Configuration, Extension, Maintainability

@TitusWinters

Higher Level Abstractions

Express Intent

Centralize.
Improve Consistency.

How Are CPU vs. RAM costs evolving?

CPU is Expensive?

Update vector's resize factor. Maybe 3x?

How Are CPU vs. RAM costs evolving?

CPU is Expensive?

Update vector's resize factor. Maybe 3x?

Fewer cycles spent moving elements around.

RAM is Expensive?

Shrink vector's resize factor. Maybe 1.5x?

Less waste.

Improve through Refactoring

Optimization through Refactoring

Centralized?

- Optimize the memory allocator
- Change vector or string allocation strategies
- Tweak the network stack
- Optimize mutex

Change implementation of existing abstractions

Optimization through Refactoring

Centralized?

- Optimize the memory allocator
- Change vector or string allocation strategies
- Tweak the network stack
- Optimize mutex

Change implementation of existing abstractions.

Distributed?

- Change from standard hashes to Abseil
- Use `string_view` more consistently
- Remove redundant string copies
- Add missing calls to `std::move`

Change **many** small things given a general pattern.

Bad Abstractions
lead to
Bad Optimization

Configuration is an Abstraction

Which is Easier to Optimize?

FooSystem - works on `int`, `double`, `string`

BarSystem - works on `T`

Socket Reader 1 - allocates 64K

Socket Reader 2 - allocates user-provided size

User Requests

What is the Right Size for this buffer?

What is the Right Size
for this buffer?
In 1990?
In 2050?

The Tradeoff:
Support more uses
vs
Retain more flexibility

Three Forms:

- Configuration
- Feature-flags
- Extension

My Design Philosophy

“We Could Build X!”

“We Could Build X!”
A rarely-used feature
is a liability.

Hyrum’s Law Applies (always)

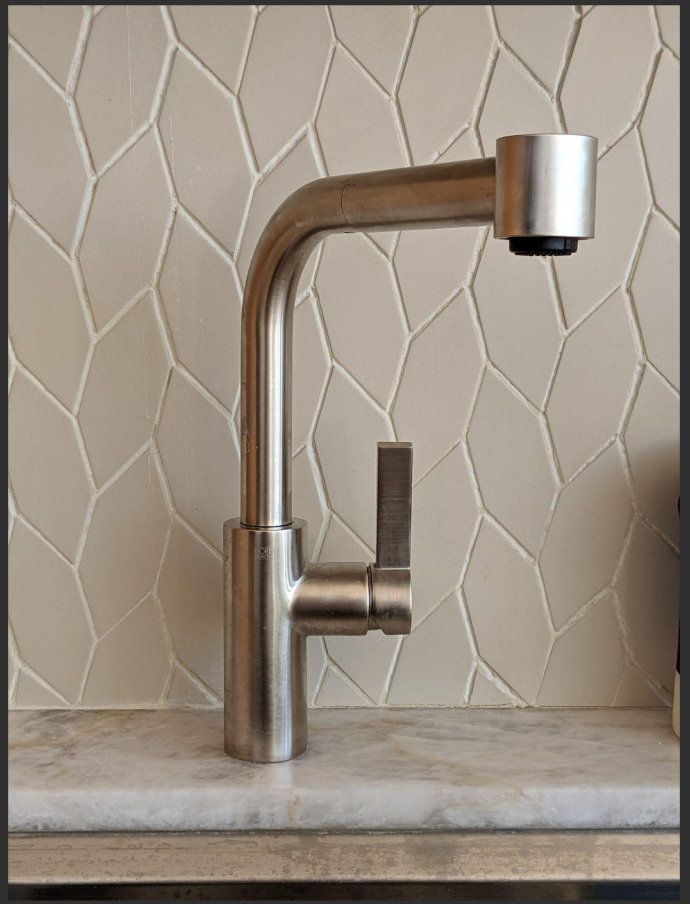
Compile Time Detection Is Good

Compile Time Detection Is Good (Shift Left)

Configuration

Configuration

Controlling Outputs



Configuration

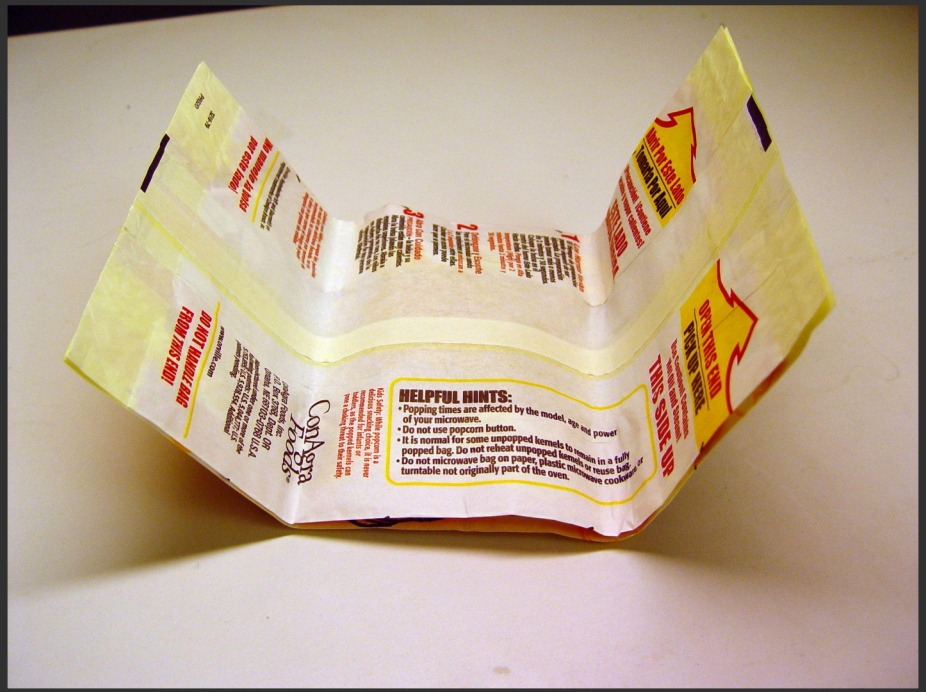
Controlling Outputs

Configuration

Controlling Inputs?

Configuration

Controlling Outputs?



Configuration

Controlling Outputs?

Popcorn Button is Bad **Today**

Do it manually

- I know better than the hardware

Use the Button

- Worse outcomes today
- Maybe it’ll get better?

Configuration - What Seems Good?

- We like to specify **outcomes**
  - We will settle for specifying ***inputs*** if those clearly map to outcomes
- False control, or low-quality outcomes are confusing

Configuration (now with C++)

Compiler Configurations: -O, -W

Configuration for Stream / Sequence Reader

- memory_budget
- seek_back
- enable_async_io
  - buffer_size
  - lookahead_budget
  - prefetch_on_open

Configuration for Stream / Sequence Reader

- memory_budget
- seek_back
- enable_async_io
  - buffer_size
  - lookahead_budget
  - prefetch_on_open

What **Outcomes** Do We Want?

Proposed Configuration for Stream / Sequence Reader

- optimize_for = {kCPU, kIO_Ops, kMem};

Proposed Configuration for Stream / Sequence Reader

- optimize_for = {kCPU, kIO_Ops, kMem};
- max_memory
- record_size_hint
- max_prefetch_threads

Time will cause
Change

Change vs. Outcomes-based Configuration

- Maintainer responsible for honoring **intent** (and semantics)
- Changing **implementation** is more allowed

Changes vs. Granular Configuration

- We are leaking implementation details
- Users are maybe depending on those
- Users may not be expert in this configuration

Changes vs. Granular Configuration

Out of 13K uses:

- 100 set the value at all

Changes vs. Granular Configuration

Out of 13K uses:

- 100 set the value at all
- 1 sets it to 256 **bytes**

Changes vs. Granular Configuration

Out of 13K uses:

- 100 set the value at all
- 1 sets it to 256 **bytes**
- 1 sets it to 256 **MB**, **N** times

Users vs. Granular Configuration

- How often is this configuration based on evidence/optimization?
- Is that evidence still valid?
  - How do we know?
- How many “power users” are highly sensitive to this configuration?
  - Does their need dominate?

Users vs. Granular Configuration

r.memory_budget = k256MB;

Users vs. Granular Configuration

num_threads = 8;

Are All Knobs Bad?

Configuration Should Be

- Orthogonal
- Focused on outcomes/intents
- Minimal
- Easy to reason about

Migration

Migration: Changing Defaults

```
std::cout << absl::StrCat(SomeDouble());

std::cout <<
  absl::StrCat(LegacyPrecision(SomeDouble());
```

Migration: Changing Defaults

```
# Visibility:  please choose a more appropriate default for the package,
# and update any rules that should be different.
package(default_visibility = ['//visibility:legacy_public'])
```

Side-note: “Legacy” naming

“You probably don’t want this setting.”

- Might be around forever as “old behavior”
- Here temporarily, but don’t use it.

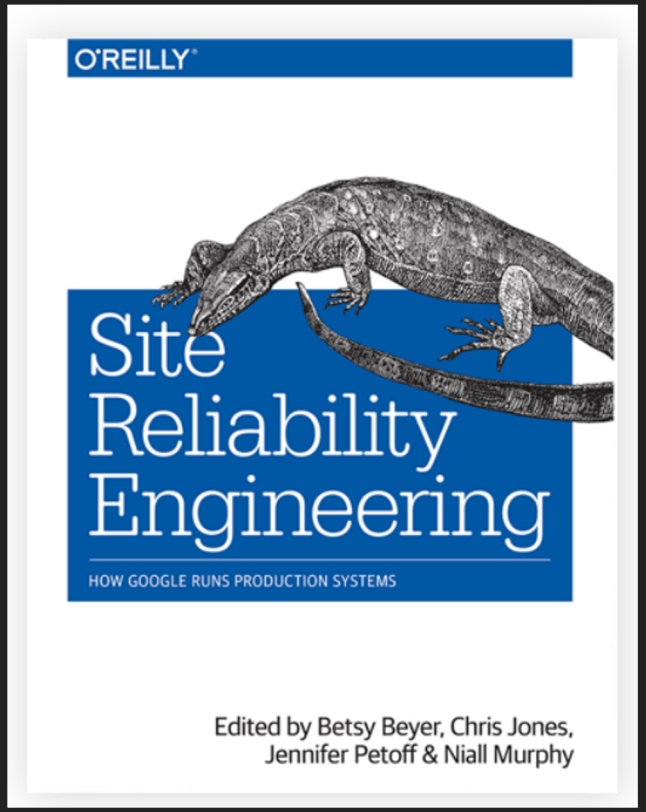
Avoid if you endorse the behavior.

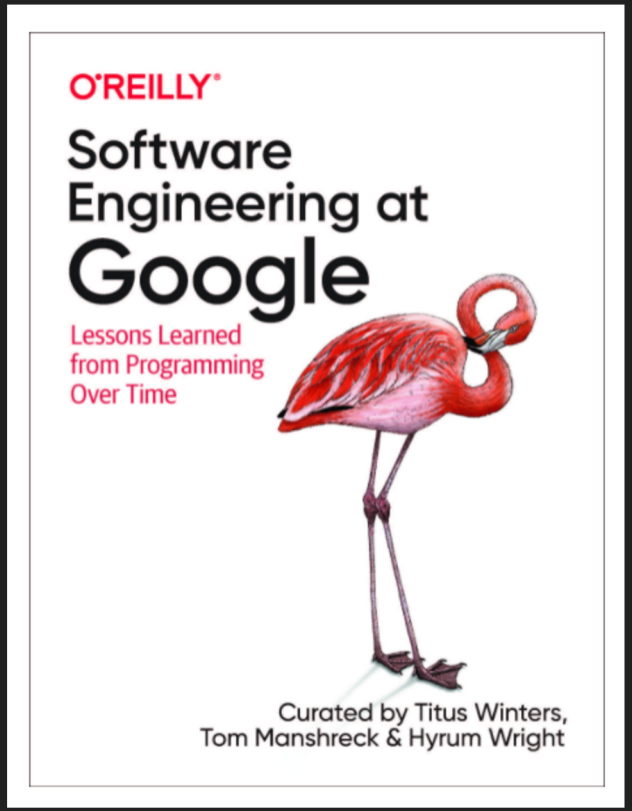
Configuration Should Be

- Orthogonal
- Focused on outcomes/intents
- Minimal
- Easy to reason about

Experimentation, Release

Experimentation, Release





Experimentation, Release

- Functionality gated by feature flags/configuration
- Management of that flag is controlled by
  - Release engineers
  - Experimental frameworks (A/B tests)
  - Rollout systems

Configuration Should Be

- Orthogonal
- Focused on outcomes/intents
- Minimal
- Easy to reason about

For release/experiments: Clean up!

Configuration Gotchas

- Platform properties?
  - `std::hardware_destructive_interference_size`
- Define acceptable changes?
  - Predict acceptability

Extension

Callbacks

Be *very* precise about how you will invoke a callback.

- Which thread(s)?
- Locks held?
- Order of invocation?
- Frequency of invocation?

Polymorphism

- Avoid?
- PIPML?
- Proceed very carefully.
  - An abstract interface is both requirements and affordances - these are hard to change.
  - ABI lurks here in more ways.

Templates, Extension Points, etc

- Proceed with care
- Document **intent**
- Concepts may help

Templates, Extension Points, etc

`std::accumulate`

“Can we change this to rely on move where appropriate?” (Yes)

Templates, Extension Points, etc

Abseil Command Line Flags

- `AbslParseFlag` / `AbslUnparseFlag`

Conclusions

Conclusions

- Configure based on outcomes and intent

Conclusions

- Configure based on outcomes and intent
- Customization fights optimization/maintenance

Conclusions

- Configure based on outcomes and intent
- Customization fights optimization/maintenance
- Extensible interfaces are hard to get right
  - And very hard to change after the fact

Conclusions

- Configure based on outcomes and intent
- Customization fights optimization/maintenance
- Extensible interfaces are hard to get right
  - And very hard to change after the fact
- The Popcorn button is a trap

Questions?