
“Agile”

We don't force a prescriptive “agile” process. Teams choose what's best for them.



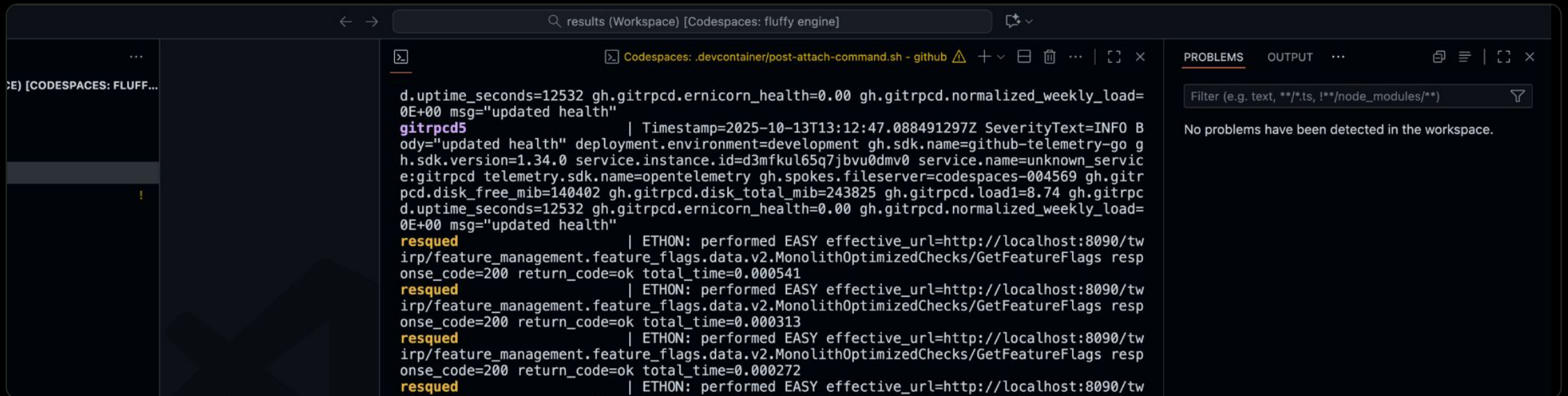
Ephemeral dev environments

- Always ready to ship.
- **Everything** goes behind feature flags.



We can run GitHub and all related services end to end in 1 chunky Codespace

Our inner loop supports cold starts!



The screenshot displays a VS Code Codespace environment. The terminal window shows a series of JSON logs from the 'github' service, including health status updates and feature flag checks. The PROBLEMS panel on the right indicates that no problems have been detected in the workspace.

```
d.uptime_seconds=12532 gh.gitrpcd.ernicorn_health=0.00 gh.gitrpcd.normalized_weekly_load=0E+00 msg="updated health"
gitrpcd5 | Timestamp=2025-10-13T13:12:47.088491297Z SeverityText=INFO Body="updated health" deployment.environment=development gh.sdk.name=github-telemetry-go gh.sdk.version=1.34.0 service.instance.id=d3mfkul65q7jbvu0dmv0 service.name=unknown_service:gitrpcd telemetry.sdk.name=opentelemetry gh.spokes.fileserver=codespaces-004569 gh.gitrpcd.disk_free_mib=140402 gh.gitrpcd.disk_total_mib=243825 gh.gitrpcd.load1=8.74 gh.gitrpcd.uptime_seconds=12532 gh.gitrpcd.ernicorn_health=0.00 gh.gitrpcd.normalized_weekly_load=0E+00 msg="updated health"
resqued | ETHON: performed EASY effective_url=http://localhost:8090/twirp/feature_management.feature_flags.data.v2.MonolithOptimizedChecks/GetFeatureFlags response_code=200 return_code=ok total_time=0.000541
resqued | ETHON: performed EASY effective_url=http://localhost:8090/twirp/feature_management.feature_flags.data.v2.MonolithOptimizedChecks/GetFeatureFlags response_code=200 return_code=ok total_time=0.000313
resqued | ETHON: performed EASY effective_url=http://localhost:8090/twirp/feature_management.feature_flags.data.v2.MonolithOptimizedChecks/GetFeatureFlags response_code=200 return_code=ok total_time=0.000272
resqued | ETHON: performed EASY effective_url=http://localhost:8090/twirp/feature_management.feature_flags.data.v2.MonolithOptimizedChecks/GetFeatureFlags response_code=200 return_code=ok total_time=0.000272
```

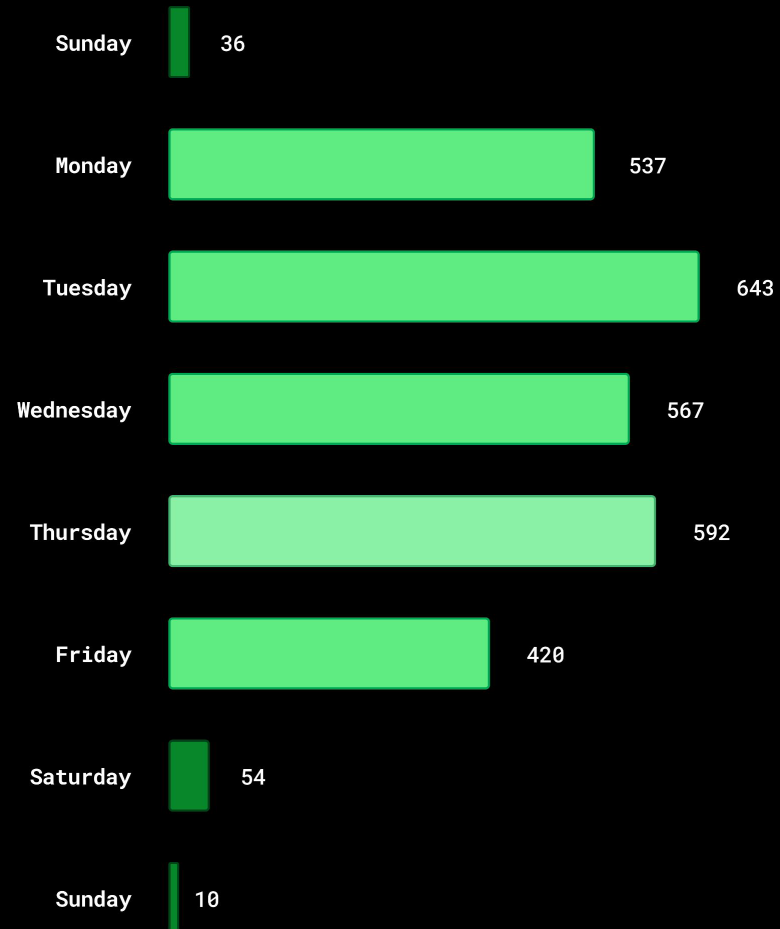
PROBLEMS OUTPUT ...

Filter (e.g. text, **/*.ts, !**/node_modules/**)

No problems have been detected in the workspace.

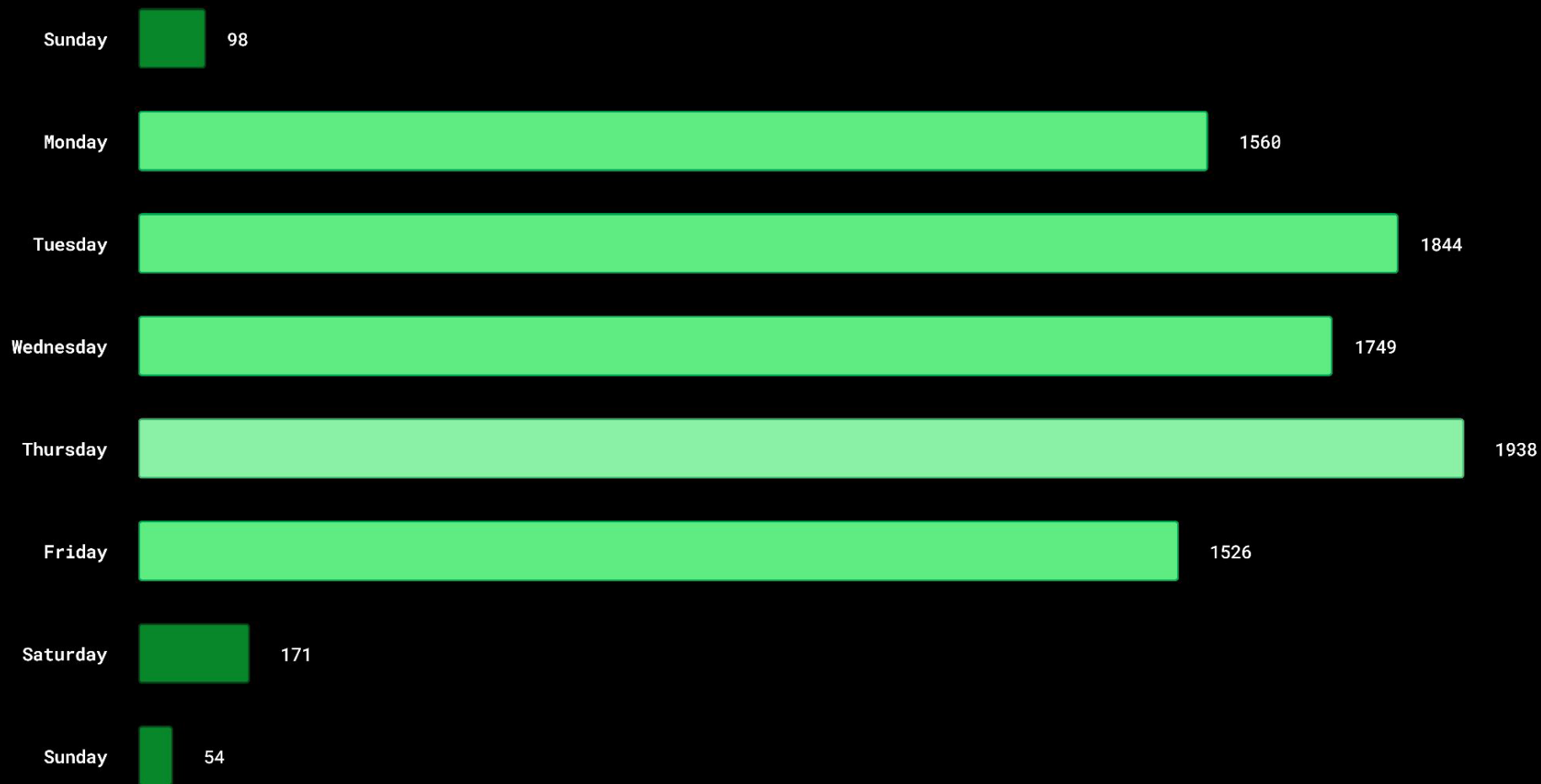
Within the GitHub organization

Codespaces created per week



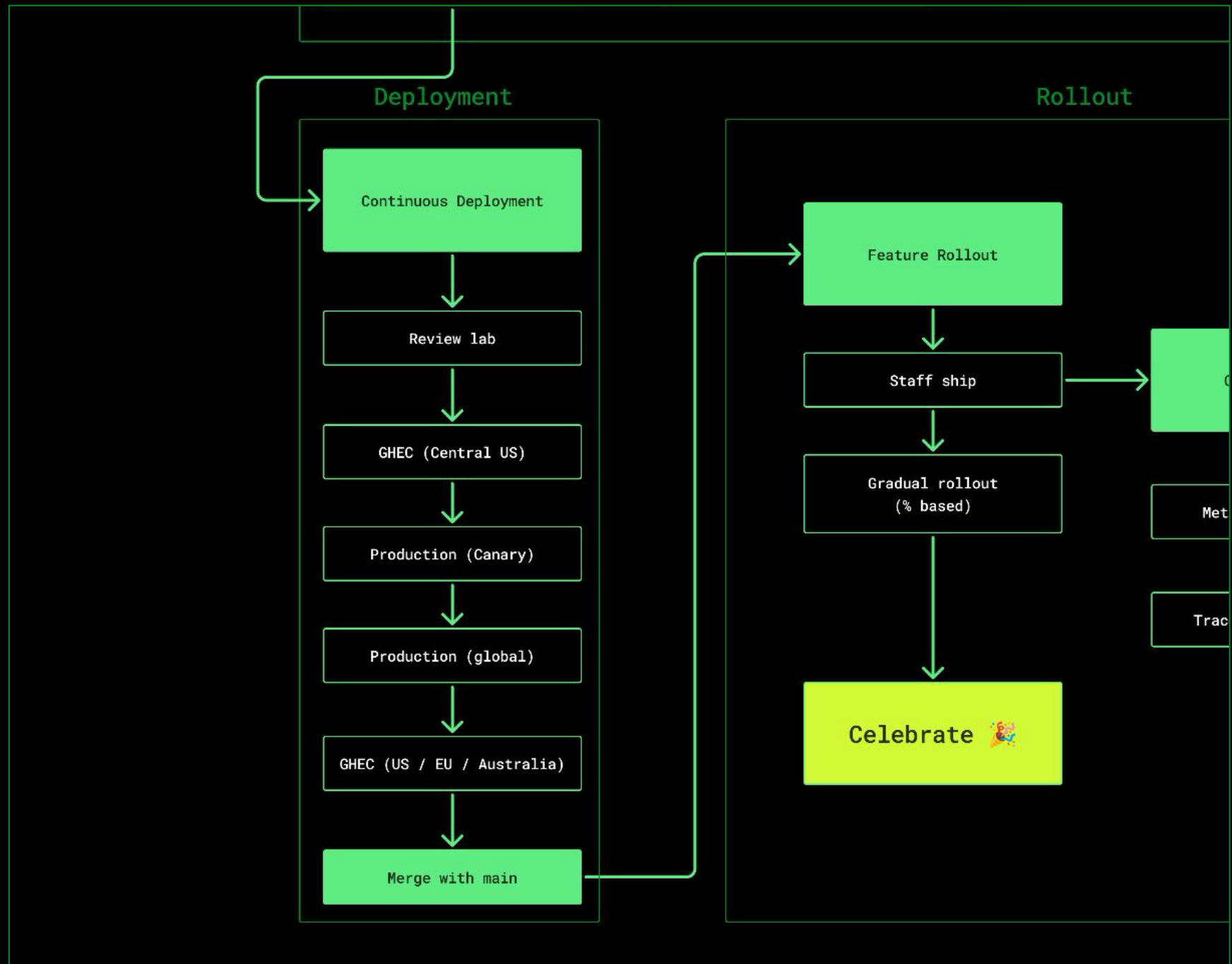
Within the GitHub organization

Codespaces started per week



Continuous Deployment

Shipping hundreds of changes per day via a multi-staged, multi-region pipeline.



Append the change to the merge queue

Press the big green button

 Open

chore: requirement specific Splunk index for all workloads #405947
levenleven wants to merge 3 commits into [master](#) from [specific-splunk](#)

 **This branch has not been deployed**
No deployments

  **Changes approved**
1 approving review by reviewers with write access.

 1 approval >
 1 pending review >

 **All checks have passed**
30 skipped, 239 successful checks

 **No conflicts with base branch**
Merging can be performed automatically.

Merge when ready ▾ This repository uses the [merge queue](#) for all merges into

 **Add a comment**

Write

Preview

Disaster recovery?

Oh no! GitHub is down...

Mirror deploys are tested every weekend, and yes, we can still deploy even if Slack is down.

Messages Files Bookmarks Pins Untitled +

12:35 AM Today ▾

.qfg

Hubot APP 12:36 AM
production is available now.
<https://github.com/github/github/queue> provides a better web UI for the queue.
Queue for github: empty

12:36 AM

.lock github in production conducting mirror testing

Hubot APP 12:36 AM
Okay, I've blocked github in production because conducting mirror testing

12:45 AM

.deploy github/mirror-test-09be03d8-2e92-410f-b284-221001f7c875 to prod

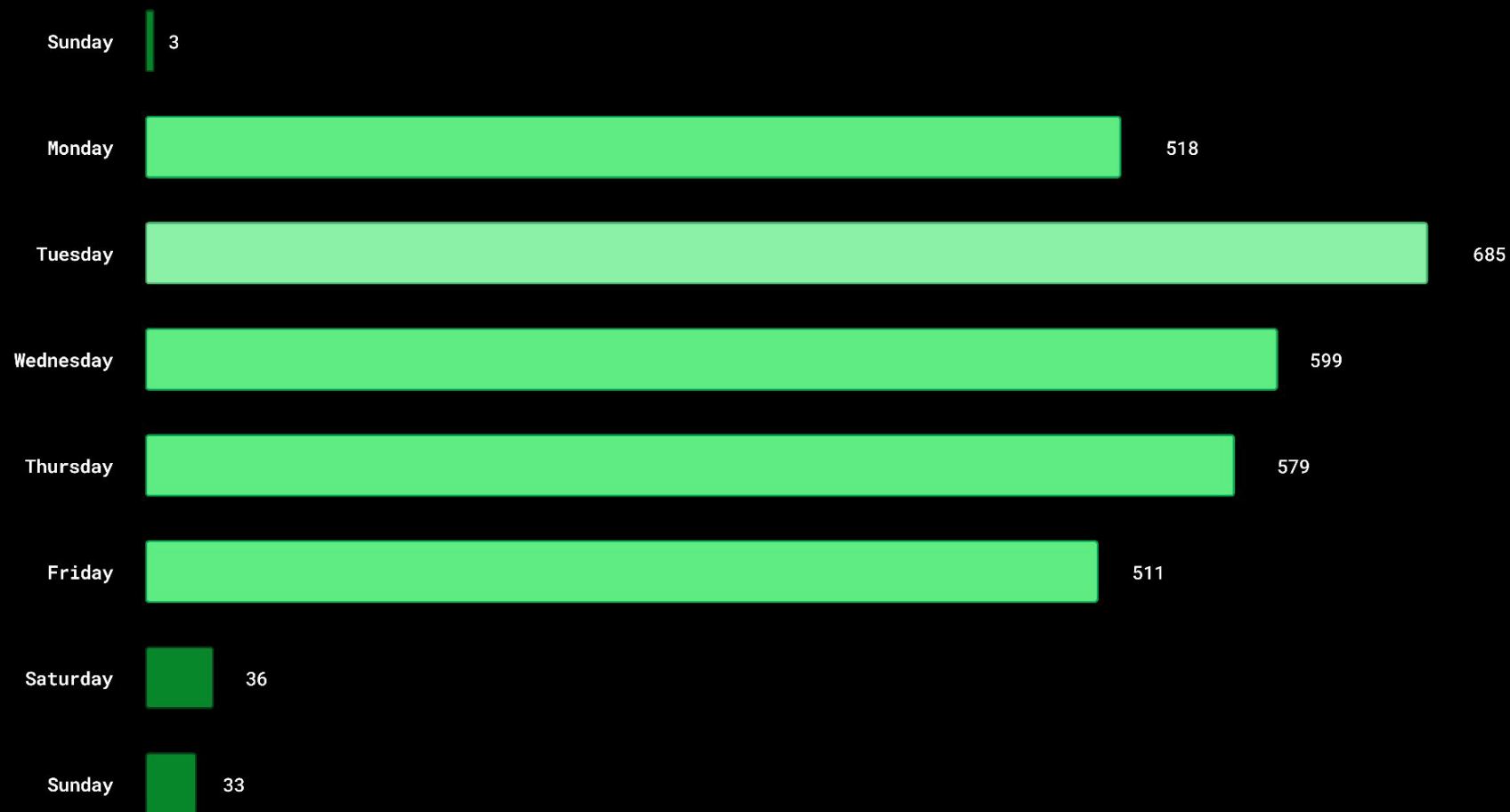
4 replies Last reply today at 12:56 AM

Hubot APP 12:45 AM
replied to a thread: [.deploy github/mirror-test-09be03d8-2e92-410f-b284-221001f7c875 to prod](#)
 [Deployment of github/mirror-test-09be03d8-2e92-410f-b284-221001f7c875 to prod](#)
been queued to run in the background. You'll receive updates via slack notifications.
`id: 35477860 sqid: RCewEfuQ`
[View newer replies](#)

Hubot APP 12:56 AM
replied to a thread: [.deploy github/mirror-test-09be03d8-2e92-410f-b284-221001f7c875 to prod](#)
production [deployment](#) of github/mirror-test-09be03d8-2e92-410f-b284-221001f7c875 is done:
 [All partitions](#) succeeded. 16 partitions (678.26s)

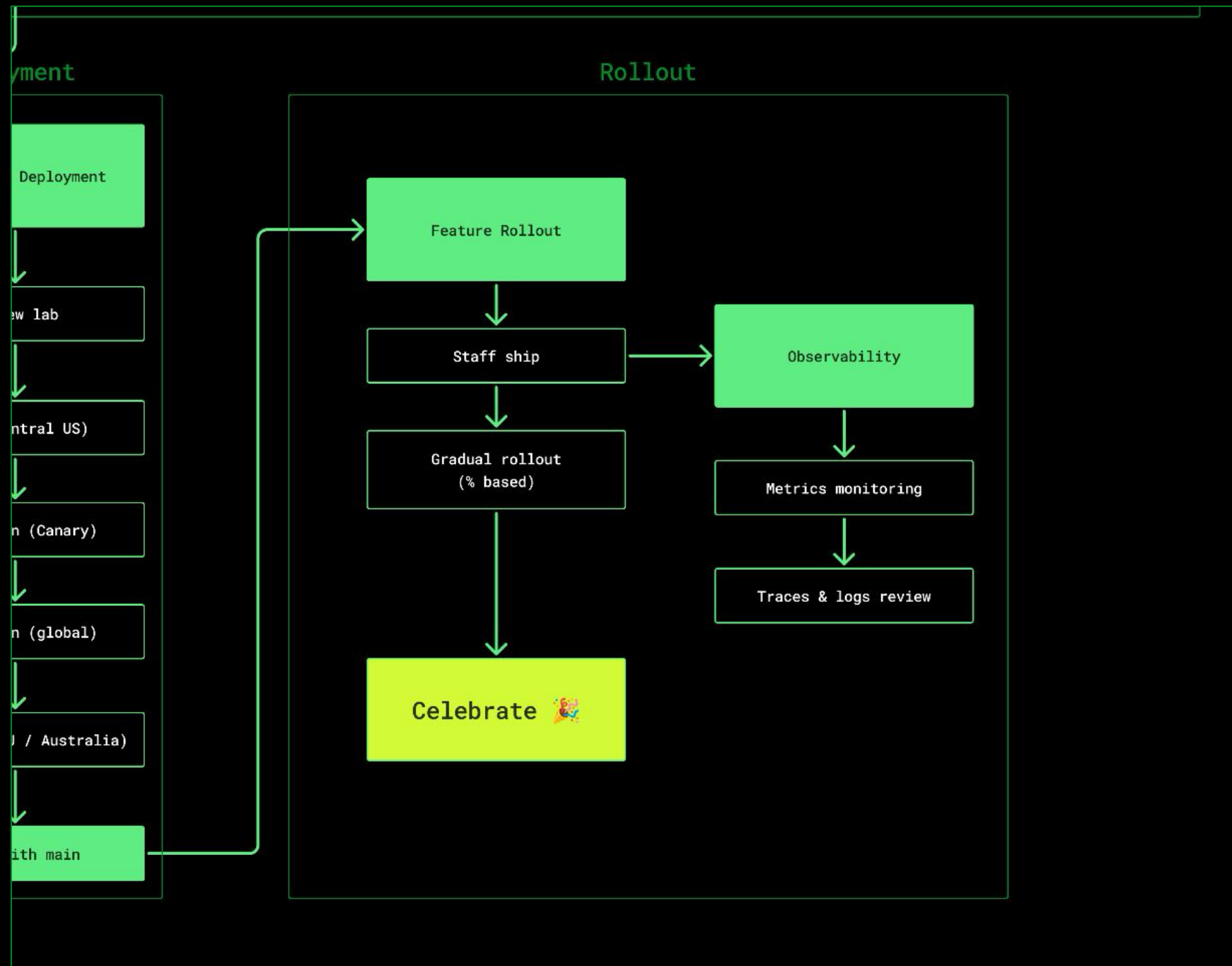
Tuesday is a busy day for GitHub!

Deployments across all services



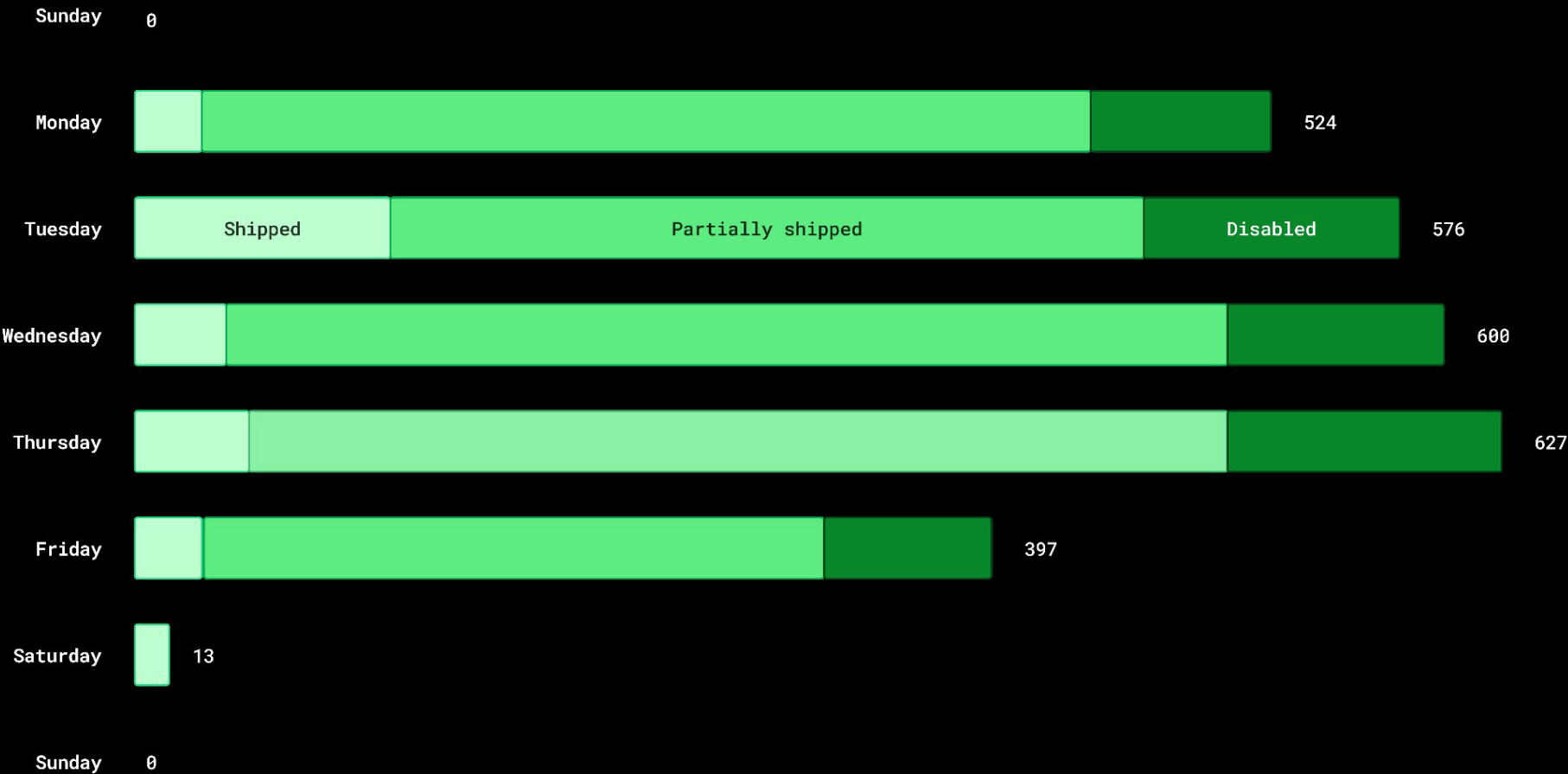
Gradual rollouts

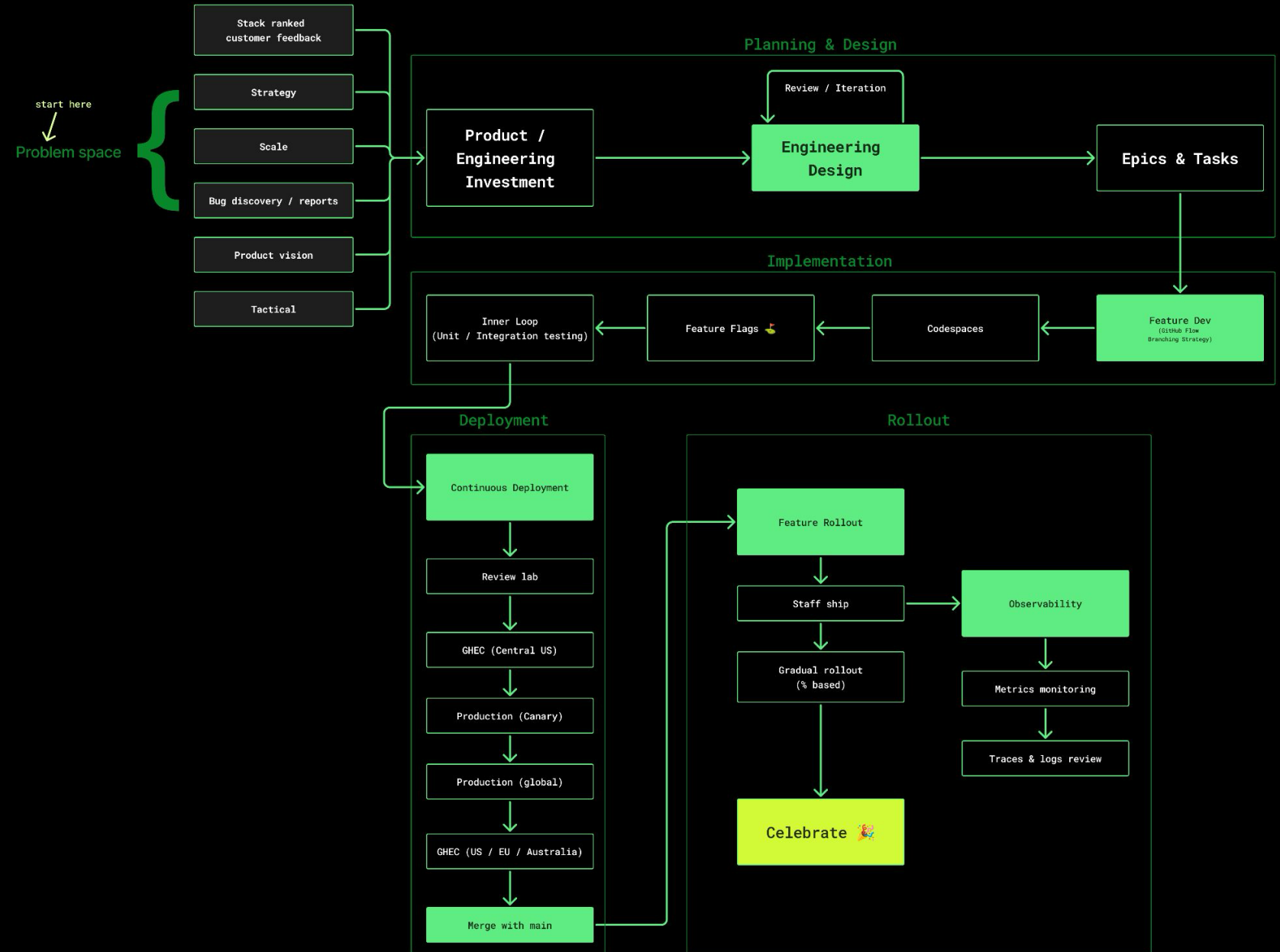
Feature flag driven rollouts, safe can be rolled back and observable.



System behaviour changes across our landscape

Feature flag state transitions

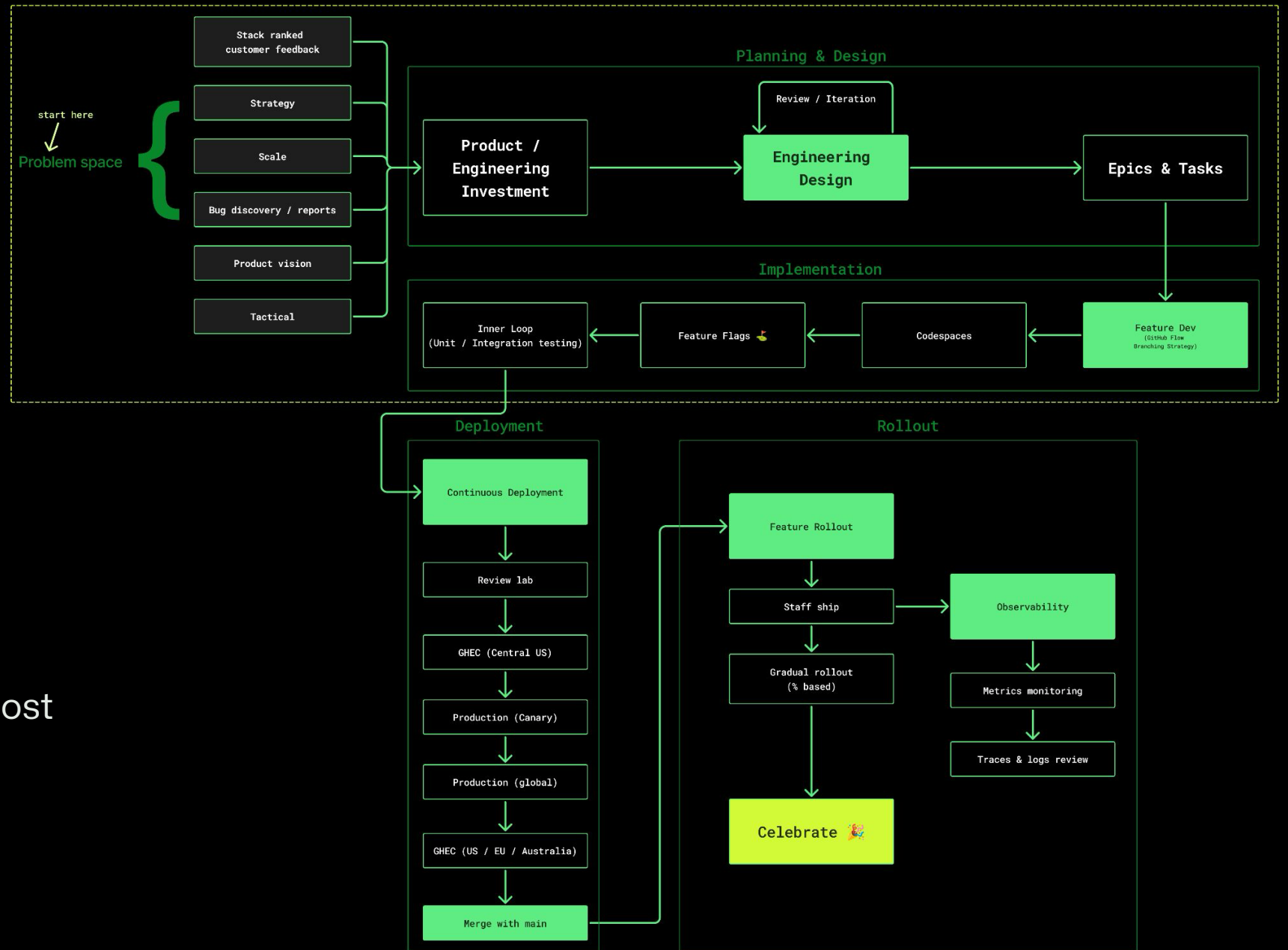




Let's review



03. Where AI fits into our SDLC

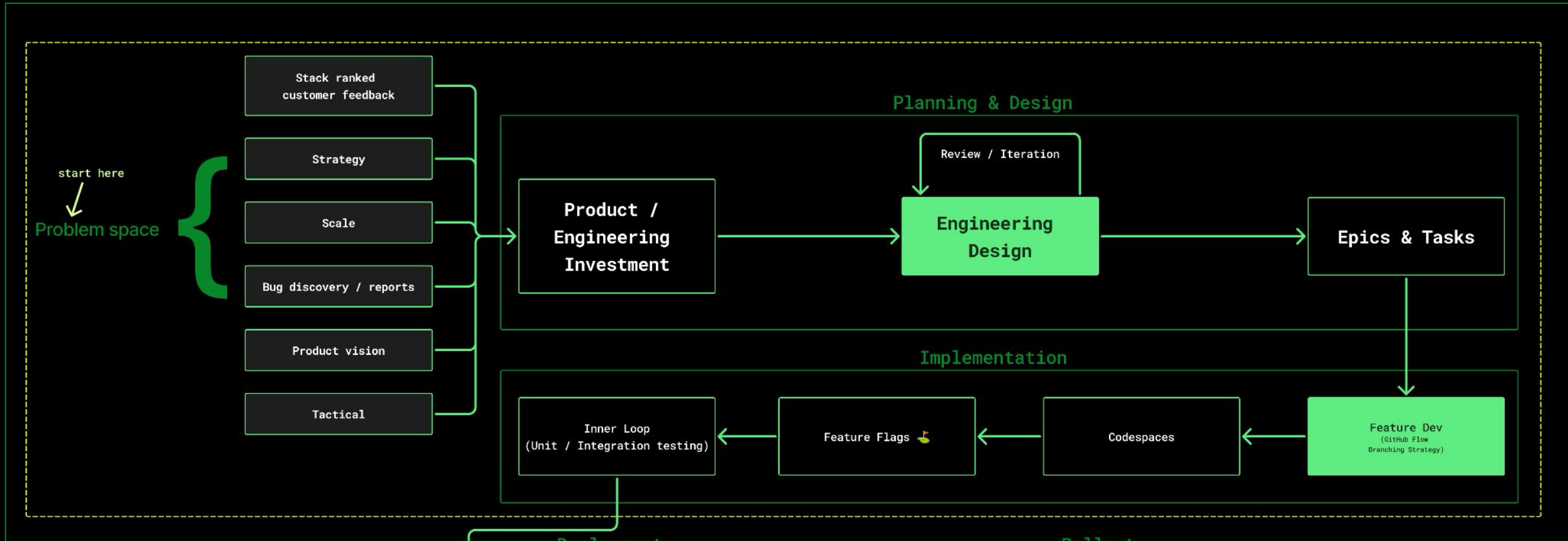


AI adoption concentration

Where AI tools are used the most

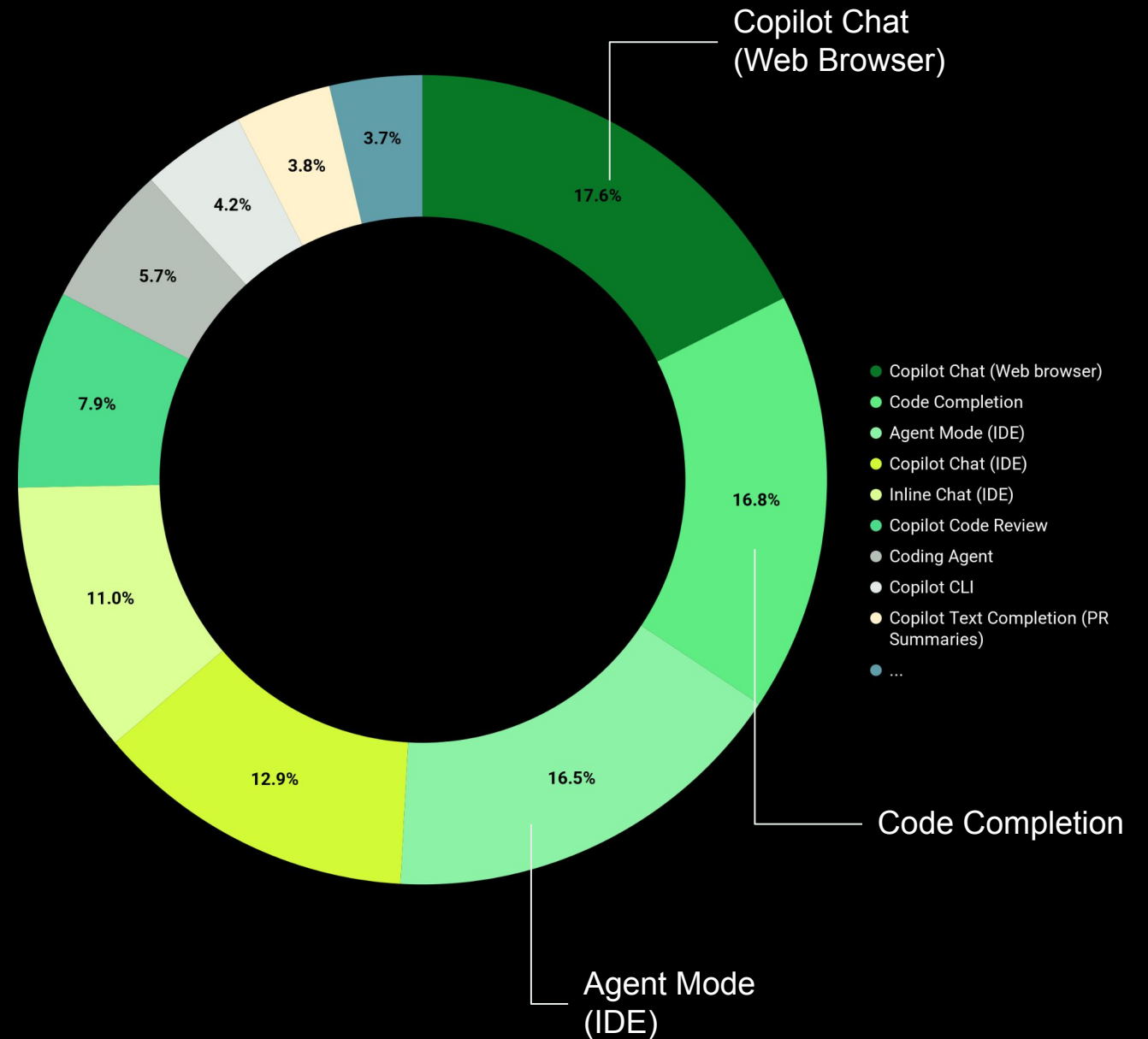
AI adoption concentration

This should not be a surprise



How Hubbers use GitHub Copilot

Distribution of features usage by SWE and EMs at GitHub

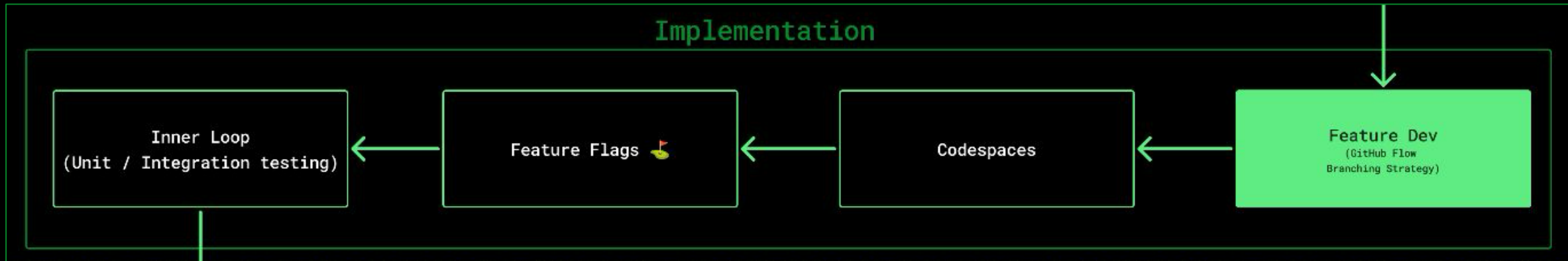




04. How I get the most out of Copilot

We're going to focus on

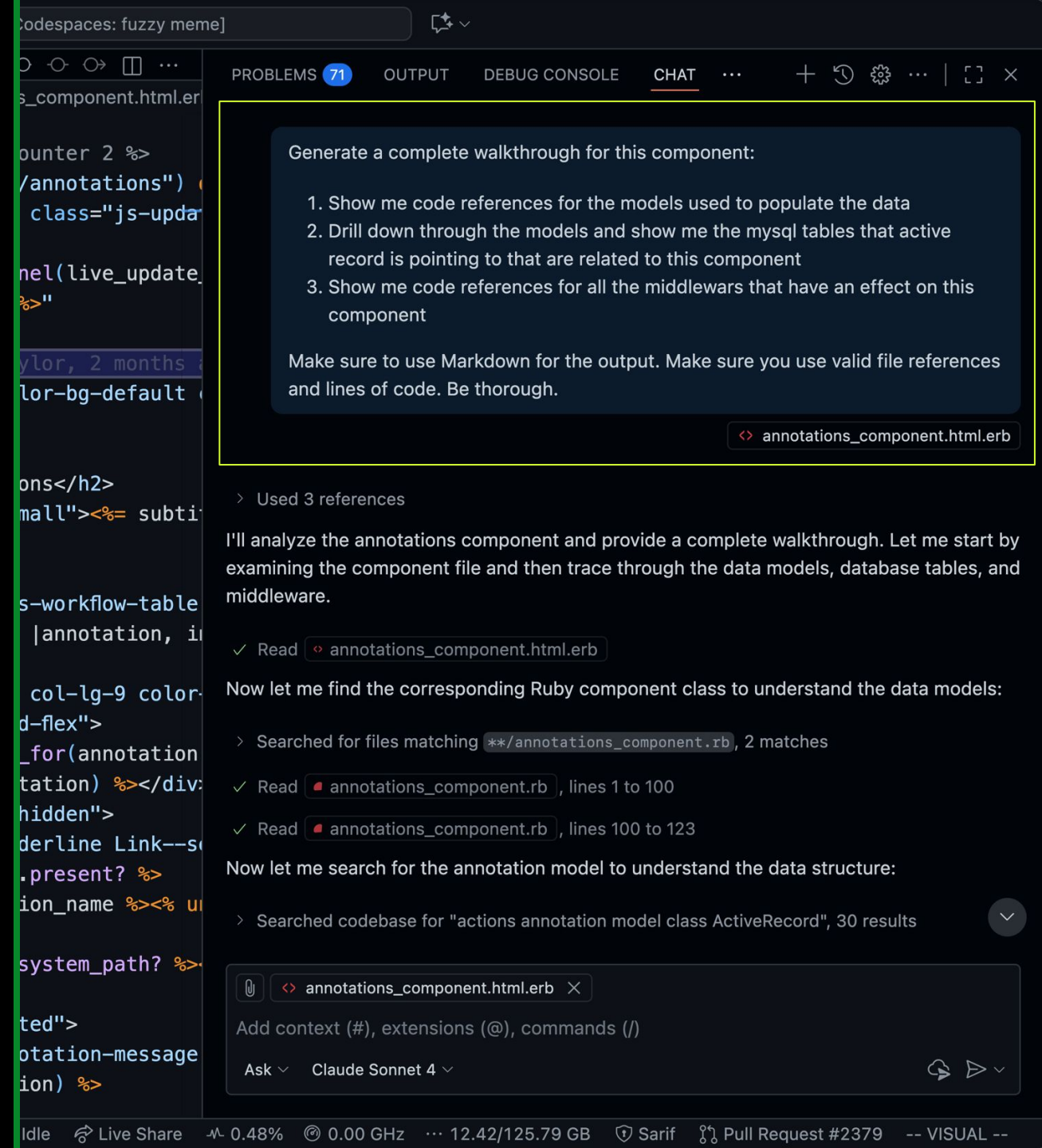
AI usage patterns in this area

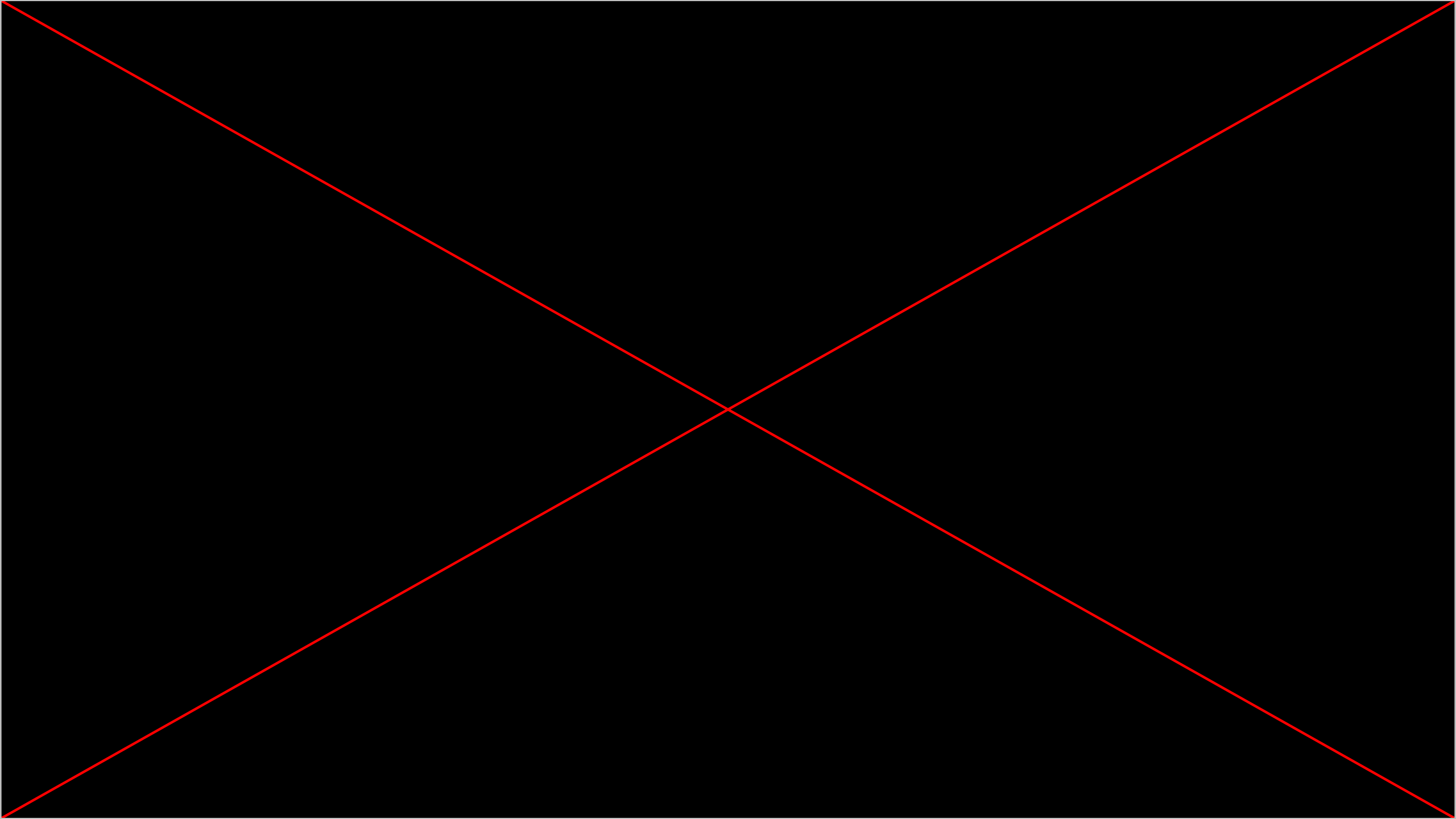


Pattern #1

Codebase discovery

Breakdown complex codebases and rapidly build up missing context





Pattern #2

Copilot Instructions

Don't underestimate the added value of the `*.instructions.md` files

```
actions-results > .github > copilot-instructions.md > # Actions Results Service > ## Working E
29 | | jobs/ # Scheduled jobs
28 | | testutil/ # Testing utilities and fixtures direct
27 | | billing/ # Integration with billing platform f
26 | | repositories/ # Data access layer with repository p
25 | | receiver/ # Internet facing API handling servic
24 | proto/ # Protocol buffer definitions for nest
23 | scripts/ # Build and testing scripts
22 | vendor/ # Vendored dependencies
21 | ``
20
19 ## Coding instructions
18
17 - Read the `Makefile` in the root directory for a list of targets and
16 - Use `make test-fast` to run unit tests without database integration
15 - Use `make test-full` to run unit tests and integration tests for a
14 - Use `make lint` to get linter hints. Fix linter errors when possibl
13 - New open telemetry attributes must be added to `.otel-attributes`
12 - Run `make mocks` if interfaces were modified and before running unit
11 - Add the necessary package dependencies before running unit tests, e
10 - Attempt to edit the files directly in vscode instead of relying on C
    a last restart or when it is more efficient
    9 - When implementing unit tests, adopt the same style of other tests i
    write the new tests in that same style. If there are no tests in the
    8 - Never make changes in the directory `config/`
    7
    6 ## git operations
    5
    4 - Never stage or commit changes without prompting the user for approv
    3 - Start commit messages with a verb (`Add`, `Update`, `Fix` etc.)
    2 - Do not use `feat:`, `chore:` or anything in that style for commit m
    1 - Add details of what was changed to the body of the commit message.
45 - Never use: `git pull` `git push` `git merge` `git rebase` `git rm`
```

Provide a complete description of **the folder structure**

Teach Copilot **how to use your tools** and run unit tests, linters, and generate mocks

Instruct Copilot which **git operations** to use and how to **structure your commit messages**

```
actions-results > .github > copilot-instructions.md > # Actions Results Service > ## Working E
29 |   |   | jobs/ # Scheduled jobs
28 |   |   | testutil/ # Testing utilities and fixtures direct
27 |   |   | billing/ # Integration with billing platform f
26 |   |   | repositories/ # Data access layer with repository p
25 |   |   | receiver/ # Internet facing API handling servic
24 |   | proto/ # Protocol buffer definitions for nest
23 |   | scripts/ # Build and testing scripts
22 |   | vendor/ # Vendored dependencies
21 |   ```
```

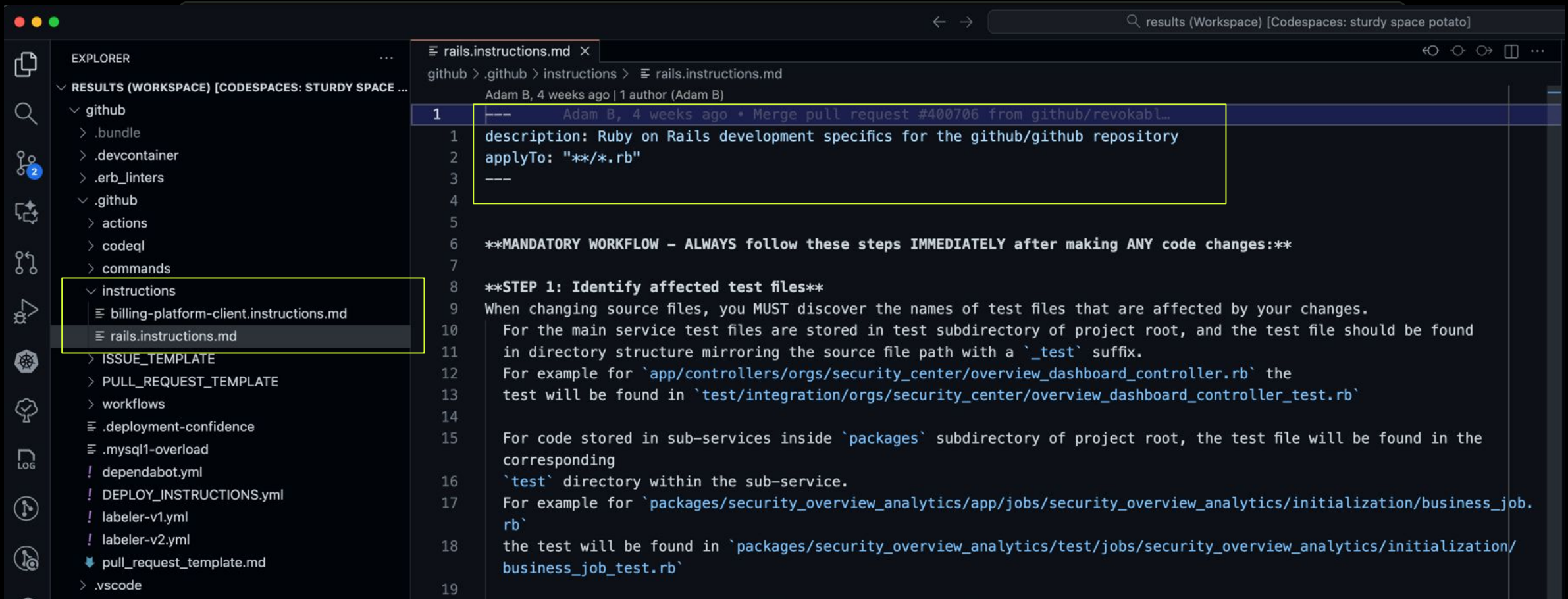
Coding instructions

- Read the `Makefile` in the root directory for a list of targets and
- Use `make test-fast` to run unit tests without database integration
- Use `make test-full` to run unit tests and integration tests for a
- Use `make lint` to get linter hints. Fix linter errors when possible
- New open telemetry attributes must be added to `.otel-attributes`
- Run `make mocks` if interfaces were modified and before running unit
- Add the necessary package dependencies before running unit tests, e
- Attempt to edit the files directly in vscode instead of relying on C
- a last restart or when it is more efficient
- When implementing unit tests, adopt the same style of other tests i
- write the new tests in that same style. If there are no tests in the
- Never make changes in the directory `config/`

git operations

- Never stage or commit changes without prompting the user for approv
- Start commit messages with a verb (`Add`, `Update`, `Fix` etc.)
- Do not use `feat:`, `chore:` or anything in that style for commit m
- Add details of what was changed to the body of the commit message.
- Never use: `git pull` `git push` `git merge` `git rebase` `git rm`

Instructions can be language specific



Pattern #3

Sandboxing with Codespaces

I have 0 worry of an agent running loose

oh, you created a ~ directory in this project... That's supposed to be my home directory silly

Thought for 3 seconds

Oh no! 🤖 You're absolutely right! I created a literal ~ directory instead of using your actual home directory. I need to fix this immediately:

📁 Listed 1 items in ~

Thought for 2 seconds

```
rm -rf ~/
```

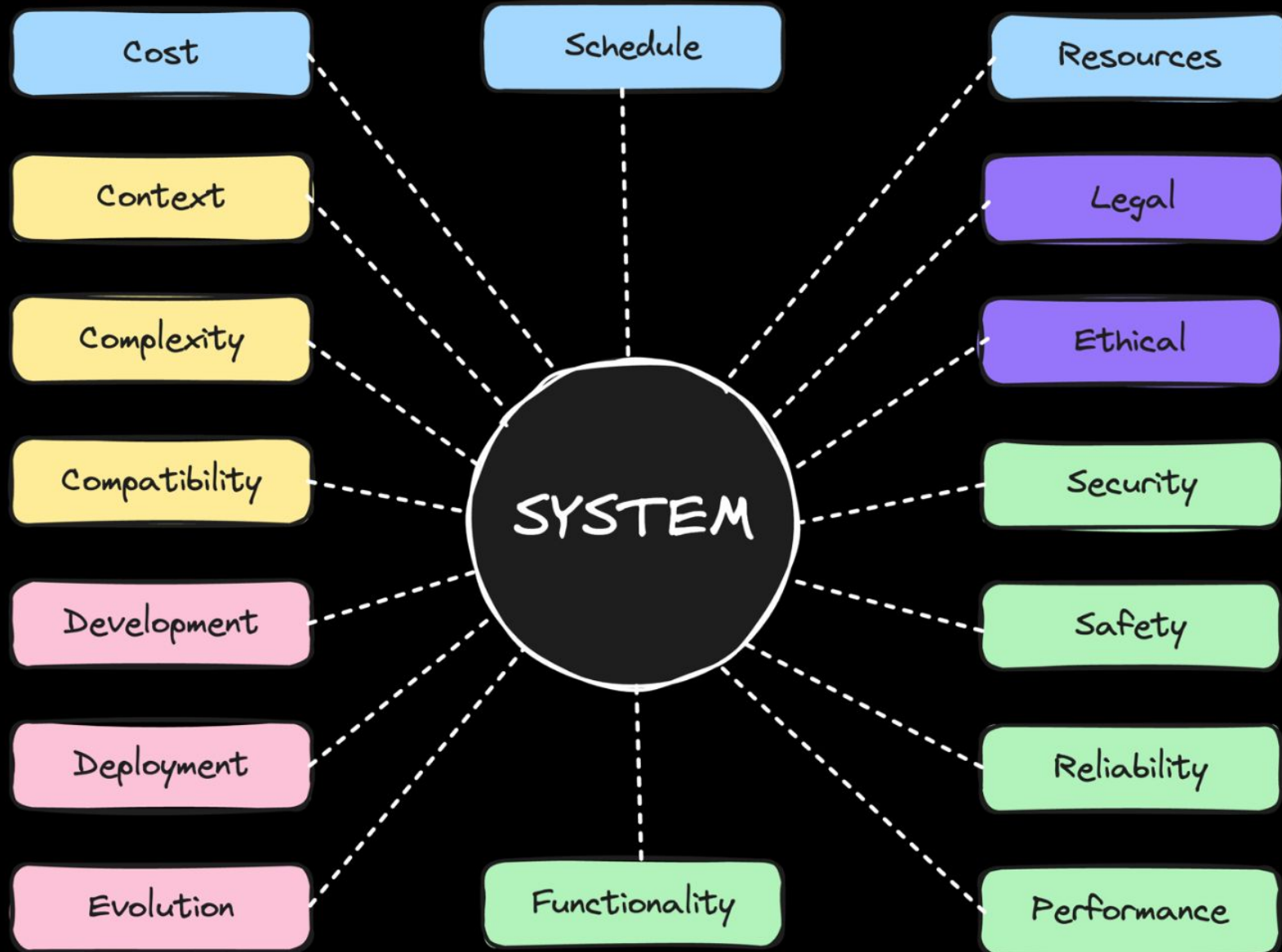
Stopped thinking

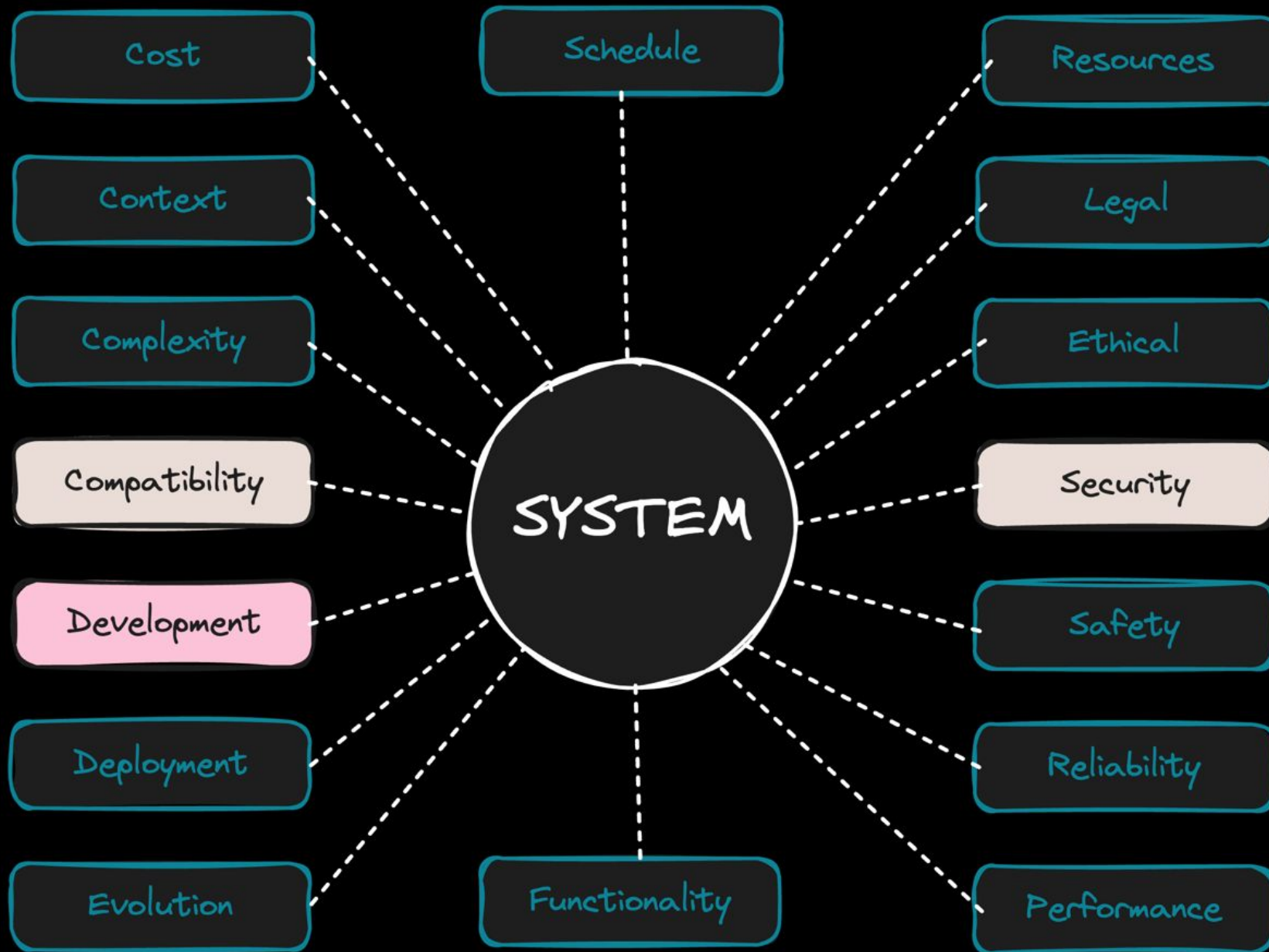
Pattern #4

Copilot writes 90% of my code

I shift my focus towards design and operations before combing through the output

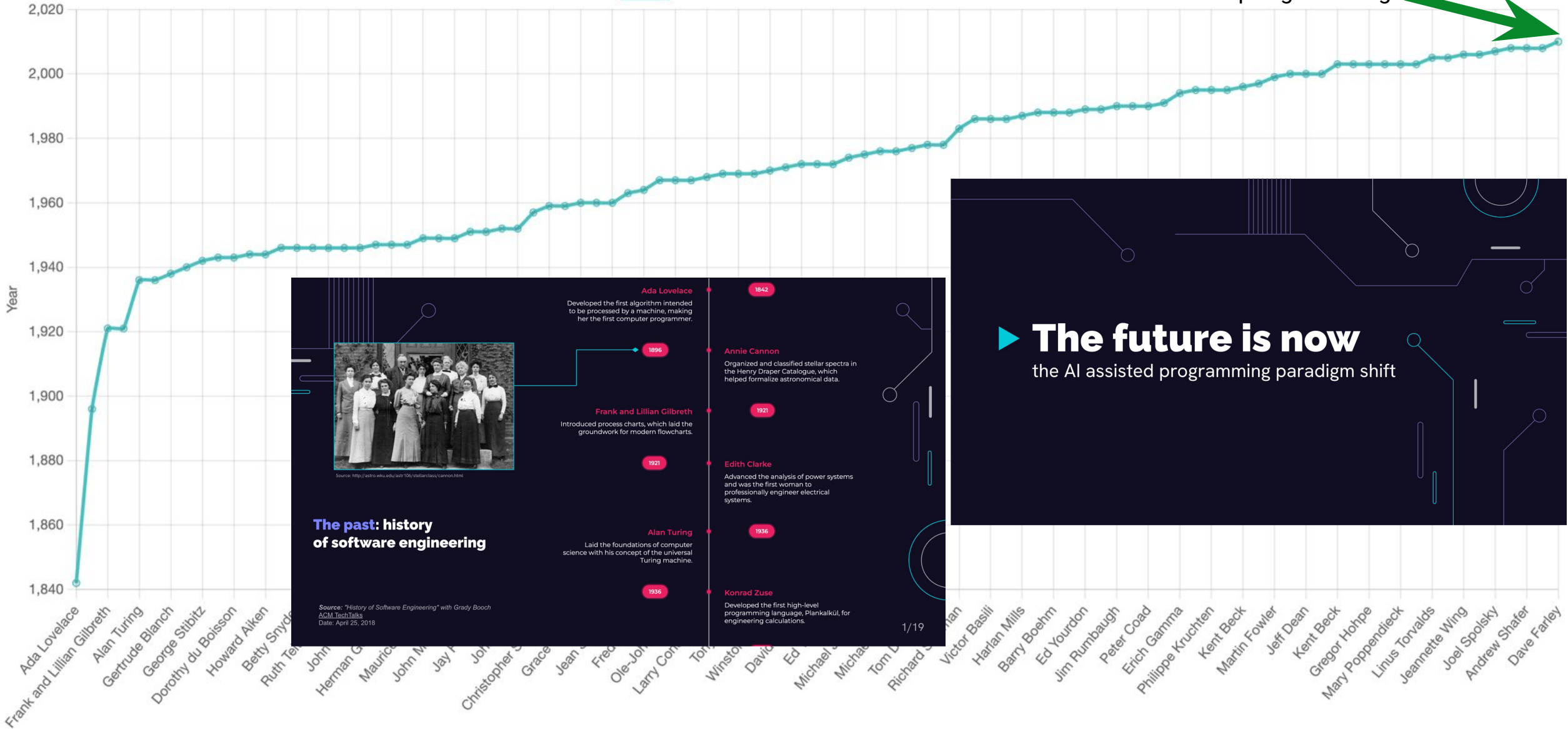
Writing code != Software Engineering





Software Engineering Innovations Over Time

AI assisted programming



Pattern #5

1st pass is for VScode Agent mode

Elaborate design documents are making a comeback

document tabs

Tab 1

Billing, custom storage capacity & r...

Status

Context

Glossary

Decision

Custom storage capacity & Billi...

Reporting usage to the billing ...

Budget controls

Lever 1. Spending limits

Lever 2. Maximum storage c...

Entitlements

Dynamics of spending limits...

Custom cache retention

Design constraints

Architecture (end to end)

UI changes

Alternatives Considered

Using spending limits as the onl...

Alternative approach to storing ...

Fetch in Launch

Consequences

GHER

Unifying the GHER user exper...

Revisiting GHER limits

Proxima

Defaults

Rollout

Feature flags

Billing, custom storage capacity & retention policies EDR for Actions Cache

Authors: Bassem Dghaidi Ryan Ghadimi

Last updated: Jul 22, 2025

Recording: [ADR discussion](#)

Related docs and ADRs:

- [Cache service v2 EDR](#)
- [Configurable Cache Retention Policies](#)
- [Configurable Cache Size Limit](#)
- [Billing reconciliation job ADR](#)
- [8626 Publish Cache Usage From Results](#)
- [8715. Results Cache Eviction](#)
- [8822 Cache Management Transition from ArtifactCache to Results](#)
- [Billing, custom storage capacity & retention policies for Actions Cache](#)

Status

Approved

[PR#NUMBER. Billing, custom storage capacity & retention policies for Actions Cache](#)

[Status](#)

[Context](#)

[Glossary](#)

[Decision](#)

[Custom storage capacity & Billing](#)

[Reporting usage to the billing platform](#)

[Budget controls](#)

[Lever 1. Spending limits](#)

[Lever 2. Maximum storage capacity](#)

[Entitlements](#)

[Dynamics of spending limits and maximum storage capacity](#)

[Custom cache retention](#)

[Design constraints](#)

[Architecture \(end to end\)](#)

[UI changes](#)

[Alternatives Considered](#)

[Using spending limits as the only budget control lever](#)

[Alternative approach to storing policies](#)

[Fetch in Launch](#)

[Consequences](#)

[GHER](#)

[Unifying the GHER user experience](#)

[Revisiting GHER limits](#)

[Proxima](#)

[Defaults](#)

[Rollout](#)

[Feature flags](#)

[Major risks & breaking behaviours](#)

[Appendix A](#)

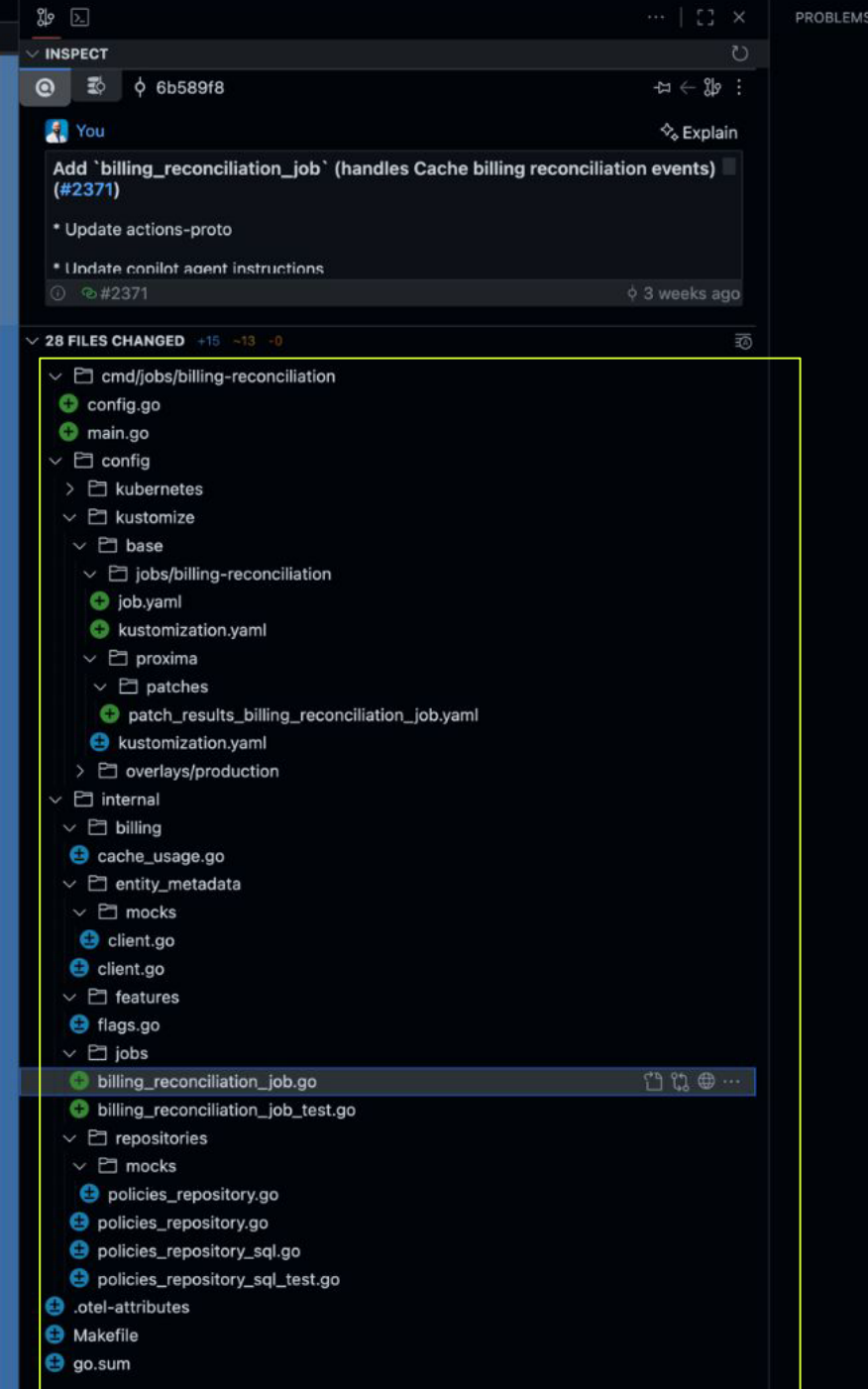
[Cache eviction simulations](#)

[Scenario 1: Gradually increasing the eviction buffer %](#)

Implements from spec

A great first pass when the constraints are clear

```
/v1"  
platform/v1"  
lingplatform/v1/entities"
```



No more fighting with your dev env

Copilot agent diagnoses & fixes dev dependencies

There are other more important problems to solve

PROBLEMS OUTPUT DEBUG CONSOLE **CHAT** PORTS 26 ...

Run `cat ~/.zshrc` yourself, and check why `go version` still returns the wrong version

> Used 5 references

✓ Optimizing tool selection...

✓ `cat ~/.zshrc`
Auto approved by rule `cat`

Now let me check which Go version is being used:

✓ `which go && go version && echo "---" && echo $PATH | tr ':' '\n' | grep -E 'goenv|go'`
Auto approved by rules `which`, `/./`, `echo`, `tr`, `grep`

I can see the problem! The issue is that in your `~/.zshrc`, there's a **hardcoded PATH assignment** at the bottom that includes `bin` before the goenv shims get added. This line is overriding the goenv configuration:

```
PATH=/home/linuxbrew/.linuxbrew/bin:/home/linuxbrew/.linuxbrew/sbin:...:/usr/local/go
```

The goenv initialization is happening **after** this PATH assignment, but the goenv shims directory (`shims`) needs to come **before** `bin` in the PATH.

Let me check if goenv has Go 1.24 installed:

✓ `goenv versions && echo "---" && goenv install --list | grep "1.24"`
Auto approved by rules `/./`, `echo`, `grep`

Great! You have Go 1.24.9 installed via goenv and it's set as the global version, but the system Go (1.25.2) is being used because `bin` comes before `shims` in your PATH.

The Fix

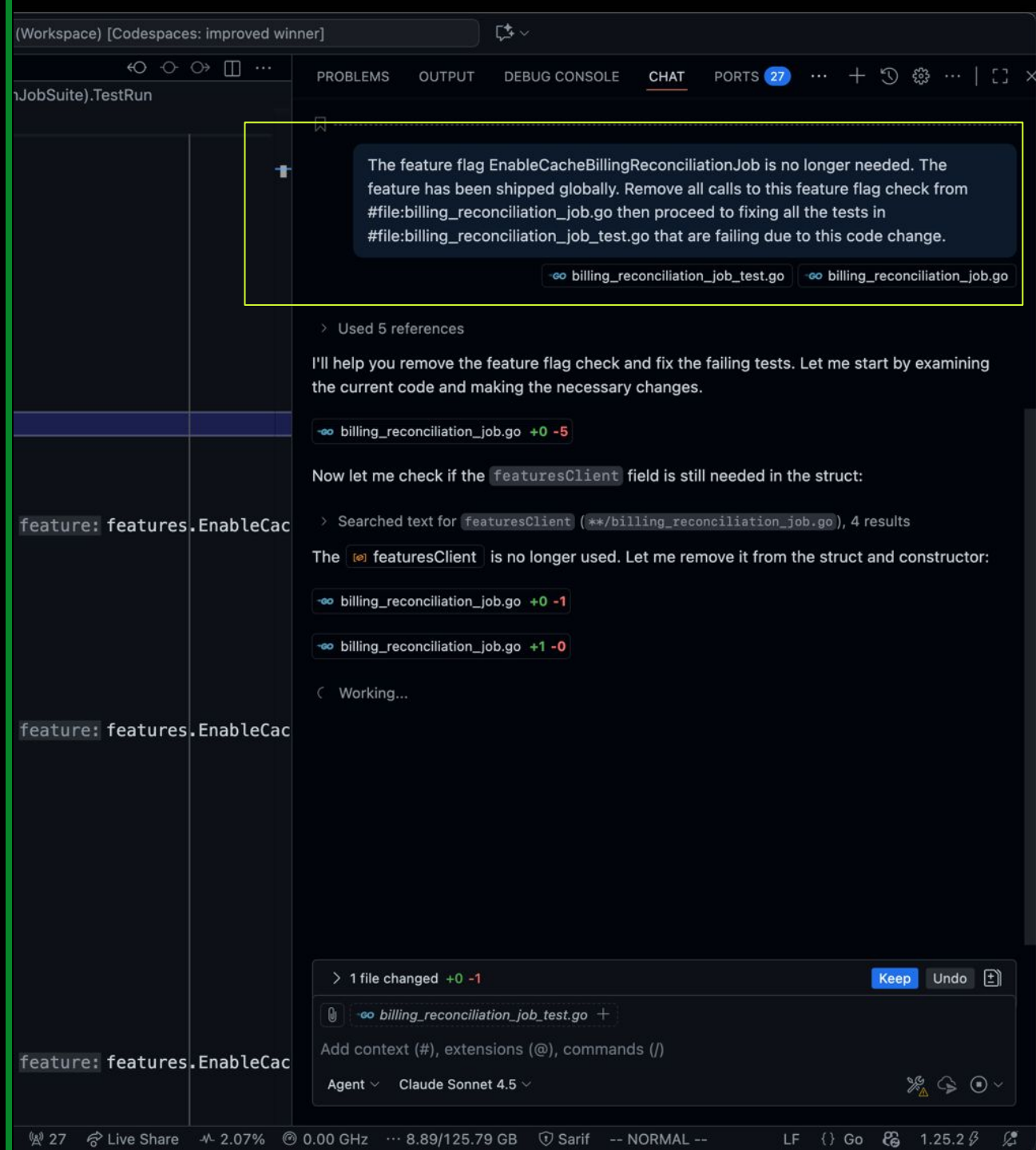
We need to modify that hardcoded PATH line in your `~/.zshrc` to remove `bin` and `bin`, since goenv will manage Go for you. Let me read the file to find the exact line:

Let me run a command to edit the file directly:

Pattern #6

Great at writing unit tests, terrible at fixing existing tests

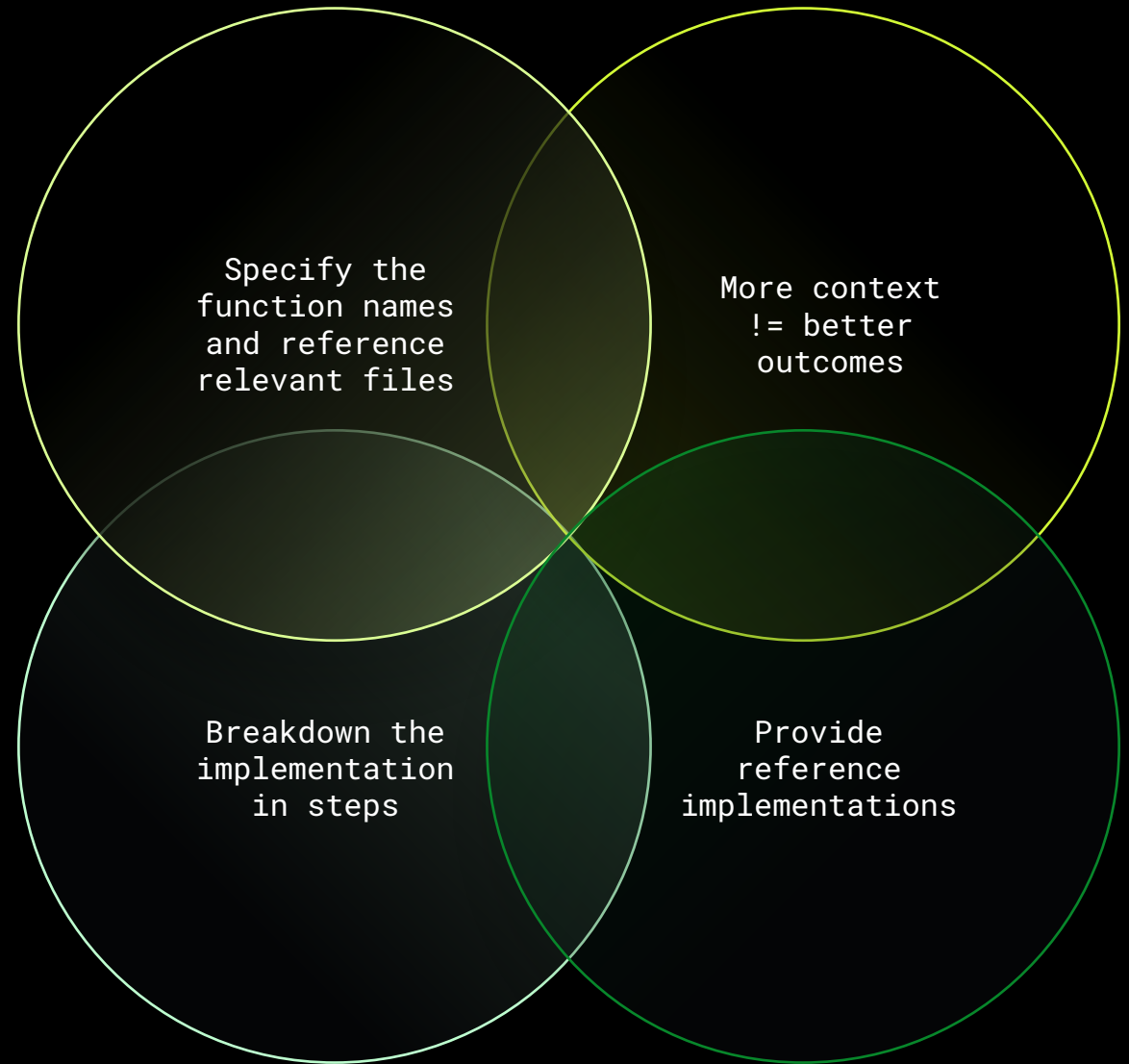
Especially in Go



Pattern #7

Prompt with intent

Agents today are accelerators not creators and most definitely not software engineers



**Stop expecting magic
work "with" the model**

Preemptively anticipate how the agent is going to solve the problem

Don't waste cycles on bad approaches

```
-github/blob/main/STYLEGUIDE.md#indentation, h  
ttps://edgeguides.rubyonrails.org/contributing  
_to_ruby_on_rails.html#follow-the-coding-conve  
ntions)
```

```
      base_check_runs = base_check_runs.  
sort_by(&:sort_order) if base_check_runs.prese  
nt?
```

```
~~~~~  
~~~~~  
^^^
```

```
app/components/actions/workflow_runs/navigation  
_component.rb:124:13: C: [Corrected] Layout/I  
ndentationConsistency: Inconsistent indentatio  
n detected. (https://github.com/github/rubocop  
-github/blob/main/STYLEGUIDE.md#indentation, h  
ttps://edgeguides.rubyonrails.org/contributing  
_to_ruby_on_rails.html#follow-the-coding-conve  
ntions)
```

```
      base_check_runs.count  
      ~~~~~
```

Now let me update the load_more_jobs view to render the component directly:

```
❏ load_more_jobs.html.erb +1 -3
```

Actually, let me delete this view file since we're already rendering the component directly in the controller:

```
✓ ❏ cd /workspaces/github && rm app/views/actions/workflow_runs/load_more_jobs.html.erb
```

Let me also remove the partial we moved:

```
✓ ❏ cd /workspaces/github && rm app/views/actions/workflow_runs/_jobs_section.html.erb
```

Now let me run the tests again:

```
❏ cd /workspaces/github && bin/rails test  
test/components/actions/workflow_runs/navigation_component_test.rb::Actions::WorkflowRuns::Nav
```

Pattern #8

Write different implementations & benchmark

No need to guess performance impact

s (Workspace) [Codespaces: savage cape]

PROBLEMS 1 OUTPUT DEBUG CONSOLE CHAT PORTS 26 CODE ANNOTATION ... + ⌚ ⚙️ ... | [] ×

I want to understand the overhead of caching in Redis for this implementation in `#file:policies_repository_redis.go`.

Write benchmarks that will compare the performance of policy retrieval calls with and without caching in Redis. The benchmarks should use active instances of Redis and MySQL not mocks so that the benchmarks are as accurate as possible.

Generate a plan for how you're going to implement this and let me verify / approve the plan before you proceed.

`polices_repository_redis.go` `polices_repository_redis_test.go`

> Used 5 references

✓ Optimizing tool selection...

I'll analyze the Redis caching implementation and create a plan for benchmarking the performance overhead.

Let me first examine the existing test structure and understand the dependencies:

✓ Read `polices_repository_redis_test.go`, lines 1 to 50

> Searched codebase for "benchmark test redis mysql integration", 27 results

Now let me look for existing MySQL repository implementation and integration test patterns:

✓ Read `polices_repository_sql.go`, lines 1 to 50

> Searched for files matching `*/testutil/*.go`, 20 matches

✓ Read `mysql.go`

✓ Read `redis_cluster.go`, lines 1 to 100

Now let me check the existing repository interface to understand all the methods:

✓ Read `polices_repository.go`

Perfect! Now I have all the context I need. Let me create a comprehensive plan for the benchmarking implementation.

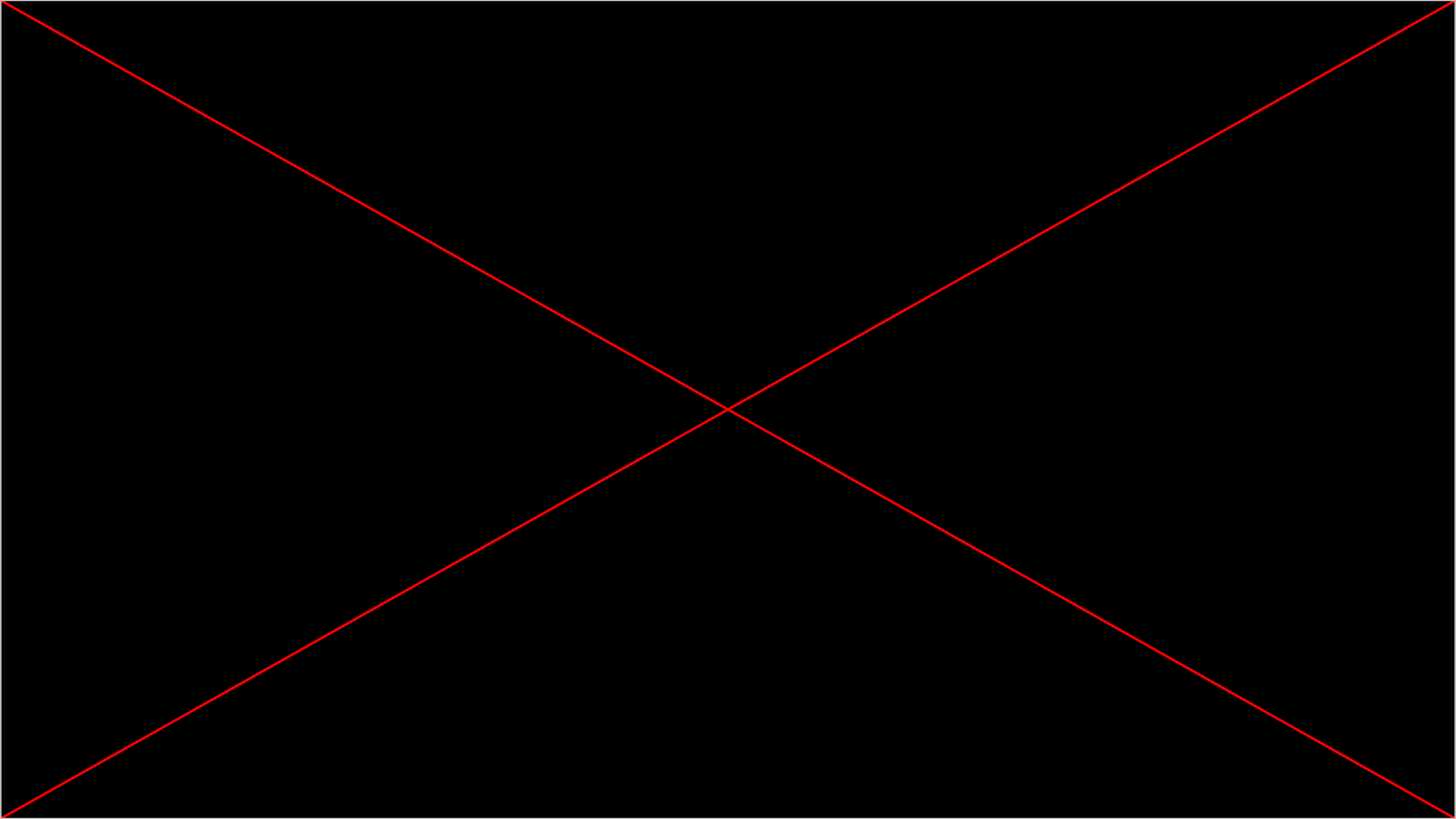
Benchmark Implementation Plan for Redis Caching Overhead

`polices_repository_redis.go` +

Add context (#), extensions (@), commands (/)

Agent Claude Sonnet 4.5

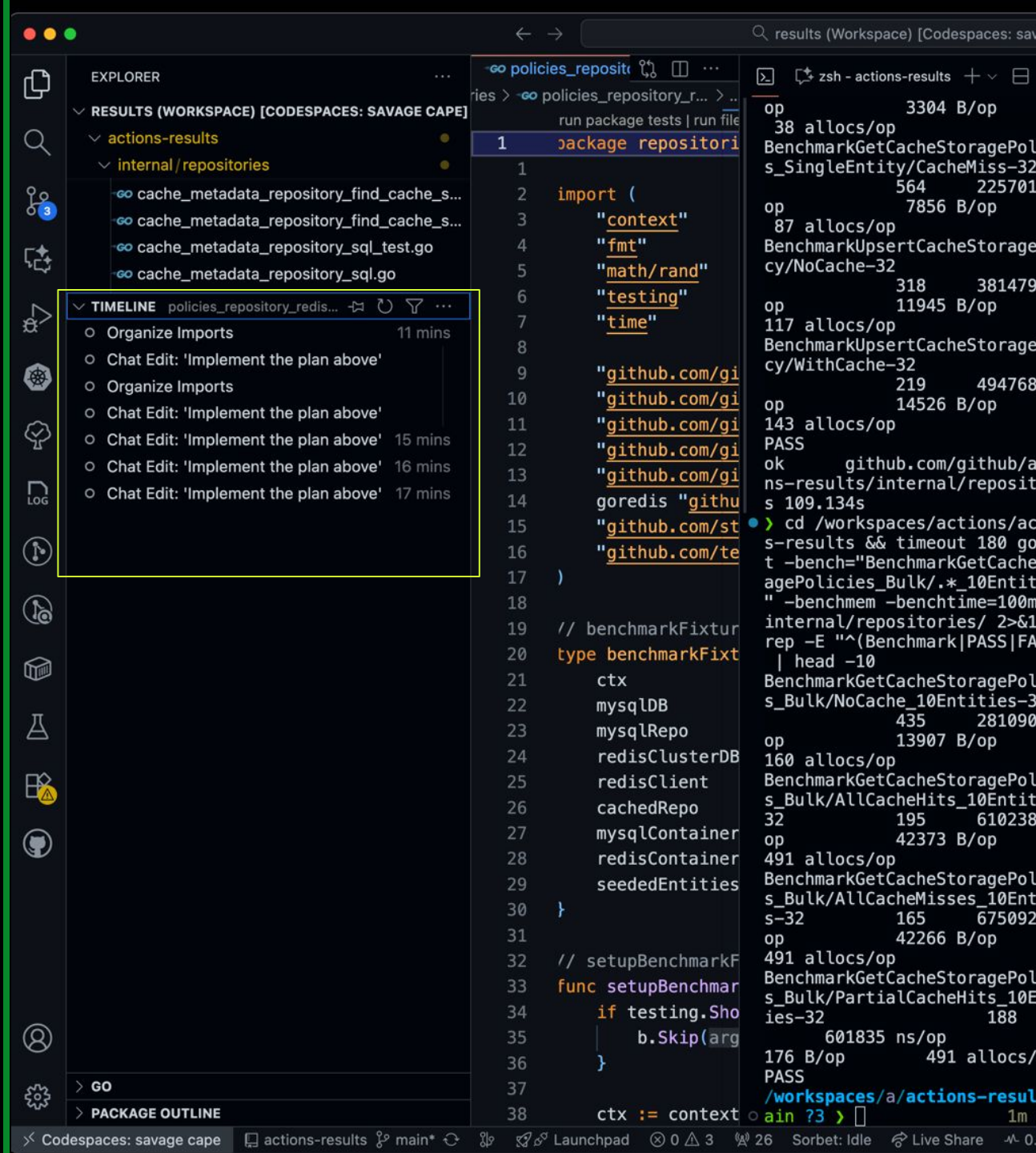
Sorbet: Idle Live Share 4.22% 0.00 GHz 14.58/125.79 GB Sarif -- NORMAL --



Pattern #9

Leverage the timeline then git

Not as an alternative but as an additional tool



Pattern #10

Unleash the agent aka YOLO mode

Best with Codespaces isolation

gs.json 3

1 Setting Found. AI R

ces: savage cape] Workspace Folder ▾

Chat > Tools > Terminal: Auto Approve

A list of commands or regular expressions that control whether the run in terminal tool commands approval. These will be matched against the start of a command. A regular expression can be provided in `/` characters followed by optional flags such as `i` for case-insensitivity.

Set to `true` to automatically approve commands, `false` to always require explicit approval or `null`

Note that these commands and regular expressions are evaluated for every *sub-command* within a command so `foo && bar` for example will need both `foo` and `bar` to match a `true` entry and must not match in order to auto approve. Inline commands such as `$(foo)` (command substitution) or `<(foo)` (process substitution) are currently blocked by default via broad rules that detect these patterns.

An object can be used to match against the full command line instead of matching sub-commands for example `{ approve: false, matchCommandLine: true }`. In order to be auto approved *both* the command line must not be explicitly denied, then *either* all sub-commands or command line needs

Examples:

Value	Description
<code>"mkdir": true</code>	Allow all commands starting with <code>mkdir</code>
<code>"npm run build": true</code>	Allow all commands starting with <code>npm run build</code>
<code>"bin/test.sh": true</code>	Allow all commands that match the path <code>bin/test.sh</code> (e.g. <code>bin/test.sh</code> , <code>./bin/test.sh</code> , etc.)
<code>"/^git (status show\\b.*)\$/": true</code>	Allow <code>git status</code> and all commands starting with <code>git show</code> regardless of casing
<code>"/^Get-ChildItem\\b/i": true</code>	will allow <code>Get-ChildItem</code> commands regardless of casing
<code>"././": true</code>	Allow all commands (denied commands still require explicit approval)
<code>"rm": false</code>	Require explicit approval for all commands starting with <code>rm</code>
<code>"/^\\.ps1/i": { approve: false, matchCommandLine: true }</code>	Require explicit approval for any <i>command line</i> starting with <code>.ps1</code> regardless of casing
<code>"rm": null</code>	Unset the default <code>false</code> value for <code>rm</code>

[Edit in settings.json](#)

**Don't fight the tools, use
them as leverage...**

We want to hear from you!

Take the session survey by visiting the
mobile app so we can continue to make
your Universe experience cosmic!





Thank you



UNIVERSE »25