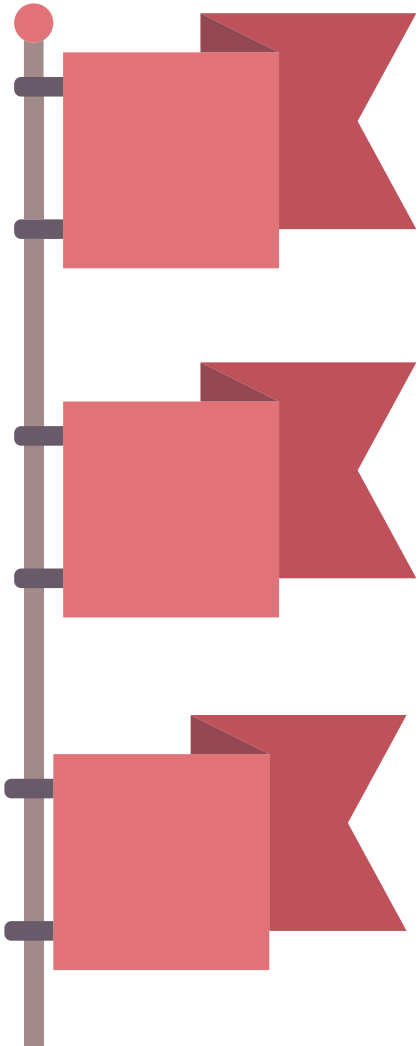


Intentional Deployment

Best Practices for Feature Flag Management

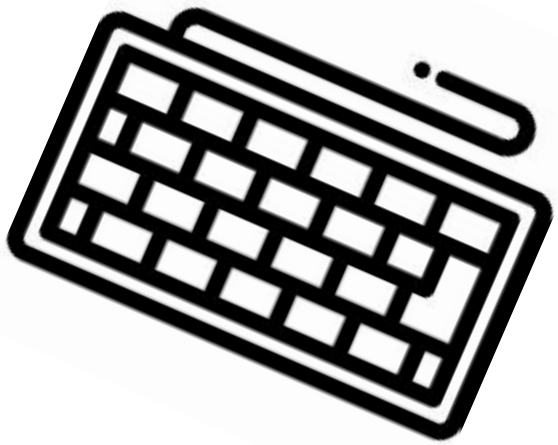
Caitlin Rubin

Intention.

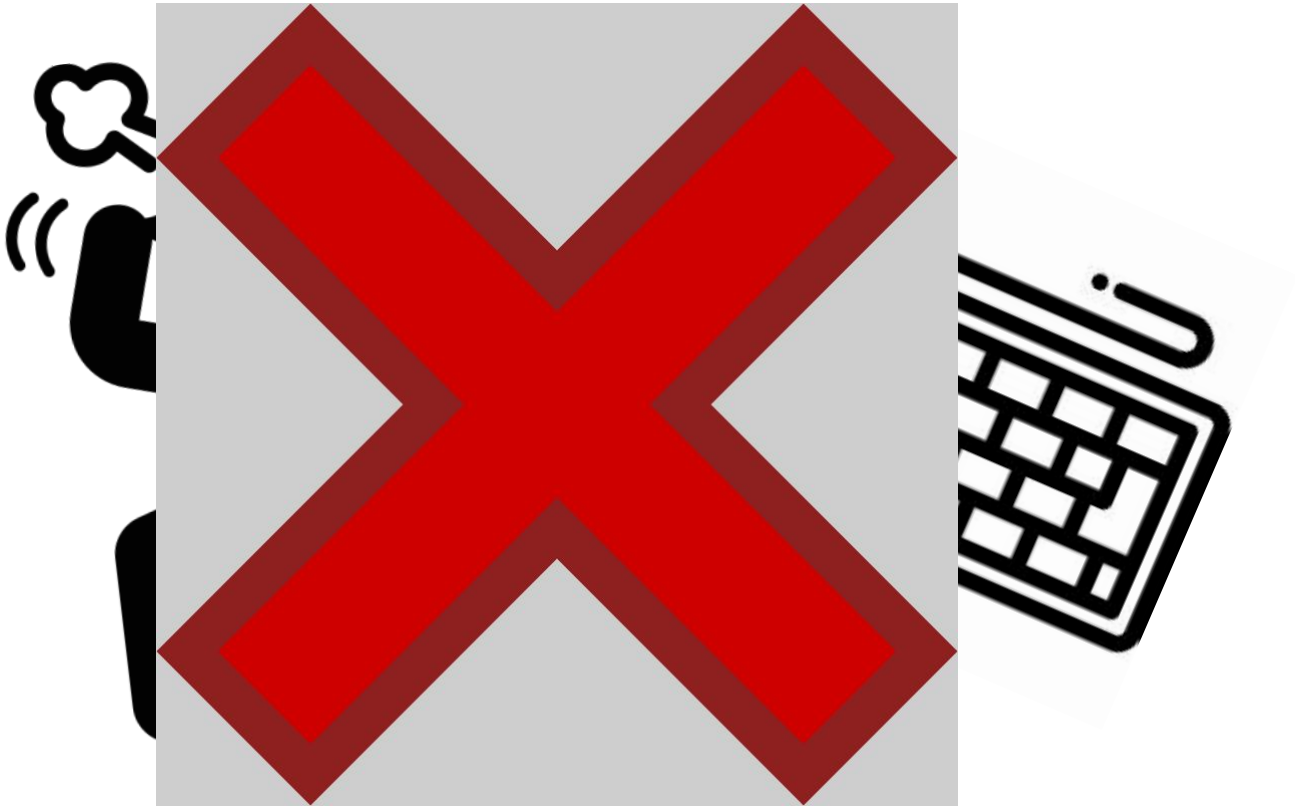
Intention.

Doing what you *meant* to do.

We (usually) write code with intention.

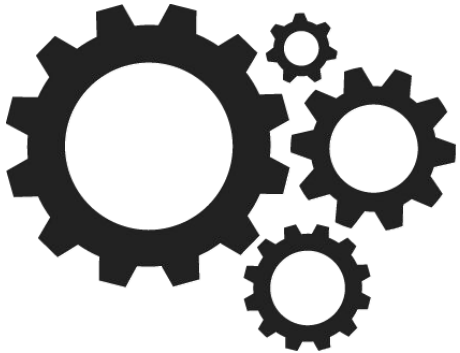


We (usually) write code with intention.

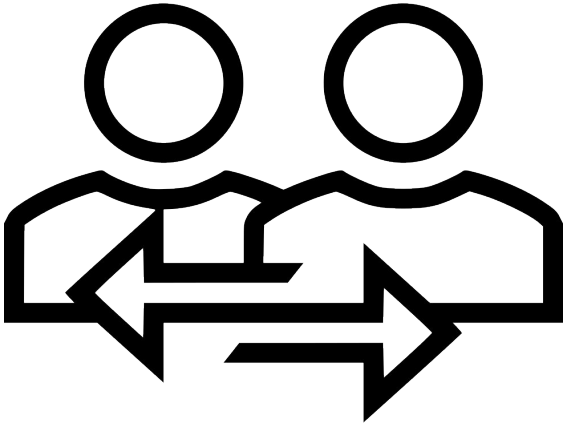


Things we do with intention:

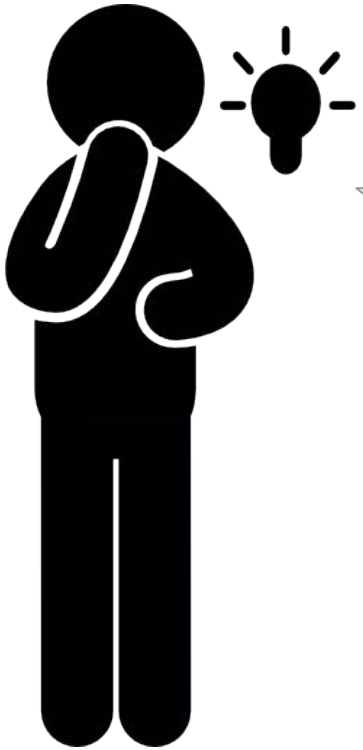
Writing code



Testing Code



Reviewing Code

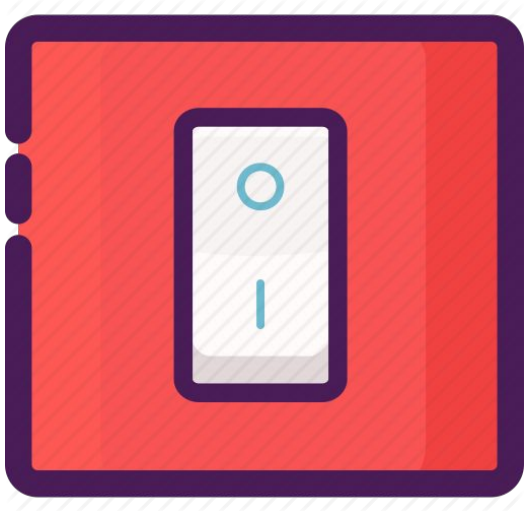


“ Let’s be *intentional* when we deploy our code into the world. ”

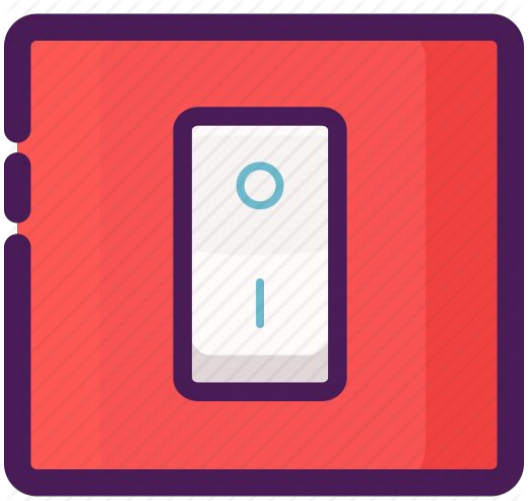
What’s an easy way to do this?

Feature flags.

Introduction: Feature Flags



Introduction: Feature Flags



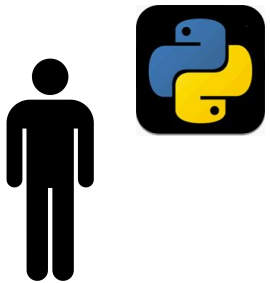

```
if feature_flag is 'on':
    do_something()
else:
    do_something_else()
```

What does that process look like?

Your code.

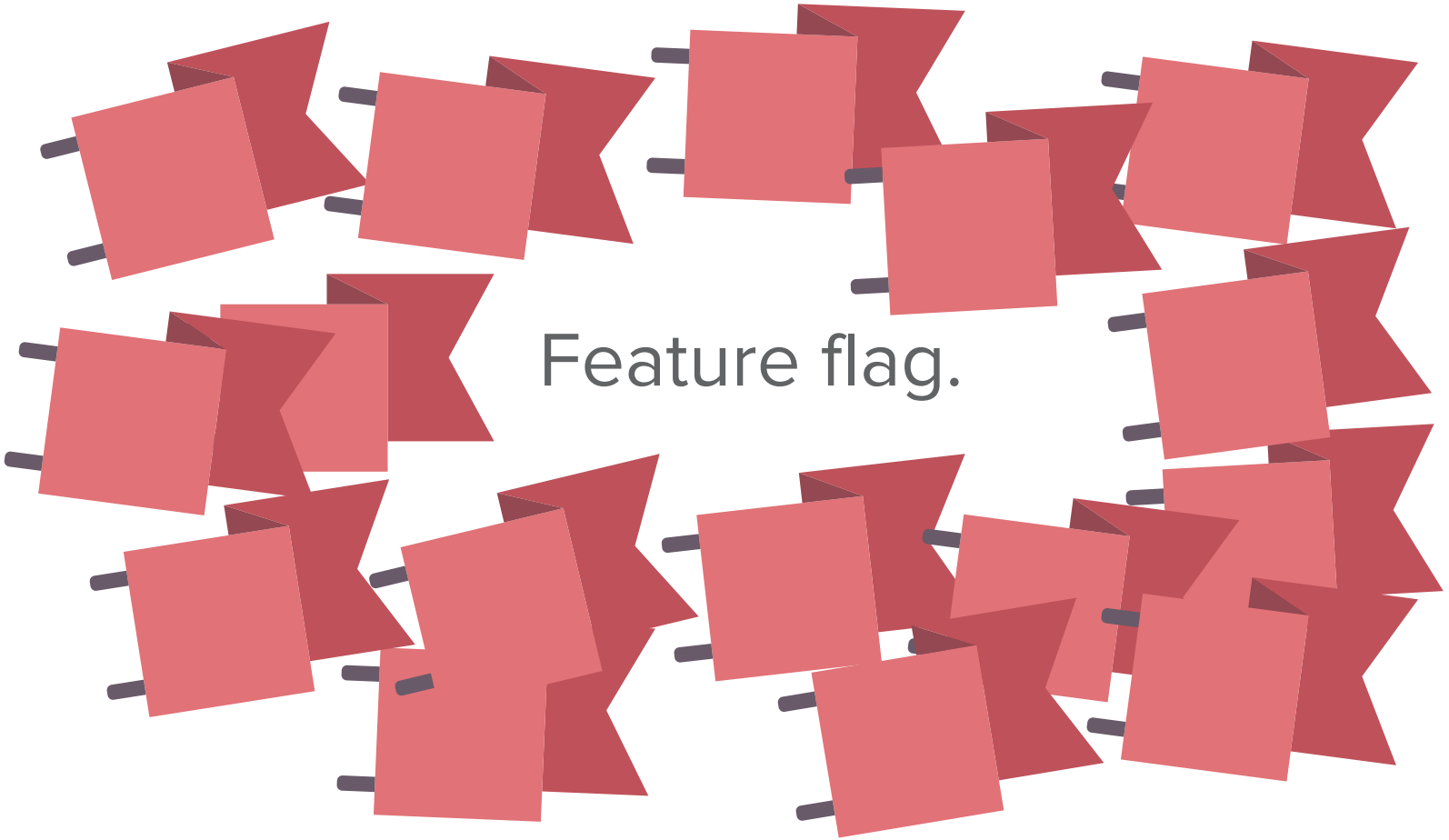
Feature flag.

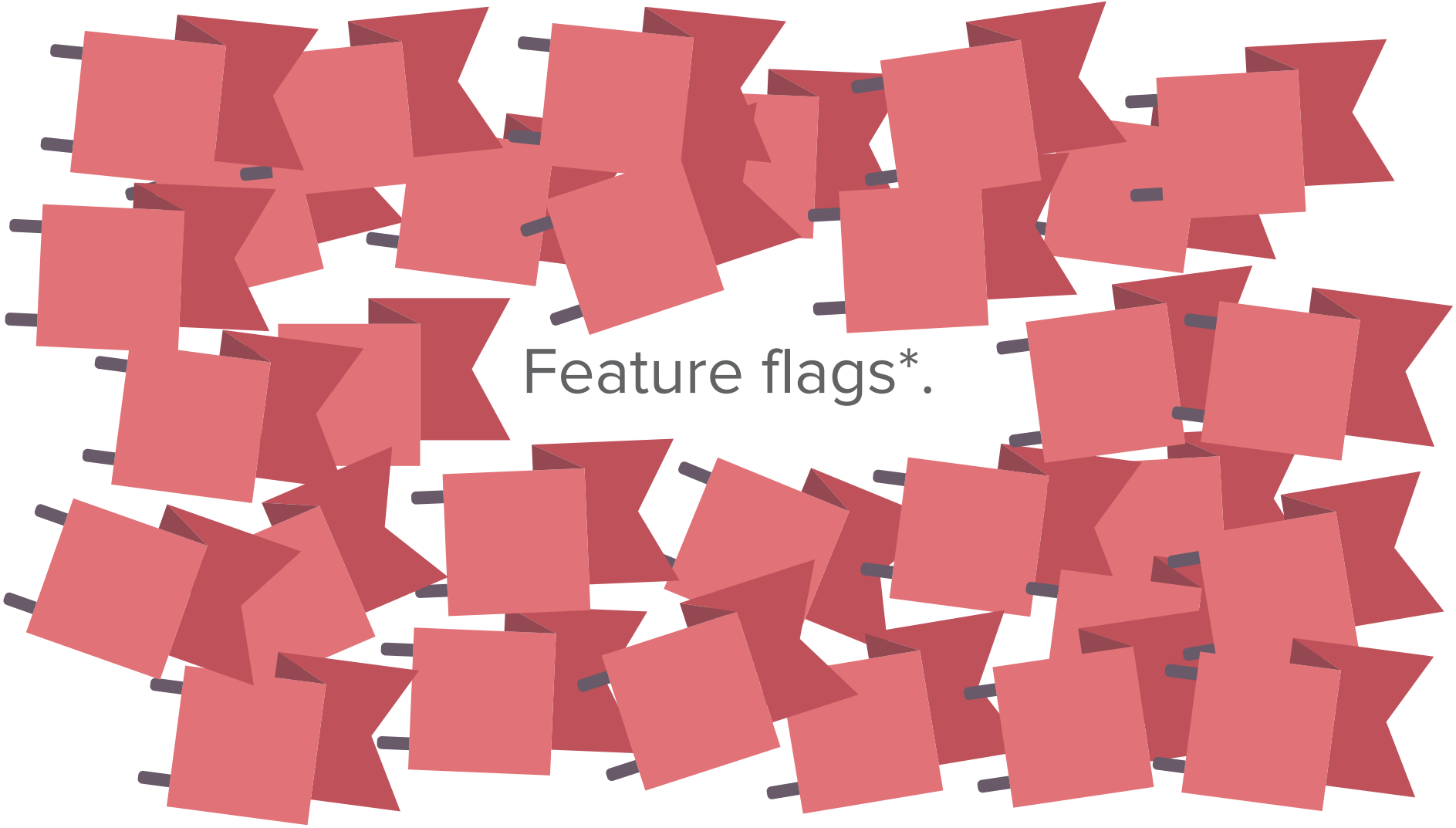
Feature flag.



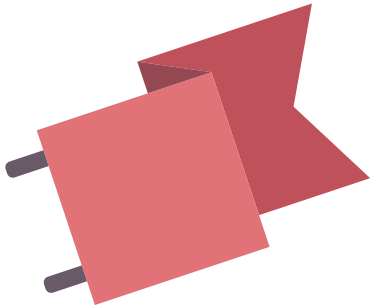
Feature flag.

Feature flag.


Feature flag.

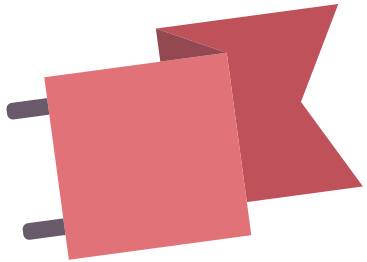

Feature flags*.

Release toggle.

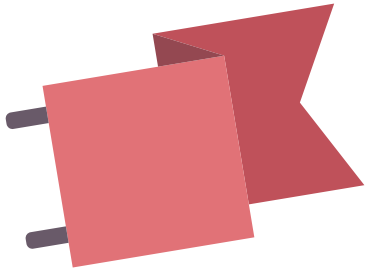


Kill switch.

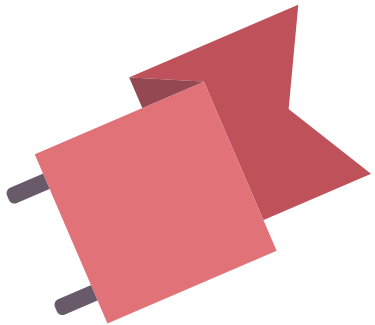
Experiment.

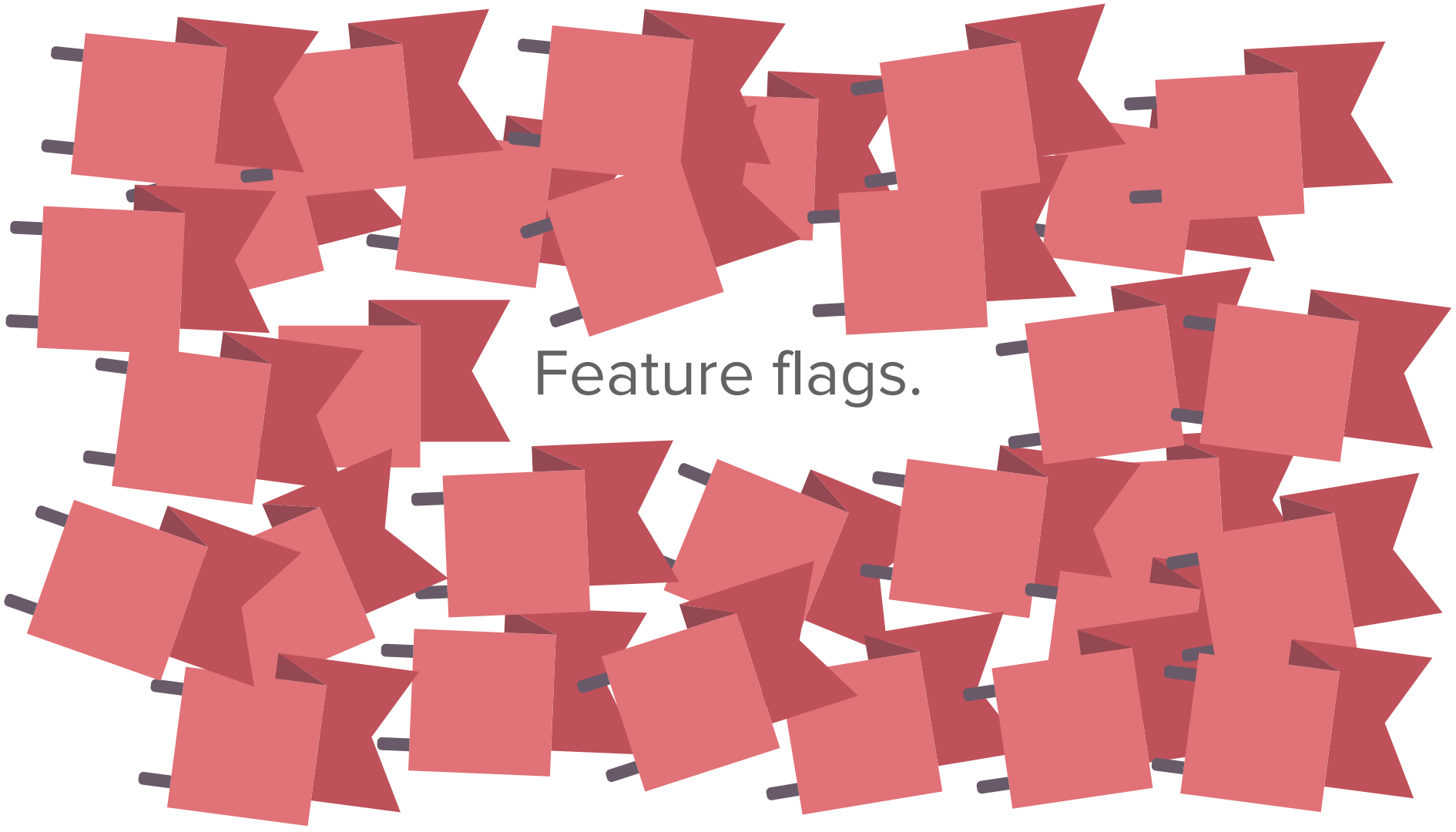


Opt In.



Opt Out.

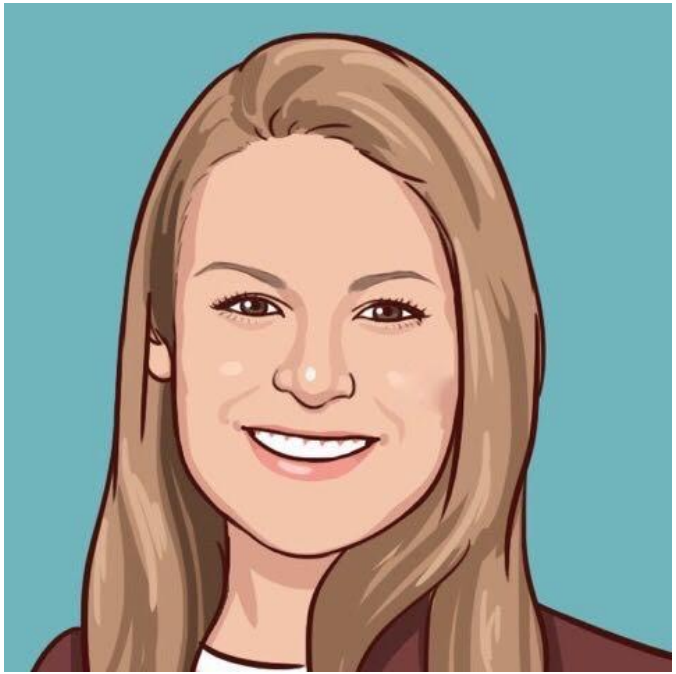



Feature flags.

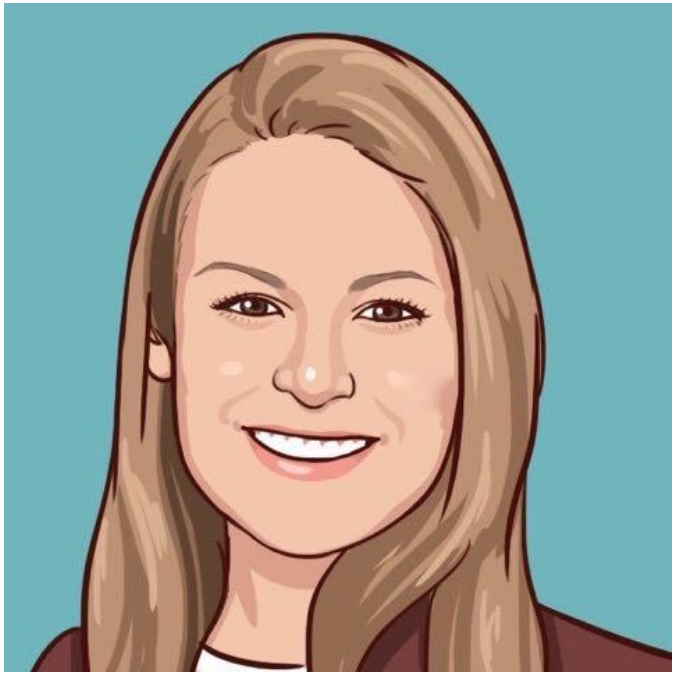
How do we fix that?

What’s the best way to do feature flags?

Introduction: Me

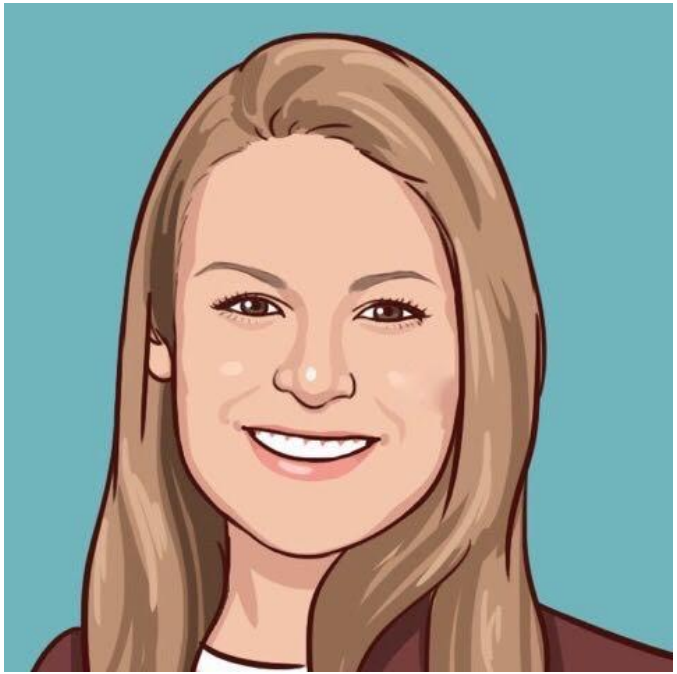


Introduction: Me



Software Engineer

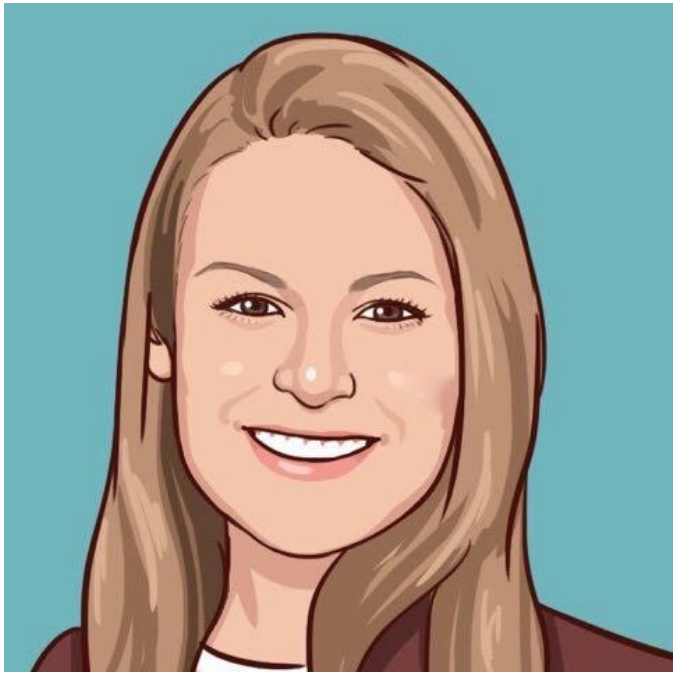
Introduction: Me



Software Engineer

Optimizely

Why should you listen to: Me



Software Engineer

...

works on feature flagging product

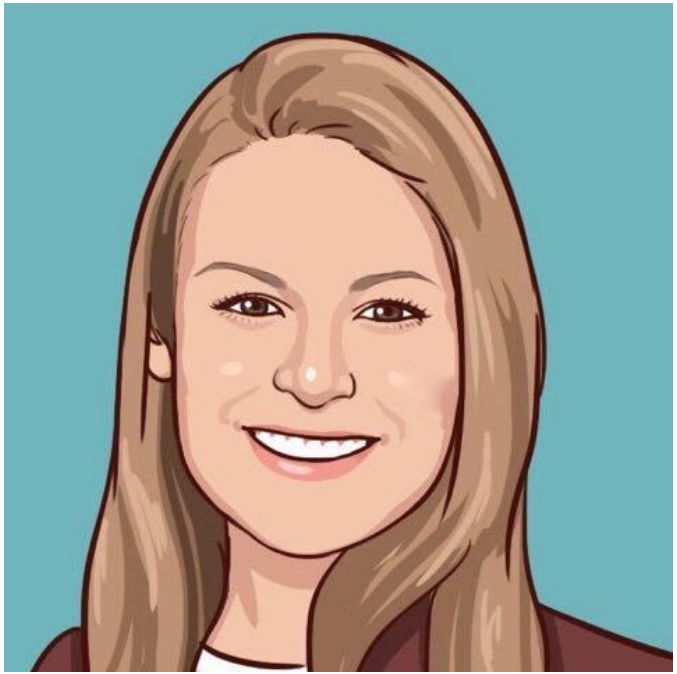
uses feature flags in development

hacks on feature flags

talks to people who use feature flags

...

Who else is in this talk?



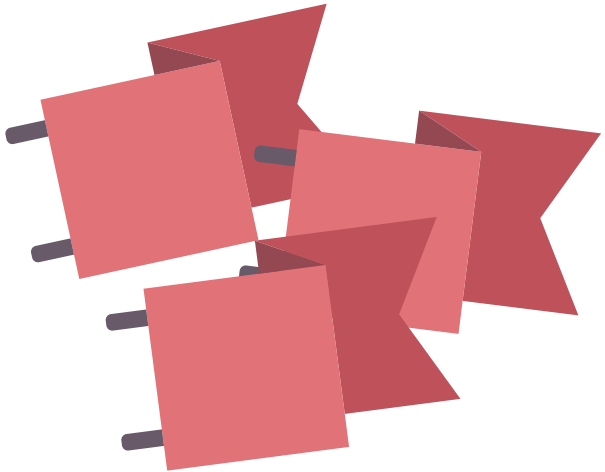
Software Engineer

+

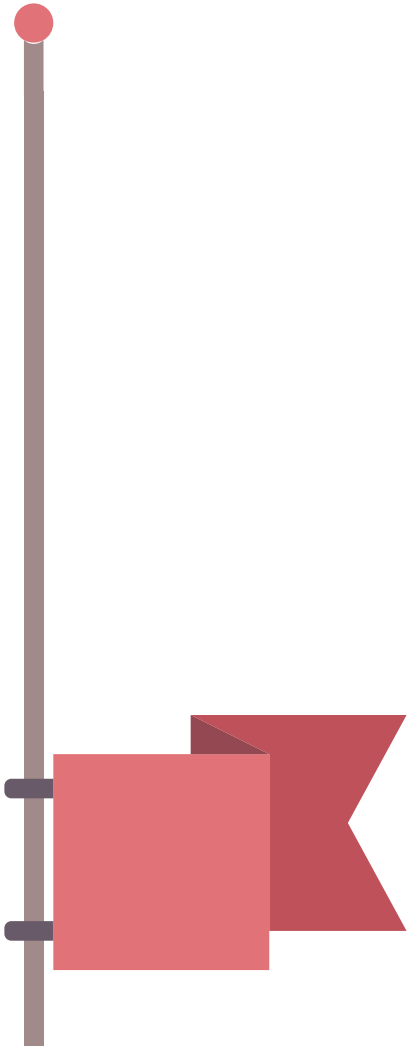
2 Backend Software Engineers
2 Frontend Software Engineers
1 SDK Software Engineer
2 PMs
1 Director of Development
1 Sales Engineer
1 Solutions Architect
2 DevOps Engineers
1 Quality Assurance Engineer

Storytime

What goes wrong when we combine feature flags and people?



+



Lowest Risk

Tech Debt and Confusion

Increases complexity and tech debt and makes it harder for engineers to understand code

```
if feature_flag is 'on':
    do_something()
else:
    do_something_else()
```

vs

```
do_something()
```

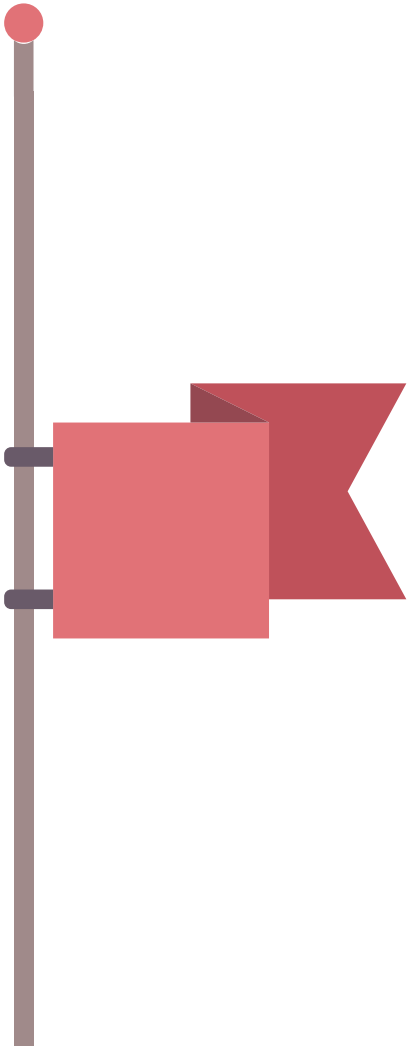
40+ instances of feature flags in our codebase, each gating 1000s of lines of code.

Taking out a feature sucks...

Taking out a feature sucks...

Taking out a feature sucks...

60+ days in the codebase



Higher Risk

Performance Impacts on Your System

Performance Impacts on Your System

What happens when...

Your feature is broken.

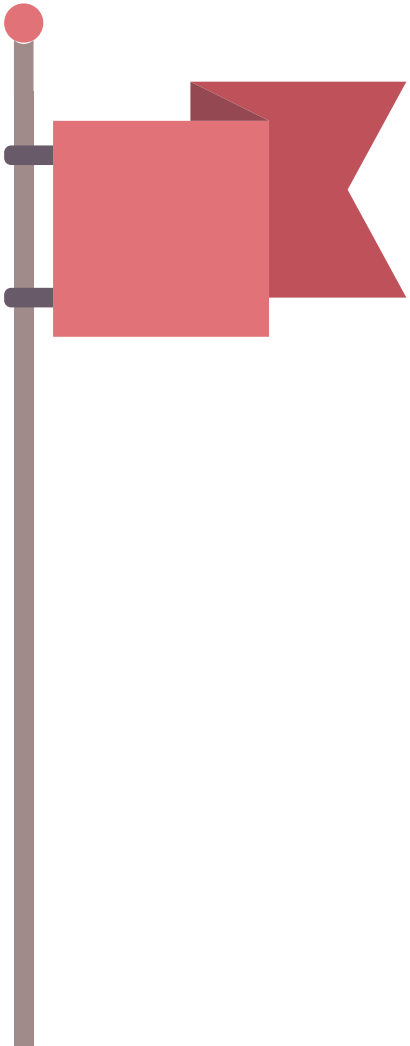
Your feature tanks.

Performance Impacts on Your System

An improperly monitored feature flag once added a performance hit of

48 seconds

to one of our API endpoints.



Highest Risk

Disaster

Knight Capital Group

- Financial services group
- Largest trader in US equities in 2012, trading $21b daily
- Held $365 million at 9:30am, August 1, 2012

Source

Disaster

Knight Capital Group

- Financial services group
- Largest trader in US equities in 2012, trading $21b daily
- Held $365 million at 9:30am, August 1, 2012

By 10:15am KCG was bankrupt:
$460 million dollar loss

Source

Disaster

Knight Capital Group

- Deployed code which repurposed an old feature flag, but accidentally did not remove the old code
- The now “on” feature flag activated this old code
  - Old code did not have validation to stop it processing orders -- and this ran for 45 minutes

Source

Disaster

Knight Capital Group

Intentional Deployment

Intentional Deployment

How can I mitigate those risks by being intentional about *how* I deploy my code using feature flags?

Intentional Deployment

How can I mitigate those risks by being intentional about *how* I deploy my code using feature flags?

You used a feature flag because you wanted to be intentional when you wrote your code.

Intentional Deployment

How can I mitigate those risks by being intentional about *how* I deploy my code using feature flags?

You used a feature flag because you wanted to be intentional when you wrote your code.

Why aren’t we being intentional about controlling those flags?

Best Practices.

Visibility + Accountability

Visibility.

Stalk your flags.

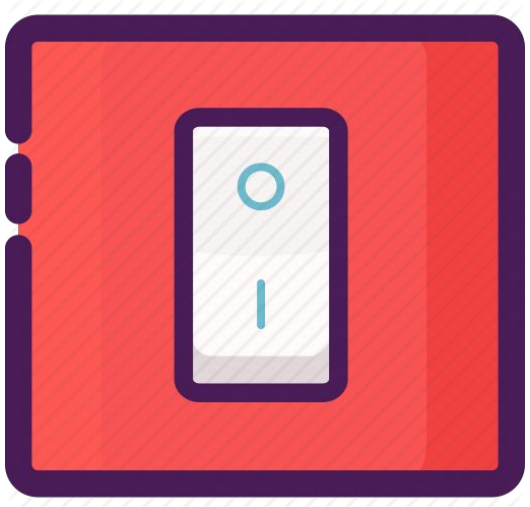
Visibility

For every flag, you should know:

- State
- History
- Metrics
- Expiration

State

What is the state of the flag?



History

For every flag:

- When did it last change?
- Why did it change?
- Who changed it?
- What's the history of the flag?
  - On/off?
  - Rollout percentage?
  - Targeted audience?

Metrics

For every flag:

- Is it being used?
  - How many times has it been evaluated?
  - When was the last time it was evaluated?
- What metrics are tracking its success/failure?

Expiration

For every flag:

- When does this flag need to be removed?
- Who’s responsible for taking it out?

Bird’s Eye View

For your entire system:

- How many flags are in the codebase?
- Where are they in the code?
- What percentage of your flags are rolled out to X%?
  - 100%?
  - 0%?
- Which flags are expired / expiring in X days?

Collaboration.

Other people might be using your flag.

A feature flag can turn code into a business decision.

A feature flag can turn code into a business decision.

You might not be the one flipping the switch.

Collaboration

Collaboration

What needs to be gated behind this flag?

- Docs?
- Marketing material?

Collaboration

What needs to be gated behind this flag?

- Docs?
- Marketing material?

Who’s controlling the flag?

- Developer, release team, PM, etc.

Collaboration

What needs to be gated behind this flag?

- Docs?
- Marketing material?

Who’s controlling the flag?

- Developer, release team, PM, etc.

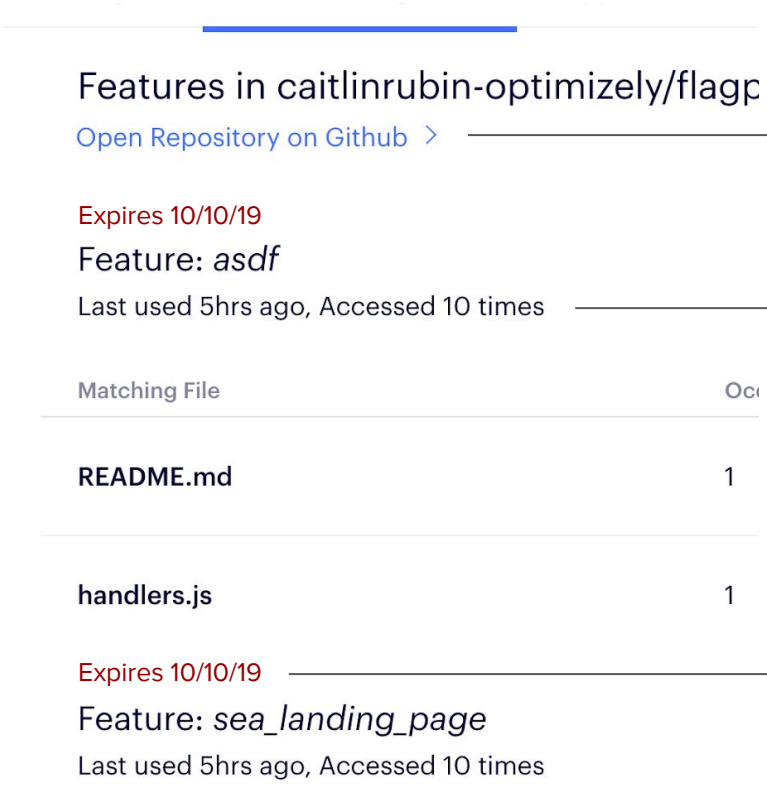
Who needs to be notified of this change?

Low Tech

Find your flags: `git grep`

Track your flags: shared spreadsheet

High(er) Tech



Where is the flag

Is it being **used?**

When does it **expire?**

Accountability.

Be accountable for your flags.

Accountability

For every flag:

Accountability

For every flag:

- Tracked in a **shared place**

Accountability

For every flag:

- Tracked in a **shared place**
- Alerting set up on your monitoring/metrics

Procedure.

Decide as a team or company:

Who owns managing your features?

- Release team?
- PM?
- Engineering manager?

WIP Limit

Are you going to limit the number of features/feature flags in the codebase?

WIP Limit

Are you going to limit the number of features/feature flags in the codebase?

- Consider feature flags *work in flight*
- How much work in flight do you want to have at any given time?

WIP Limit

Are you going to limit the number of features/feature flags in the codebase?

- Consider feature flags *work in flight*
- How much work in flight do you want to have at any given time?
- The people feeling the consequences of tech debt are not always the people prioritizing work

Feature Flag Checklist

What's the list of *must-do's* before someone implements a feature?

Feature Flag Checklist

What's the list of *must-do's* before someone implements a feature?

- What tracking do they need?
- What's the procedure they need to follow?
- What kind of changes *should not* be behind a feature flag

Lifespan

How long can any type of feature flag stay in your codebase?

Lifespan

How long can any type of feature flag stay in your codebase?

- Short lived flags: 1 day? 1 week? 1 month?
- Long lived flags?

Lifespan

How long can any type of feature flag stay in your codebase?

- Short lived flags: 1 day? 1 week? 1 month?
- Long lived flags?

Depends on

- How long does your system take to get meaningful results?

Expired Features

Expired Features

- Prioritize removing flags as **part of the work of the feature**

Expired Features

- Prioritize removing flags as **part of the work of the feature**
- Retire flags which are expired or unused

How do I make sure expired features are removed?

- Feature stand-up
- Ticket flag removal when the feature is created
- Write the PR to take it out *when it goes in*

Procedural Change

Like with any procedural change, it’s important that you:

Procedural Change

Like with any procedural change, it’s important that you:

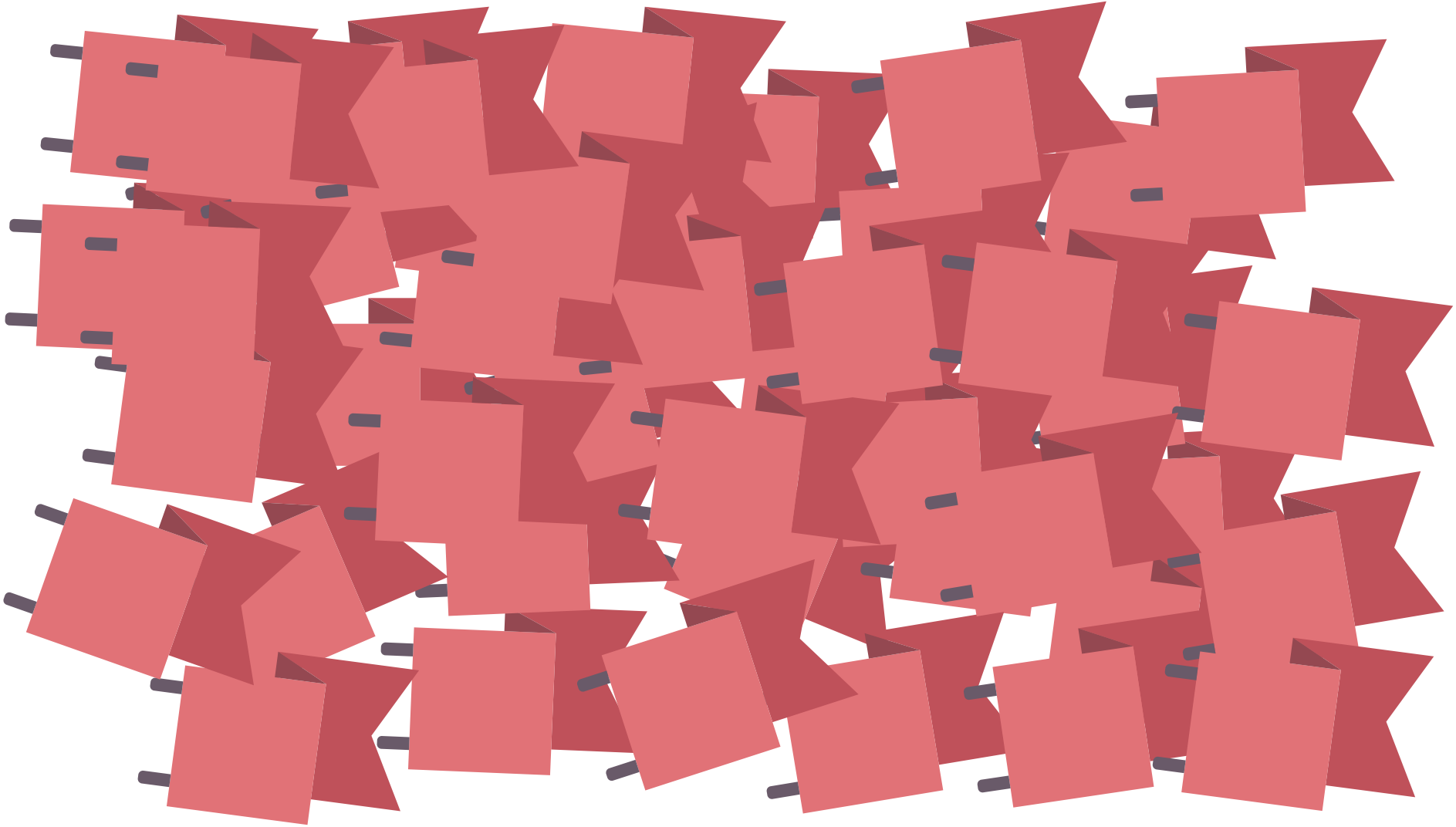
- Ensure review
- Gain buy in
- Communicate widely

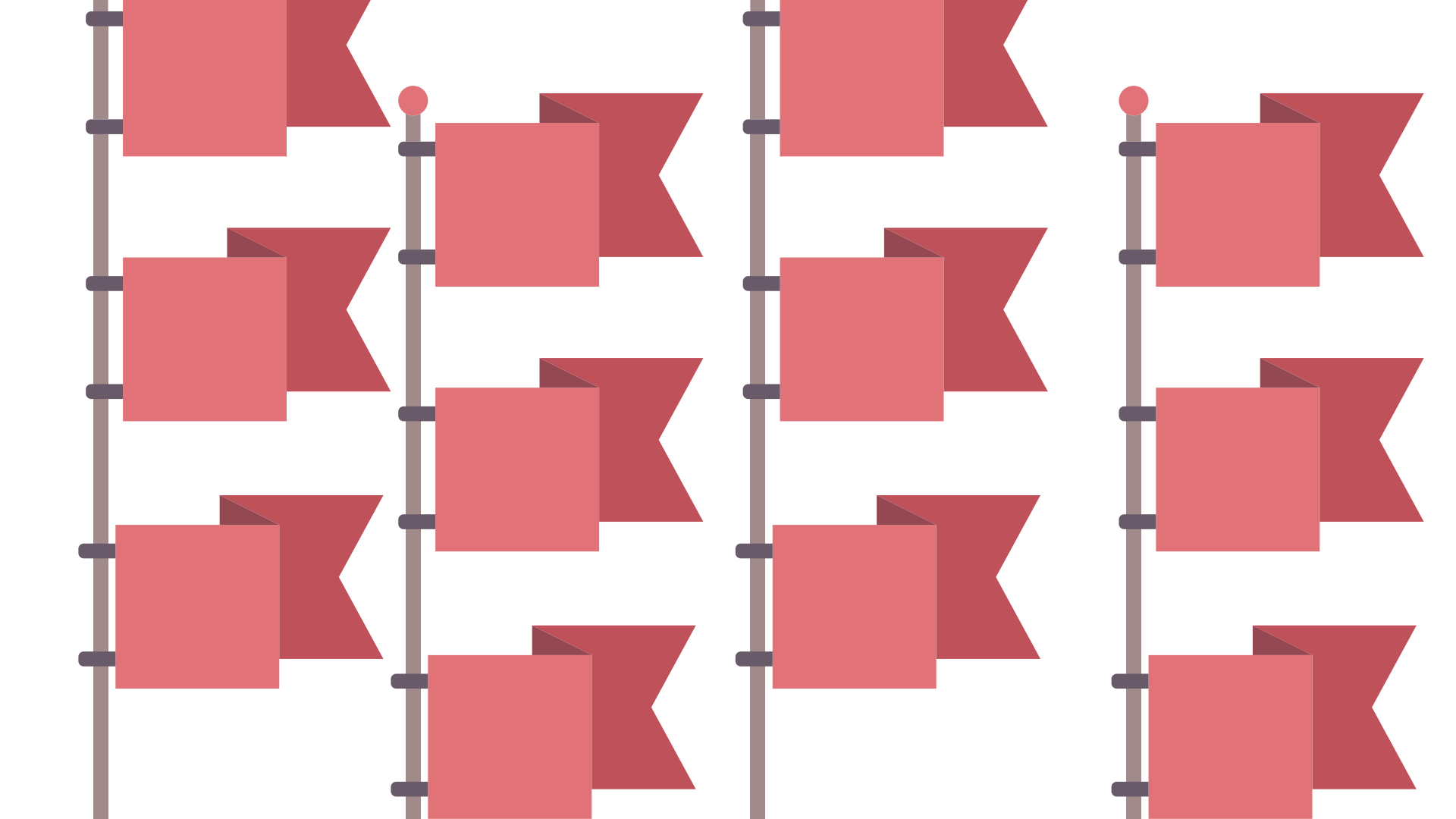
Other tips

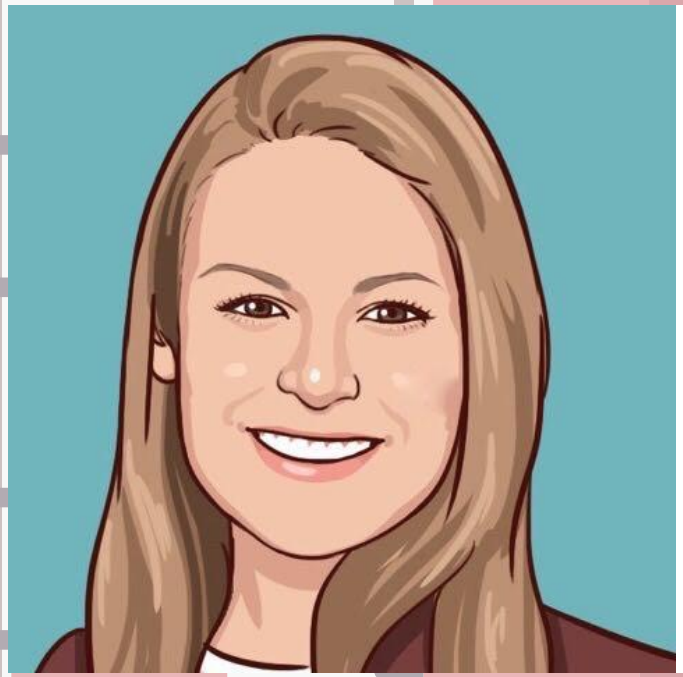
- Try to keep flags
  - As small as possible
  - As short-lived as possible
  - As top level as possible
    - “Bubble up” feature flags to the highest point in the code they can live

Summary

- Manage your feature flags, otherwise bad things can happen
- Follow best practices:
  - Visibility
    - Stalk your flags
  - Accountability
    - Monitor your flags
    - Retire your flags







Email

> caitlinarubin@gmail.com

LinkedIn

> https://www.linkedin.com/in/caitlin-rubin-a3b1a2103/