CPP-Summit

Google

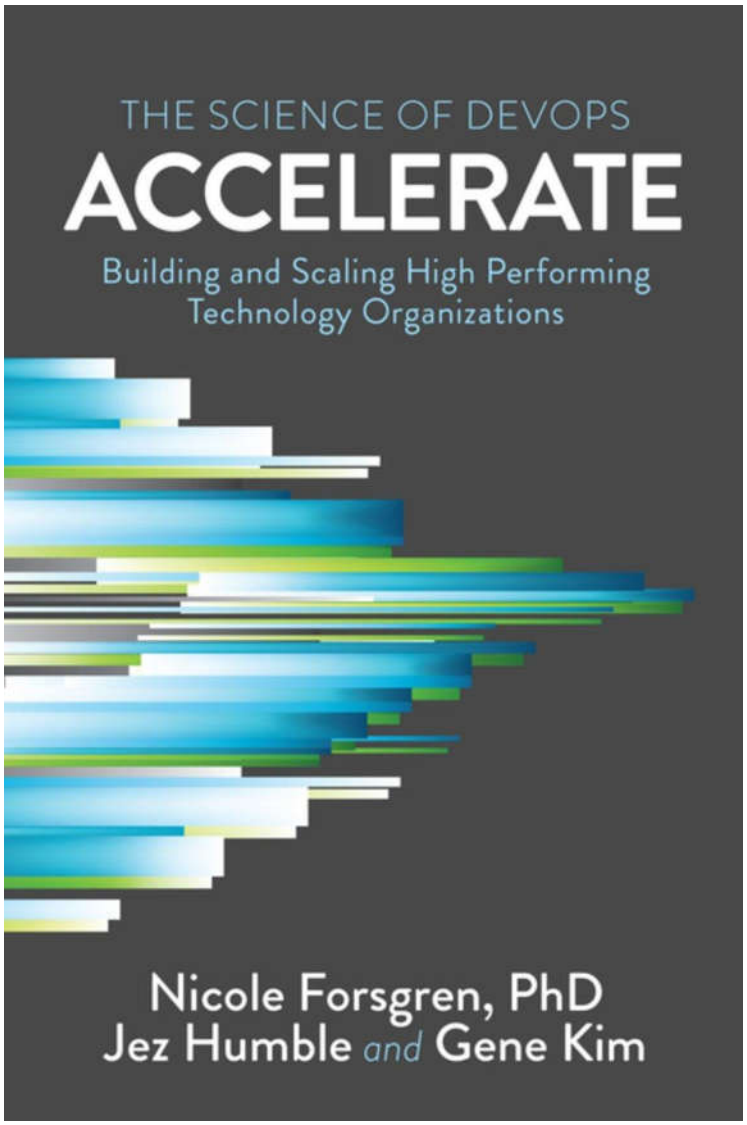
Shifting Left

Cost vs. Fidelity, and Emerging Truths

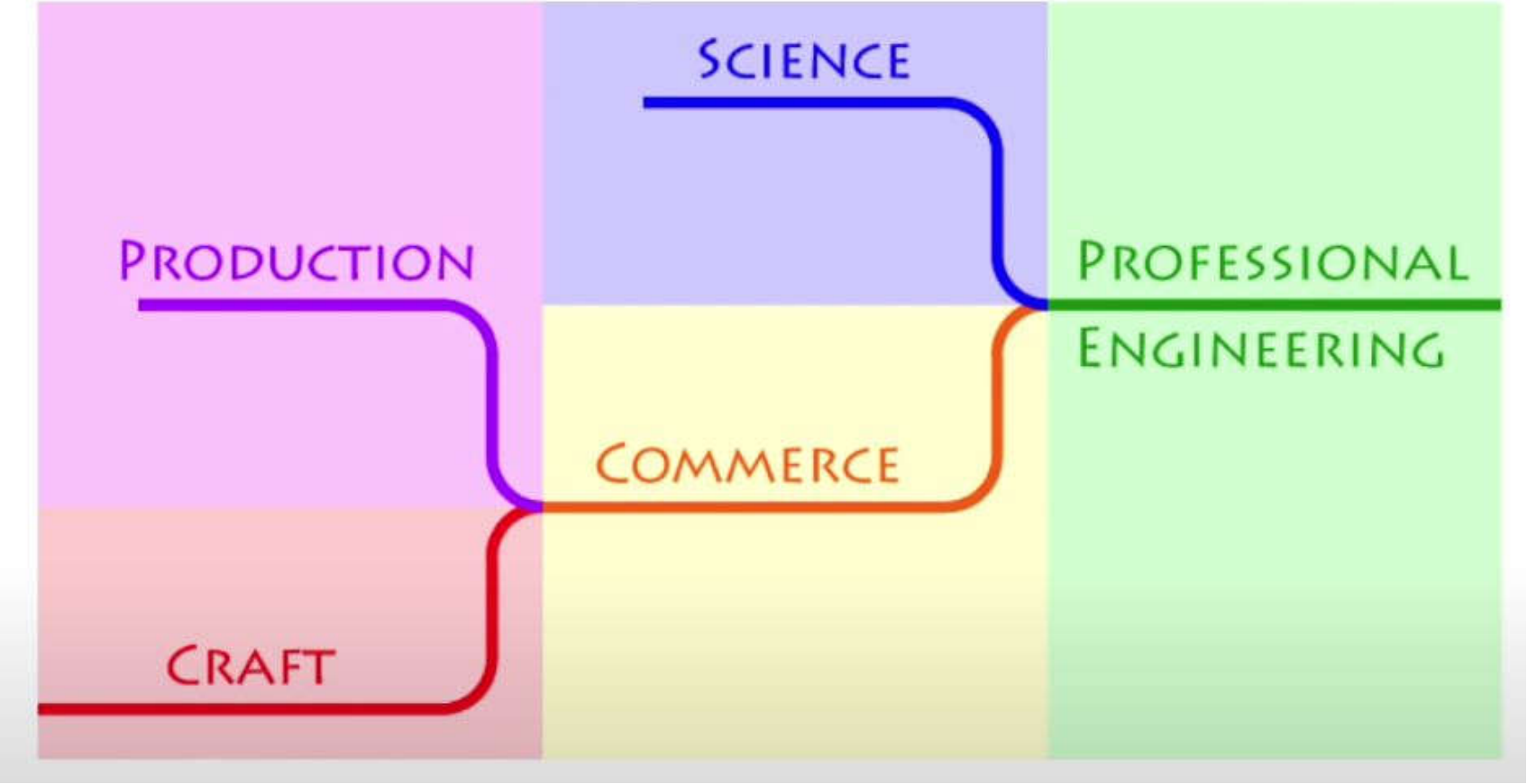
Titus Winters

Software Engineering

Software Engineering?


THE SCIENCE OF DEVOPS
ACCELERATE
Building and Scaling High Performing Technology Organizations
Nicole Forsgren, PhD
Jez Humble and Gene Kim

GOTO 2015 • Progress Toward an Engineering Discipline of Software • Mary Shaw



Software Engineering: First Contact

Define “Software Engineering”

Dave Parnas: “The multi-person development of multi-version programs.” (Circa 1970)

Russ Cox: “Software engineering is what happens to programming when you add time and other programmers.” (Circa 2018)

“It's Programming if "clever" is a compliment.

It's Software Engineering if "clever" is an accusation.”

Software Engineering != Programming

Truth #1: Hyrum's Law

With a sufficient number of users of an API,

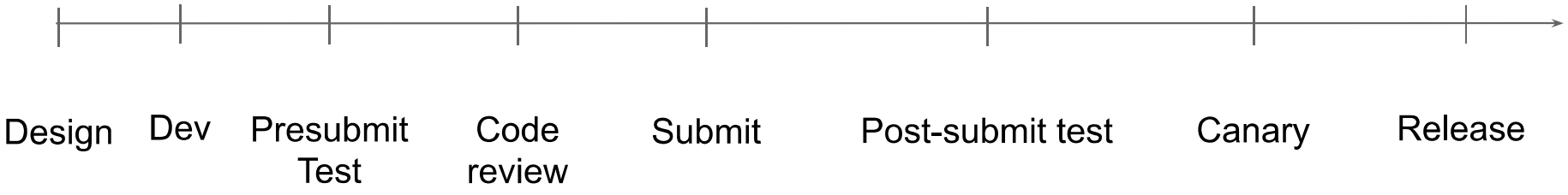
it does not matter what you promise in the contract:

all observable behaviors of your system will be depended on by somebody.

Truth #2: Change will Happen

Over the expected lifespan of your code, **sustainable** code is capable of changing everything that *ought* to change, safely.

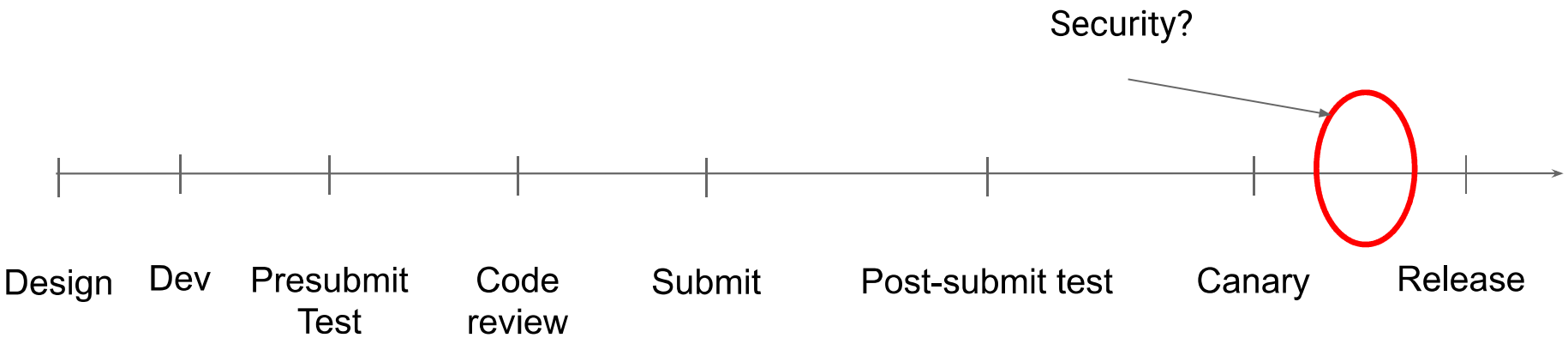
Truth #3: Shifting Left - Earlier is Cheaper



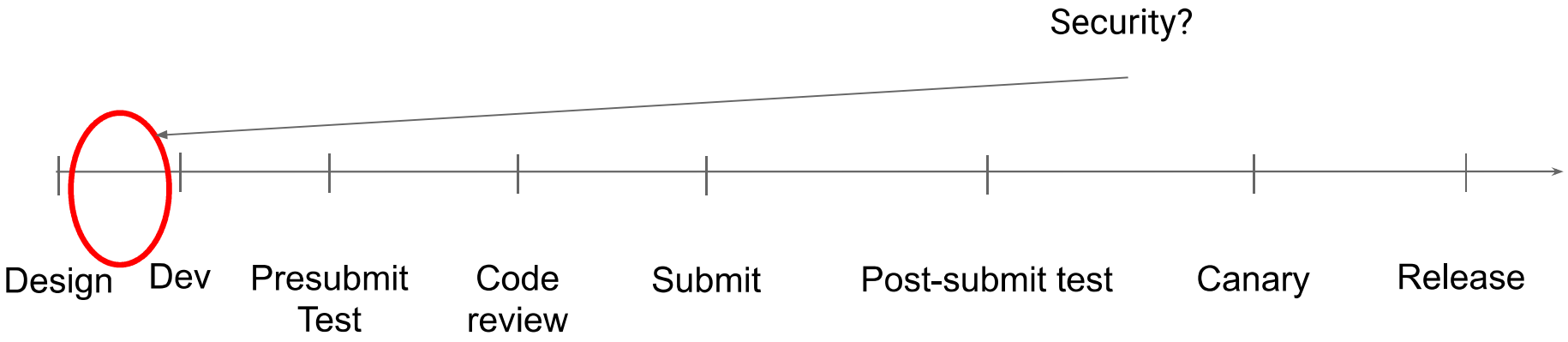
Shift Left

"Shift Left on Security"

Shift Left



Shift Left

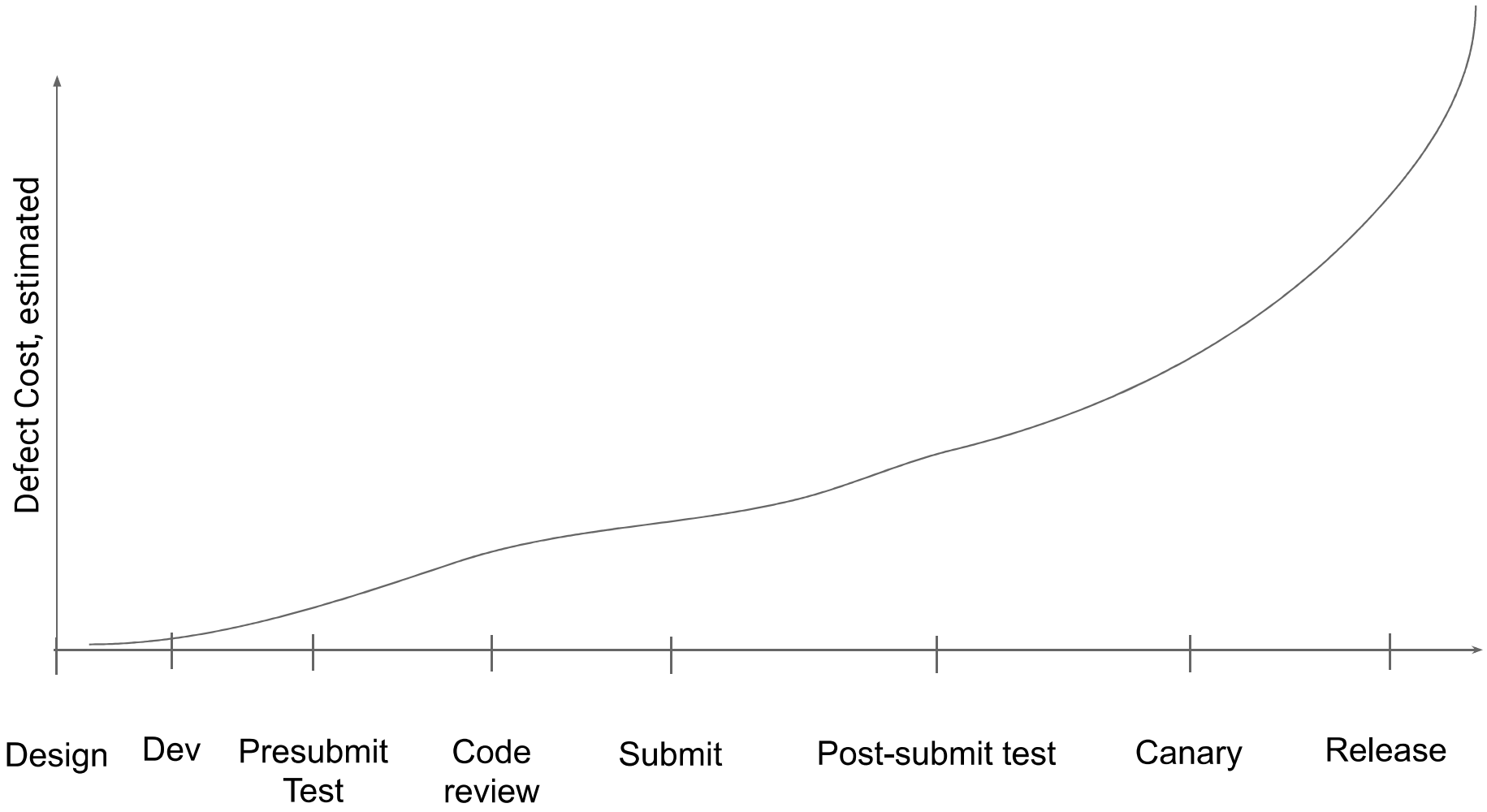


Shift Left

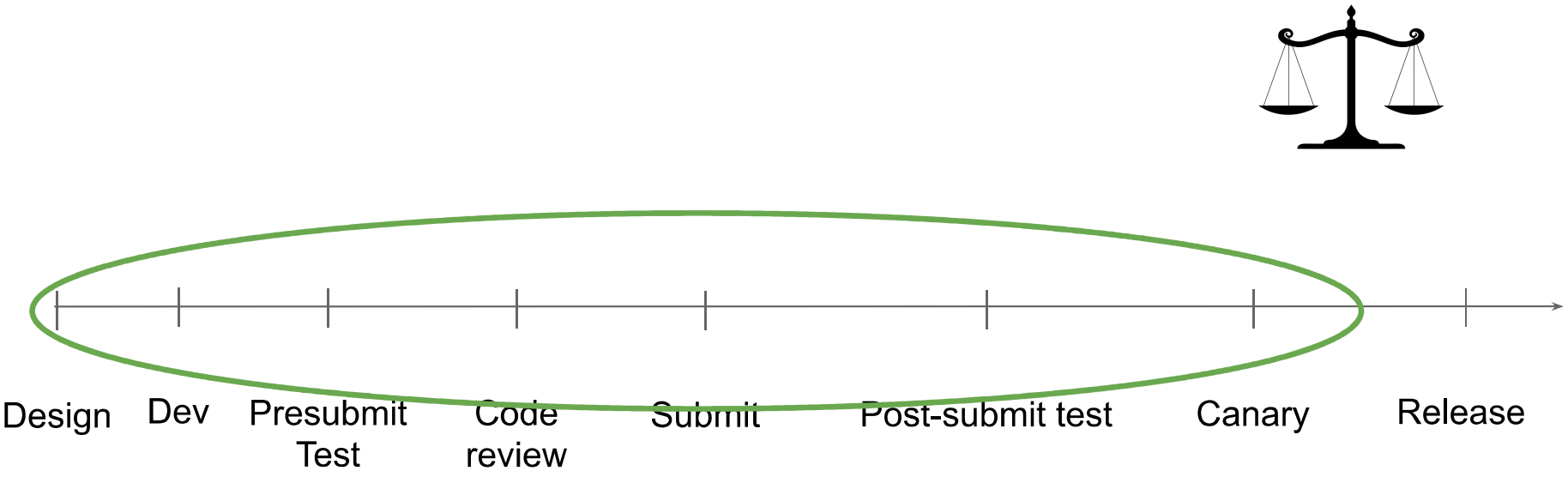
Defect Cost != Speed, Fun, Productivity

(Software Engineering != Programming)

Shift Left



Shift Left



Shift Left

Continuous Integration

is

Alerting

CI is Alerting

Invariant checks:

- Tests are evaluating invariants tied to logical correctness
- Monitoring is based on application health invariants

Paging / Alerting because a poorly chosen line was crossed: brittle

Paging / Alerting off and on for unclear reason: flaky

These are (or could be) the same invariants!

CI is Alerting

Highest ground-truth value alerts: Constant probing “Is the site up?”

Everything else: Proxies for “Is the site *healthy*?”

(A *prediction* of whether the site will stay up.)

CI is Alerting

Unittests

Flaky / brittle tests

End-to-end tests

Keep trunk green

Monitored stats/metrics

Flaky stats / cause-based alerts

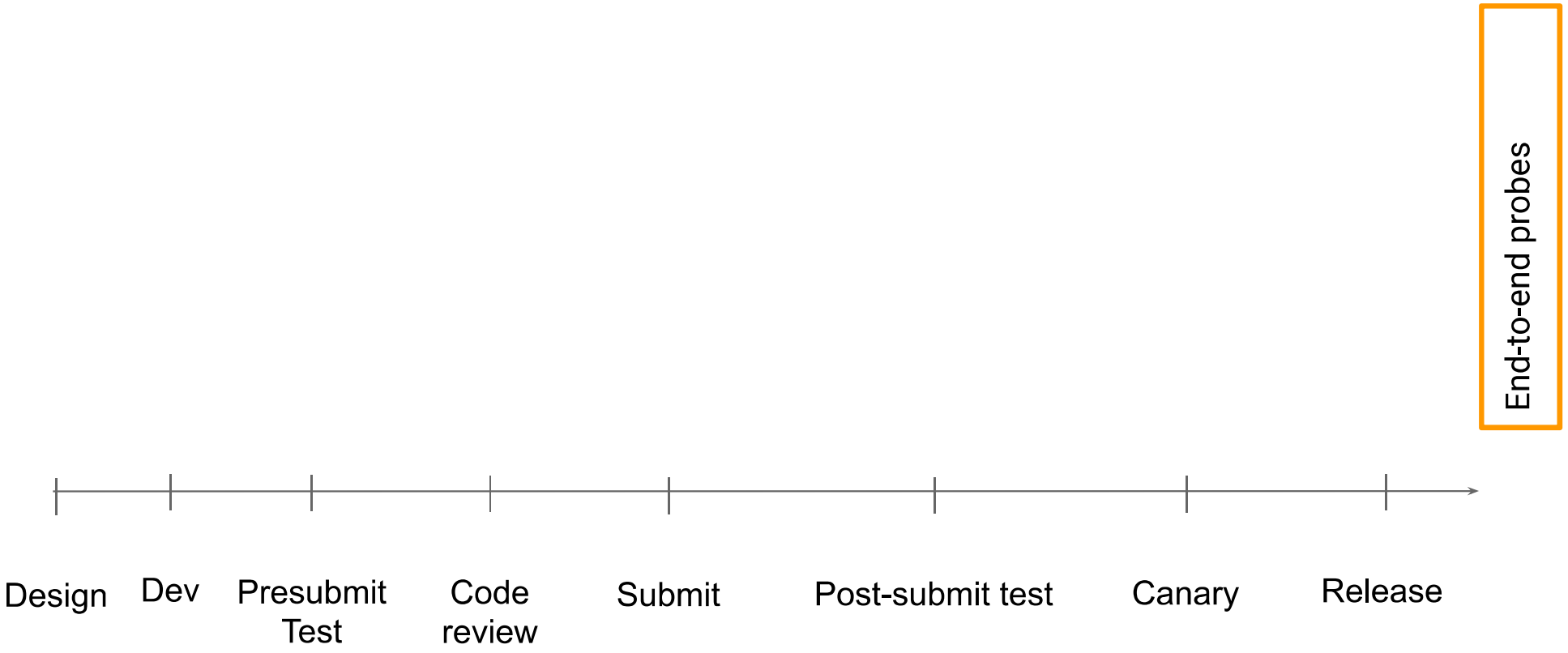
End-to-end probers

Error budgets

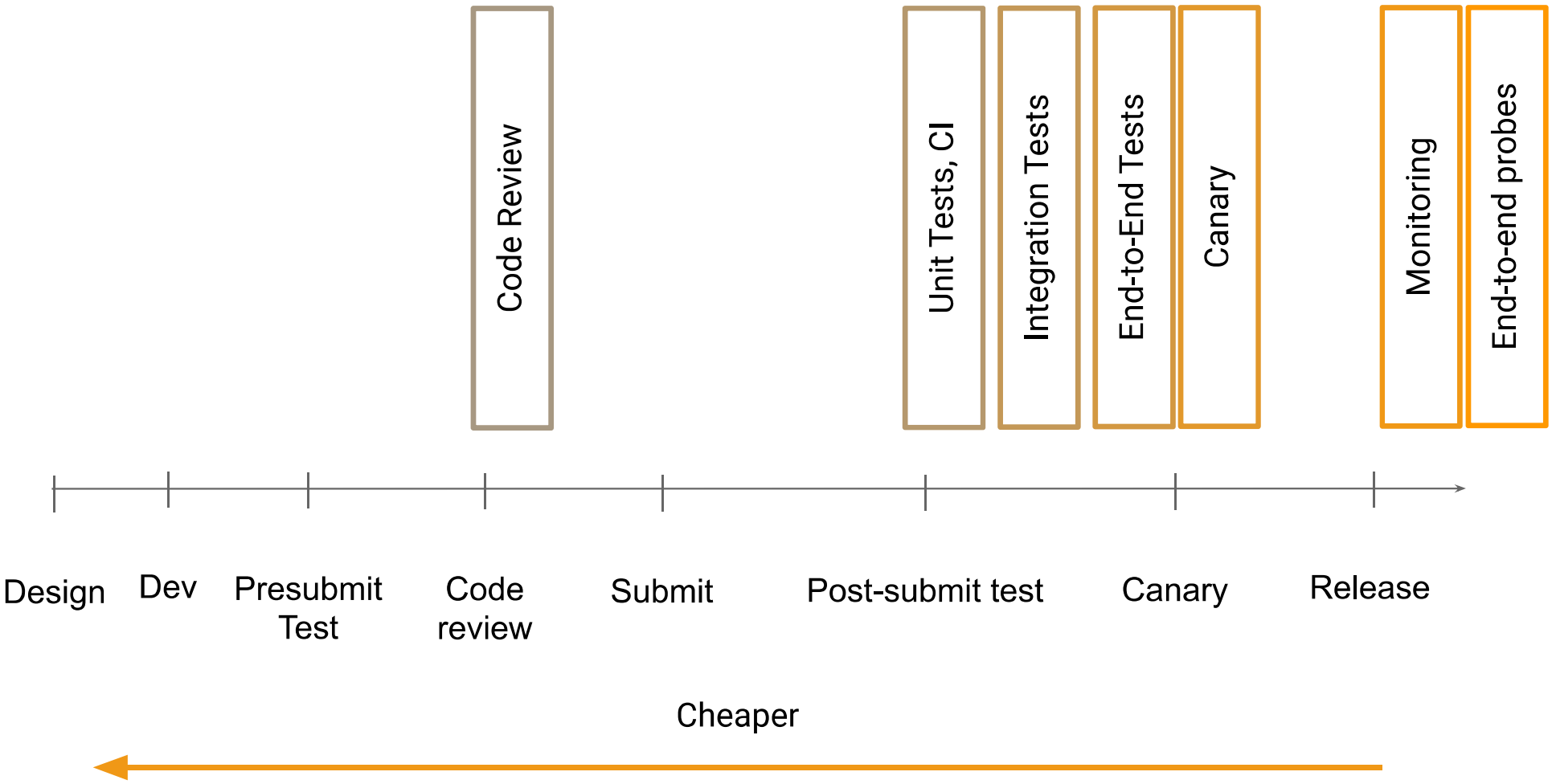
CI is Alerting

We’re optimizing for “product fitness”

Product Fitness, Proxies



Product Fitness, Proxies



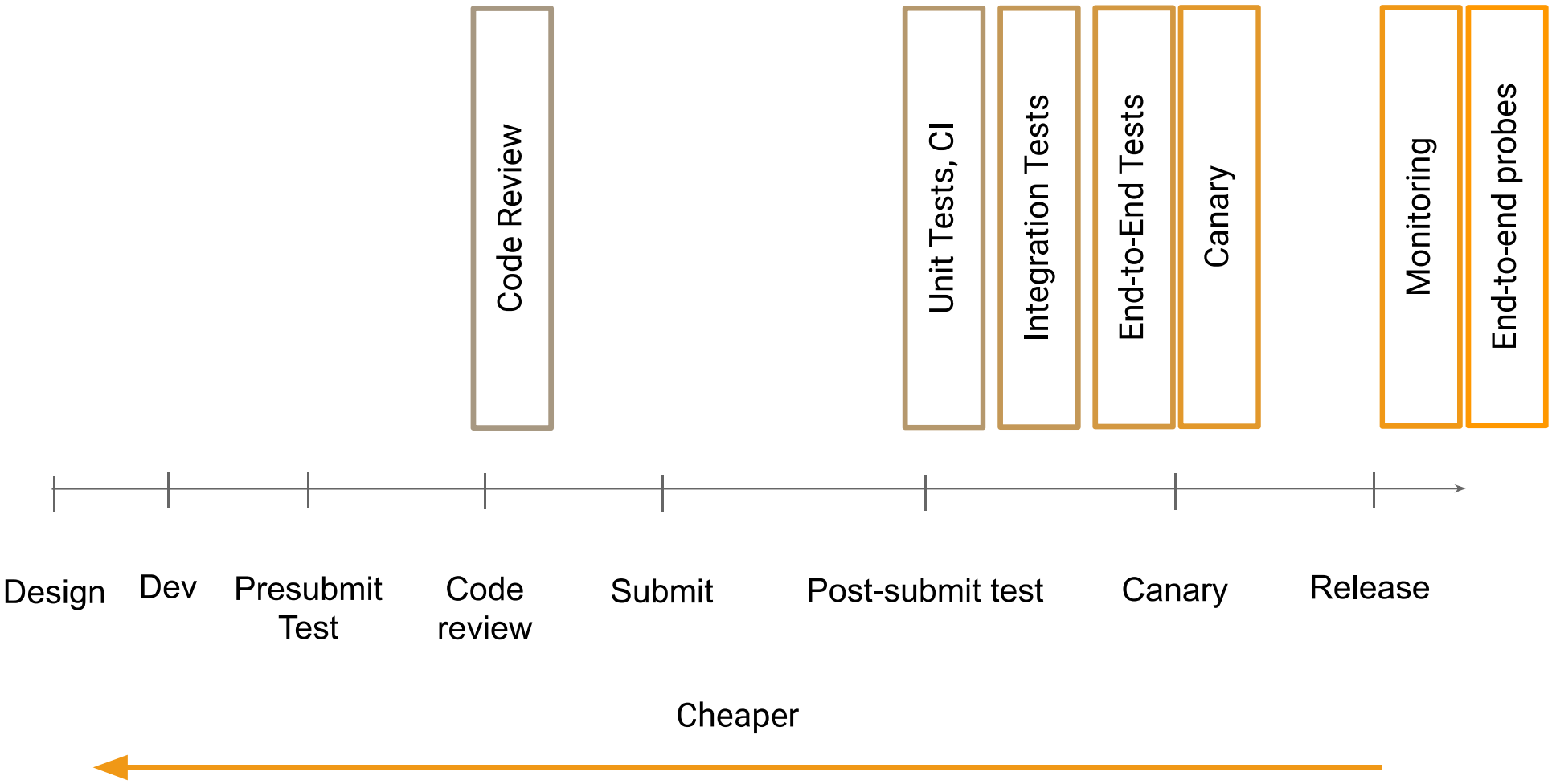
SRE Policies in the SWE Workflow

Error Budgets

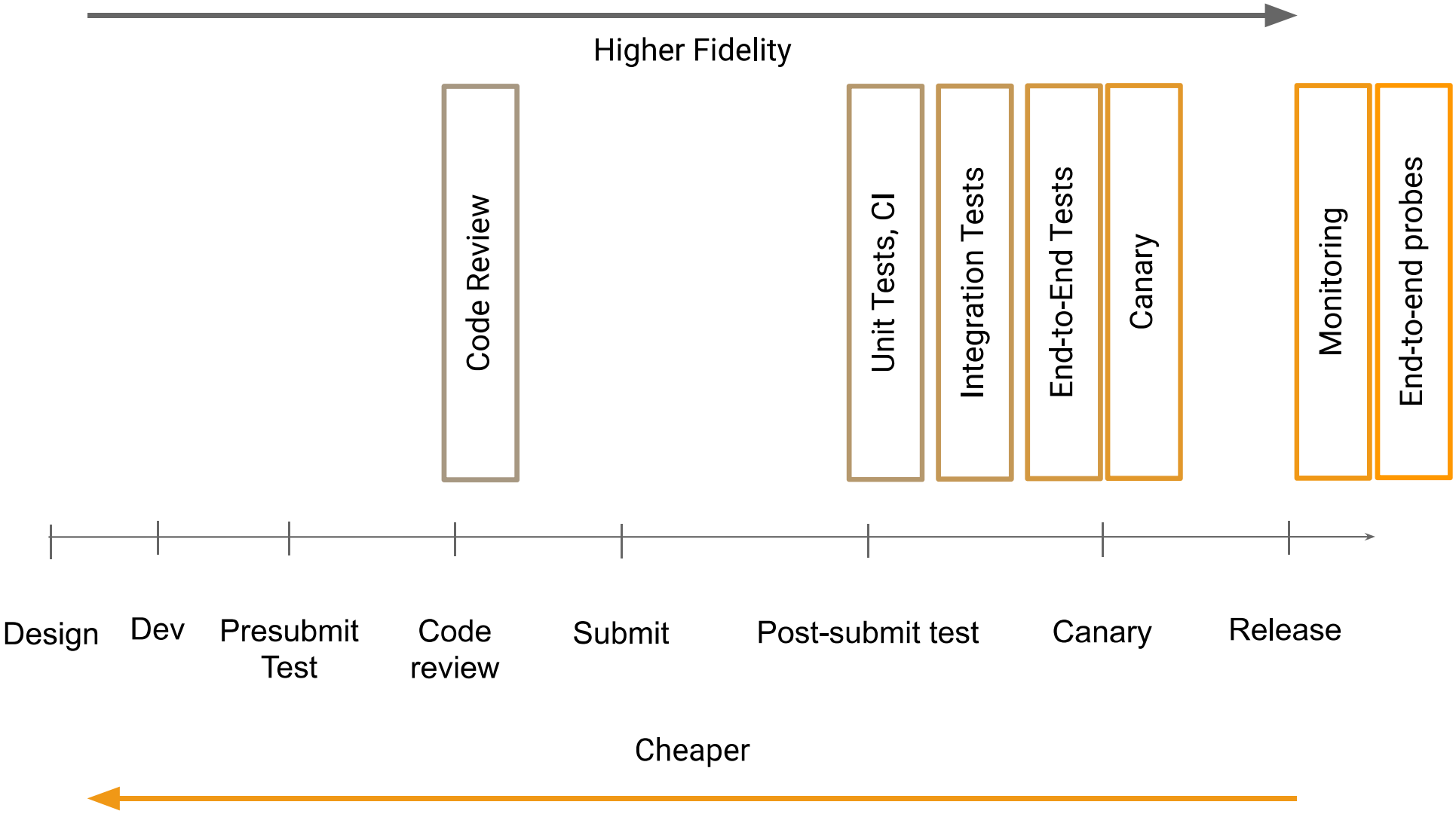
Shift Left

We’re optimizing for “product fitness”

Product Fitness, Proxies



Product Fitness, Proxies



Truth #3: Shifting Left

"Shifting Left" in your engineering workflow is trading between risk/cost and fidelity.

The SWE Workflow

- Everything is about **intra**-team communication, product fitness, or both.
- Workflow that requires 1:N communication proportional to org size is a risk.
- Shifting any process to the right will eventually be a problem - those costs often grow super-linearly with your team size.
- Arguments of the form “but we already have X that solves this, so we don’t need Y” should be ignored when X is further to the right than Y.
- Arguments of the form “X catches more than Y, so let's drop Y” should also be ignored when X is further to the right than Y. It may catch more, but it’s still cheaper to do it earlier if you can manage - and we aren’t aiming for perfection in any given stage.

Software Engineering?

Software Engineering as a Discipline

Some understanding of the boundary of the field.

Some emerging truths.

Questions?