Software Engineering - Principles

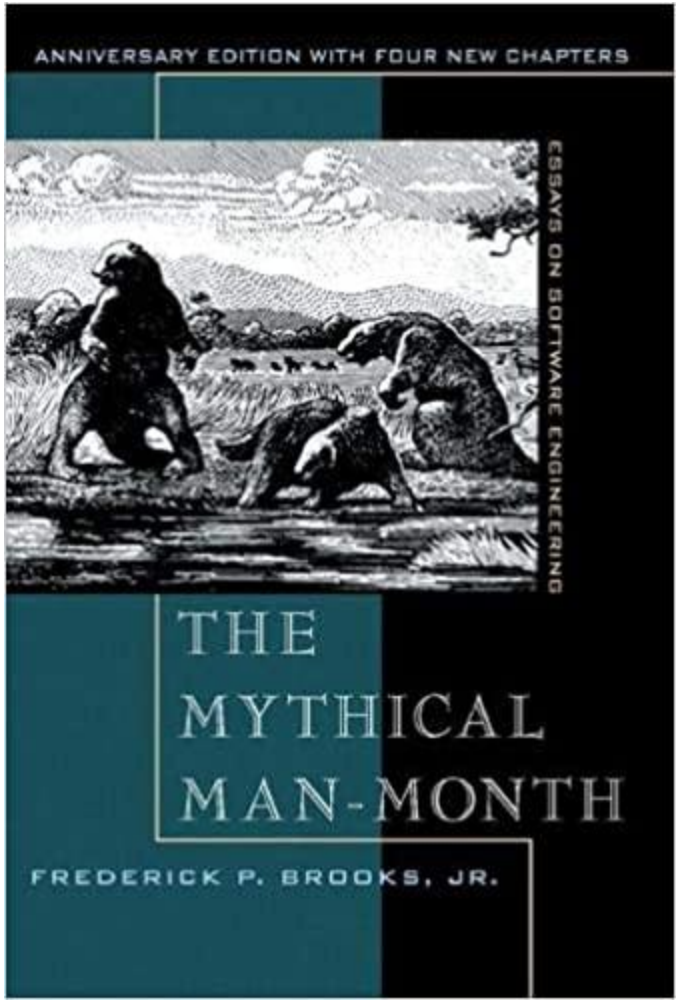
aka SWE Book Theses

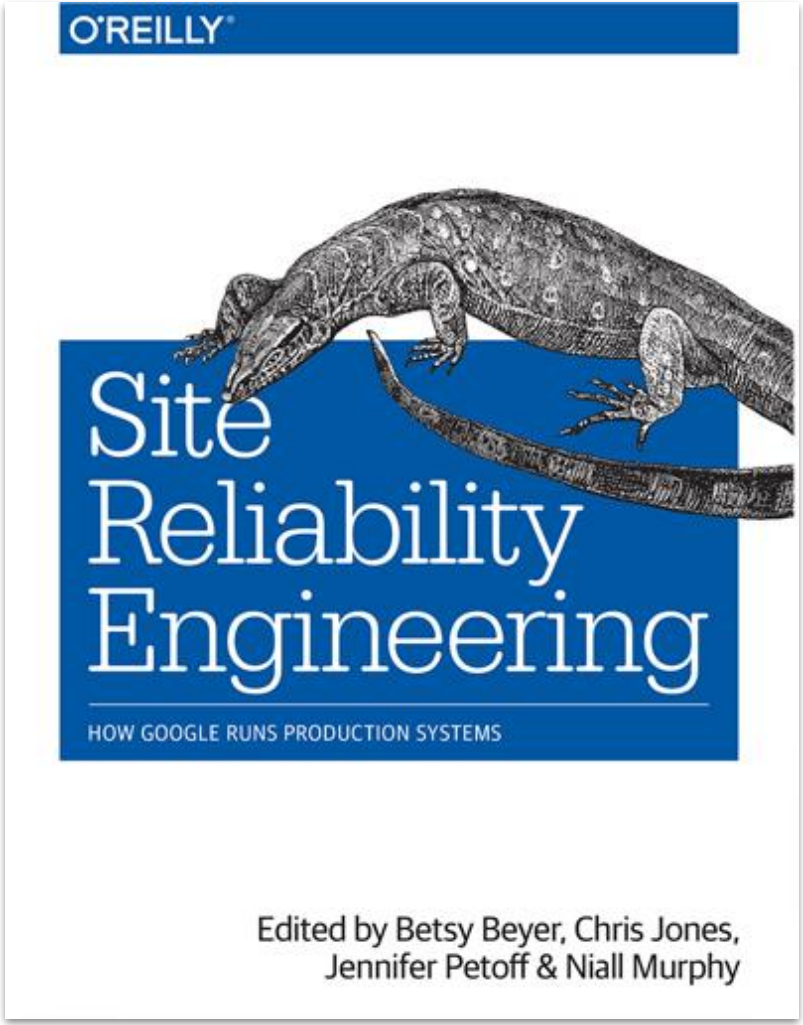
titus@google.com
@TitusWinters

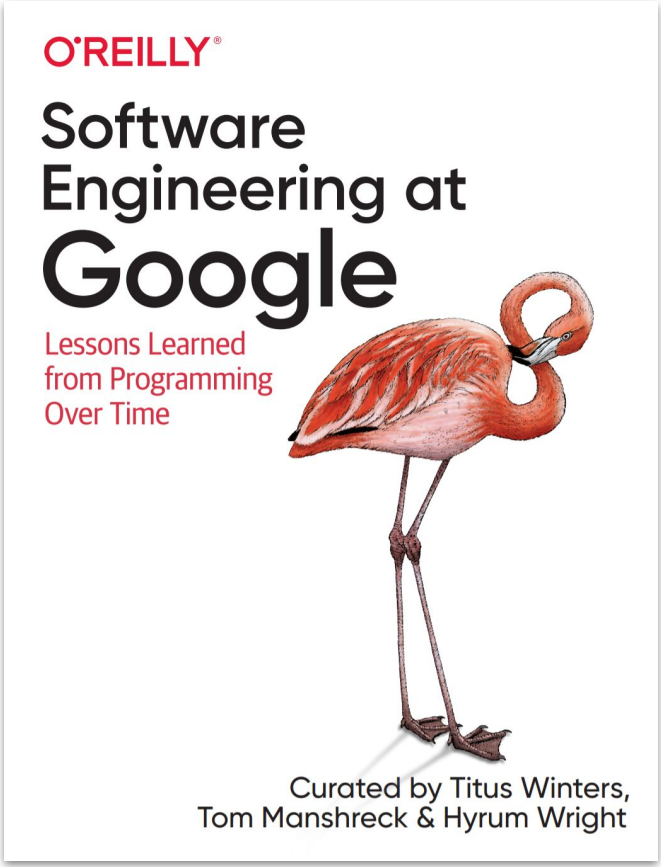
What’s the Secret?

What’s the Secret?

Hint: No Silver Bullet




O'REILLY
Site Reliability Engineering
HOW GOOGLE RUNS PRODUCTION SYSTEMS
Edited by Betsy Beyer, Chris Jones, Jennifer Petoff & Niall Murphy

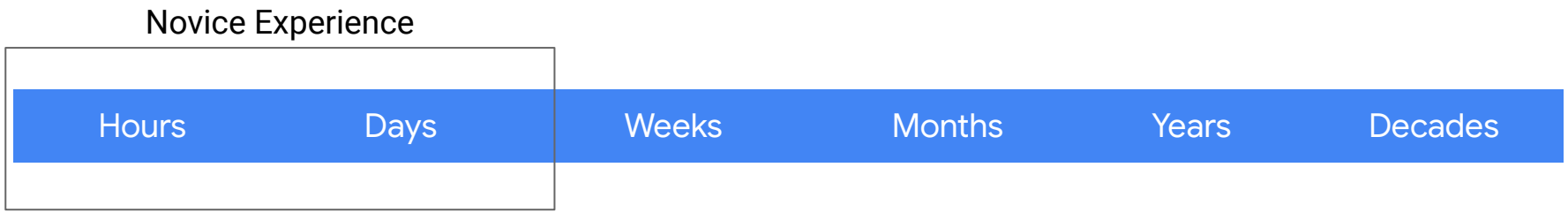


[Software Engineering is] the Multi-Person Construction of Multi-Version Programs.

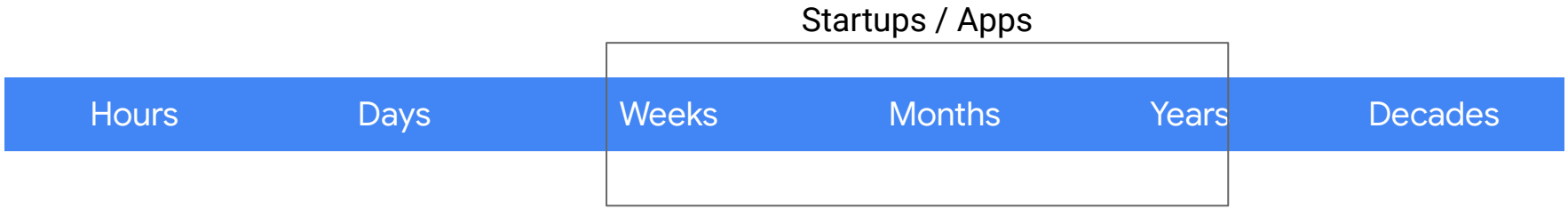
Principle #1 - Time

What’s the expected lifespan of this code?

Principle #1 - Time



Principle #1 - Time

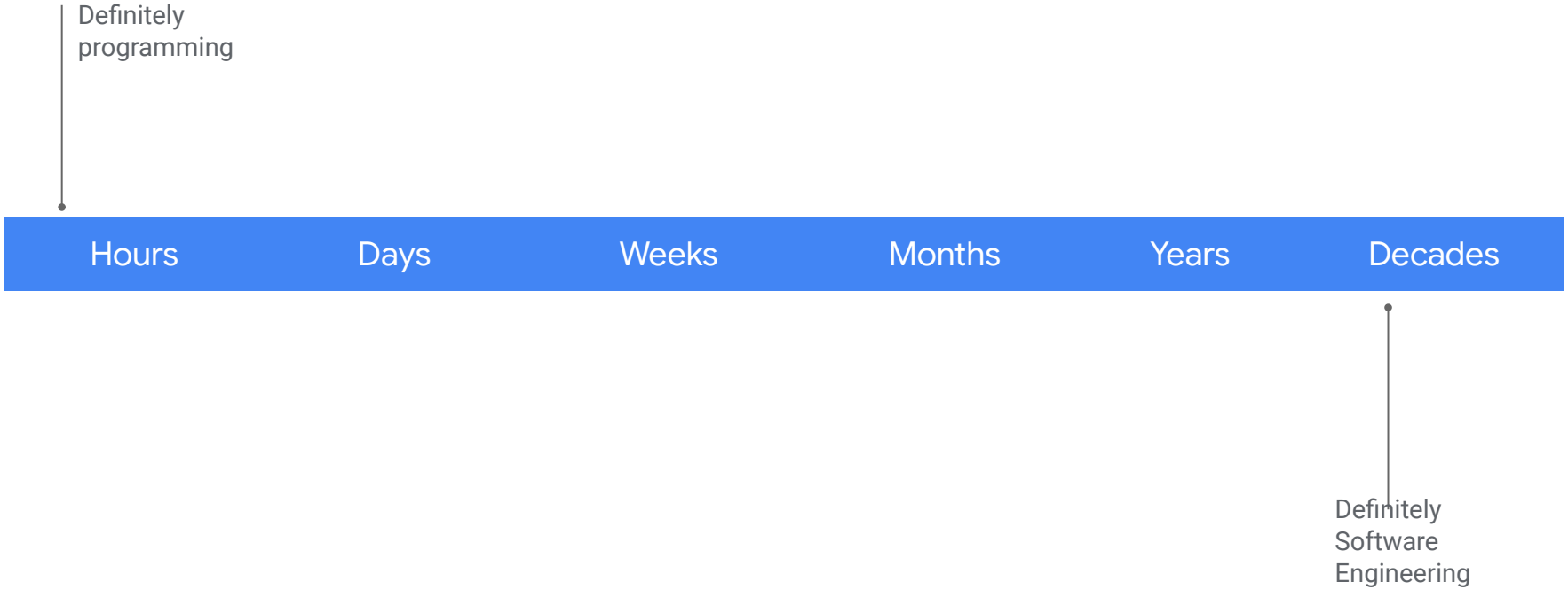


Principle #1 - Time

Principle #1 - Time

Definitely programming

Principle #1 - Time



Principle #1 - Time

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.

Randomness

Multiplicative Difficulty

Overcoming
Hyrum's Law

Multiplicative Difficulty

Overcoming Hyrum's Law

Operating without Policy/Precedent/ Experience

Multiplicative Difficulty

Overcoming Hyrum's Law

Operating without Policy/Precedent/ Experience

Larger than Usual Upgrade

Principle #1 - Time

Sustainability is the goal: for the expected lifespan of your code, you are **able** to change all of the things that you ought to change, safely.

Principle #1 - Time

Many developers have never worked on a sustainable project with a recognized 5+ year lifespan.

Principle #1 - Time

Software Engineering is not merely programming - it is the art of making a program resilient to change over time.

Principle #1 - Time

- Keep in mind the expected lifespan
- Understand that long lifespans are rare, hard to plan for, and not well understood
- Sustainable code is capable of change - that probably means different things at different time scales.
- Sustainable is often hard to get to.

Principles #2: Scale

When change over time leads to growth,

where do we start to fail?

Principles #2: Scale

- Hardware resources (CPU, RAM, Disk, Network)
- Software resources (Addresses, ports)
- Human resources

Principles #2: Scale

Traditional Deprecation:

- **Mark the old version deprecated, introduce a new one, and call it good.**

Principles #2: Scale

Traditional Deprecation:

- Mark the old version deprecated, introduce a new one, and call it good.
- **Mark the old version deprecated, introduce a new one, and mandate everyone update their code by some date. Delete the old one on that date.**

Principles #2: Scale

Traditional Deprecation:

- Mark the old version deprecated, introduce a new one, and call it good.
- Mark the old version deprecated, introduce a new one, and mandate everyone update their code by some date. Delete the old one on that date.
- **Find a brave engineer to go through and build a single change that modifies the API in question and all of the users and land it all in one change.**

Principles #2: Scale

Traditional Deprecation:

- Mark the old version deprecated, introduce a new one, and call it good.
- Mark the old version deprecated, introduce a new one, and mandate everyone update their code by some date. Delete the old one on that date.
- Find a brave engineer to go through and build a single change that modifies the API in question and all of the callers and land that refactoring in one step.

Better Deprecation:

- The team responsible does the bulk of the work.

Principles #2: Scale

In a successful organization, everything that must be done repeatedly* must consume sub-linear resources - especially sub-linear human effort and communication.

Scale: Weekly Merge Meeting

About 1 in 4 SWEs have had a regularly-scheduled meeting to discuss “merge schedule.”

- `git` makes it more common to have heavily-branched workflow
- long-lived dev branches are risky to merge
- manage the risk: merge carefully and rarely

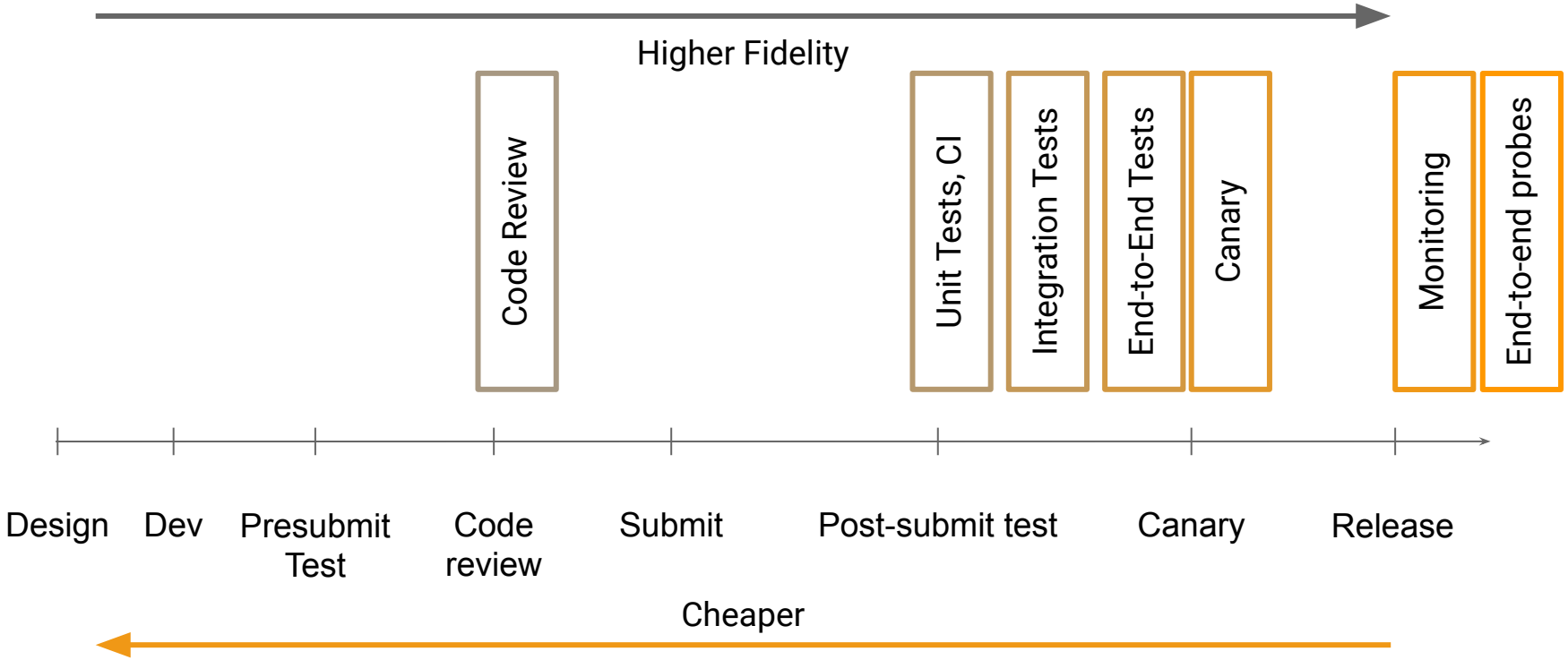
How does this scale?

Scale: No Weekly Merge Meeting

Published research: “Trunk-based development leads to better outcomes.”

- No long-lived dev branches
- No choices where to commit
- No choices which version to depend upon

Time & Scale, Shifting Left



Principle #2 - Scale

- Be mindful of superlinear scaling costs
- Anything that must be done repeatedly by humans should be sub-linear
- Expertise and automation usually pay off super-linearly
- The “normal” way of doing things may have scaling problems.

Principles #3: Tradeoffs

Make evidence-based decisions.

Principles #3: Tradeoffs

Make evidence-based decisions.

Aim for sustainability.

Principles #3: Tradeoffs

Make evidence-based decisions.

Aim for sustainability.

No super-linear scaling.

(Especially for humans.)

Principles #3: Tradeoffs

Make evidence-based decisions.

Aim for sustainability.

No super-linear scaling.

(Especially for humans.)

Re-evaluate as needed.

What’s the Secret?

Google SWE “Secrets”

1. Software Engineering is more than just programming, it’s Time
    - Especially consider the impact of time
2. Be mindful of scale
    - Super-linear scaling is bad for required processes
    - Expertise/specialists can provide super-linear impact in their domain
3. Make evidence-based decisions
    - No “because I said so”
    - Evidence will change over time, re-evaluate as needed

Google SWE Book

- Pillars (these)
- Culture (happy devs, productive teams)
- Policies and Processes (how to make things work smoothly)
- Tools (tech)

Google SWE “Secrets”

It’s programming if “clever” is a compliment.

It’s software engineering if “clever” is an accusation.