Swim between the flags

Who am I?

I’ve held roles from Software Engineer to VP of Engineering

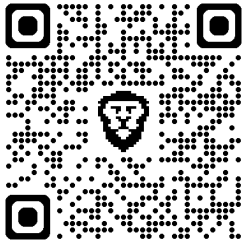
Currently a Principal Software Engineer at Westpac

I’ve been building large software systems for ~20 years

Have worked for 11 companies across 5 countries

https://jasonvella.com

https://www.linkedin.com/in/jvella/



Introducing the Flag Marshalls

Once upon a time, in a company far, far away was a software development team called the Flag Marshalls. They were given a problem to solve … and what a problem it was …


DARK LAUNCH
LIGHTYEAR
imgflip.com

Software delivery is problematic

Different audiences have a different appetite for change …

Engineering

- Deliver in small, frequent iterations
- Prefer evolution to big bang

Go To Market (GTM) squad

- Deliver in big drops
- Send out the marching band!

End users can be adverse to frequent change

- Change == risk and some people don’t like risk (banks, govt, legal …)
- User base may not be uniform in risk aversion

The solution

Dark Launch

- The concept of decoupling deployment from release

Deploy the code to production but in an “inactive” state

- Cornerstone of modern DevOps
- Inactive can mean different things in different contexts

Has been common practice for decades

- Facebook pioneered Dark Launching in 2009 (Facebook Blog)
- Mentioned in Google’s SRE book in 2016

How does this help?

Logic can be tested and deployed in iterations but disabled for the end user

- Happy engineers

When ready for launch the feature is enabled and BAM! … the end user sees the new functionality as though it were rolled out in one go

- Happy GTM team

Conceptually as simple as a boolean but in practice they can be multivariant and allow for early adopter groups, staged rollouts, etc

- Happy end users


FEATURE FLAGS
makeameme.org

Definitions

A feature toggle in software development provides an alternative to maintaining multiple feature branches in source code - Wikipedia

Feature flags are a software development concept that allow you to enable or disable a feature without modifying the source code or requiring a redeploy - Launch Darkly

Decouple deployment and release. Feature flags help you modernise your development process and deploy to production safely - Flagsmith

Moving parts

Flag provider

- A way to create and remove flags
- Some way to enable/disable flag
- A way to programmatically query the value of a flag

Flag consumer

- Your code
- Can be as simple as a conditional

```
if (flagProvider.GetFlagValue("DeliveryEnabled"))
{
    //show the delivery button
}
```

Flag providers

Home Grown

Reconsider your life choices

Cloud Native

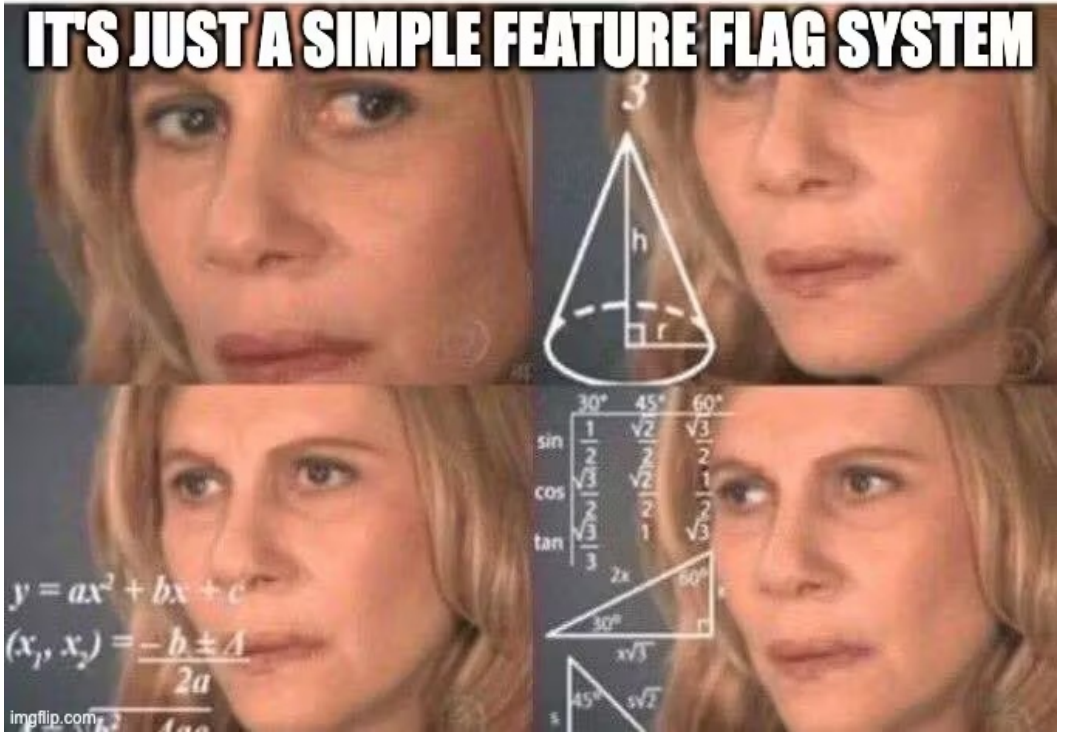
Azure App Configuration
AWS App Config Feature Flags

Open Source

Unleash
Flagsmith

SaaS

LaunchDarkly.io
Split.io




DARK LAUNCH
LIGHTYEAR
imgflip.com

Feature release day…

Marching band deployed

Customers on the edges of their seats

We launch and BAM … nothing … back to work …

All seems well, pats on the back and all that

Wait

The first calls start trickling into the call centre

Murmurs start to circulate through product channels

An uneasiness settles over the engineering floor

Oh dear

Calls are increasing in volume

Pitchforks and flaming torches are out on social media

This is not good …

Fix it!

The team is under pressure to solve the problem

Debugging and fixing forward is too slow

We need this sorted yesterday

We turn off the feature flag effectively rolling back the feature release

Remember we sent out that marching band?

The dawn of the safety flag

Having proved its worth as a panic button the humble feature flag soon becomes a staple tool in the engineering toolbelt

Most changes now come wrapped in a *safety* flag

We deploy with confidence as we can just turn things off if any issues arise

Characteristics of a safety flag

If we think about the use case we are now solving for we can extrapolate that safety flag implementations have some or all of the following characteristics:

Fine grained with a narrow scope

Nested or dependant on other flags

Multiple flags for a single feature build (often one per iteration)

Flag numbers increase as flag scope decreases


YOU GET A FLAG
AND YOU GET A FLAG
makeameme.org

Feature flags aren’t free (quite the opposite)

Potential changes to logic to ensure existing *behaviour* remains unchanged

- It’s often “stitched in” rather than a binary if/else

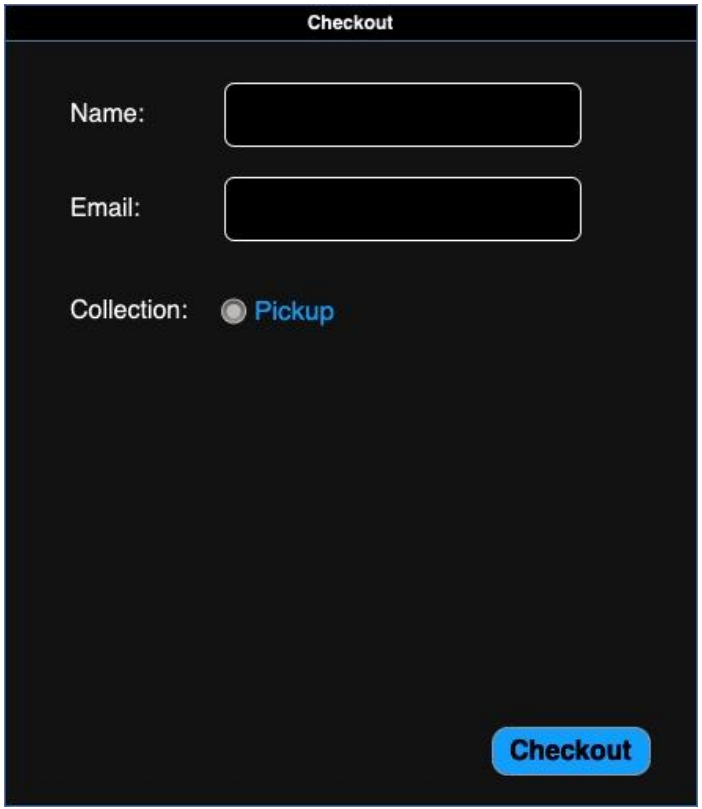
Increased test effort to ensure both sides of the flag work as expected

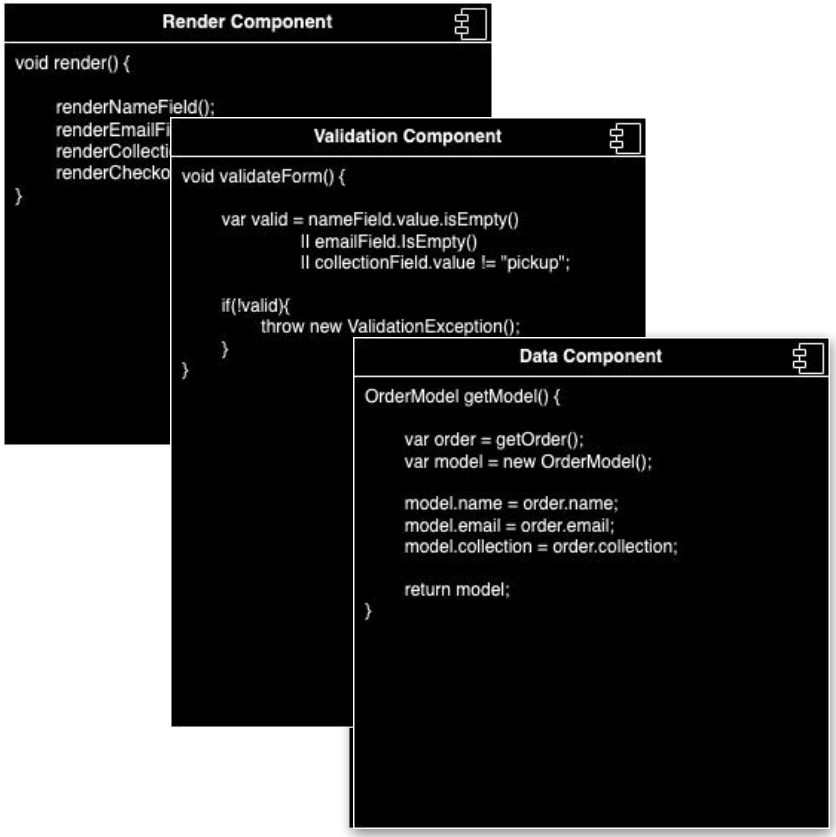
- If we aren’t testing both sides the flag is not useful
- Test combinations grow exponentially as flag numbers grow linearly

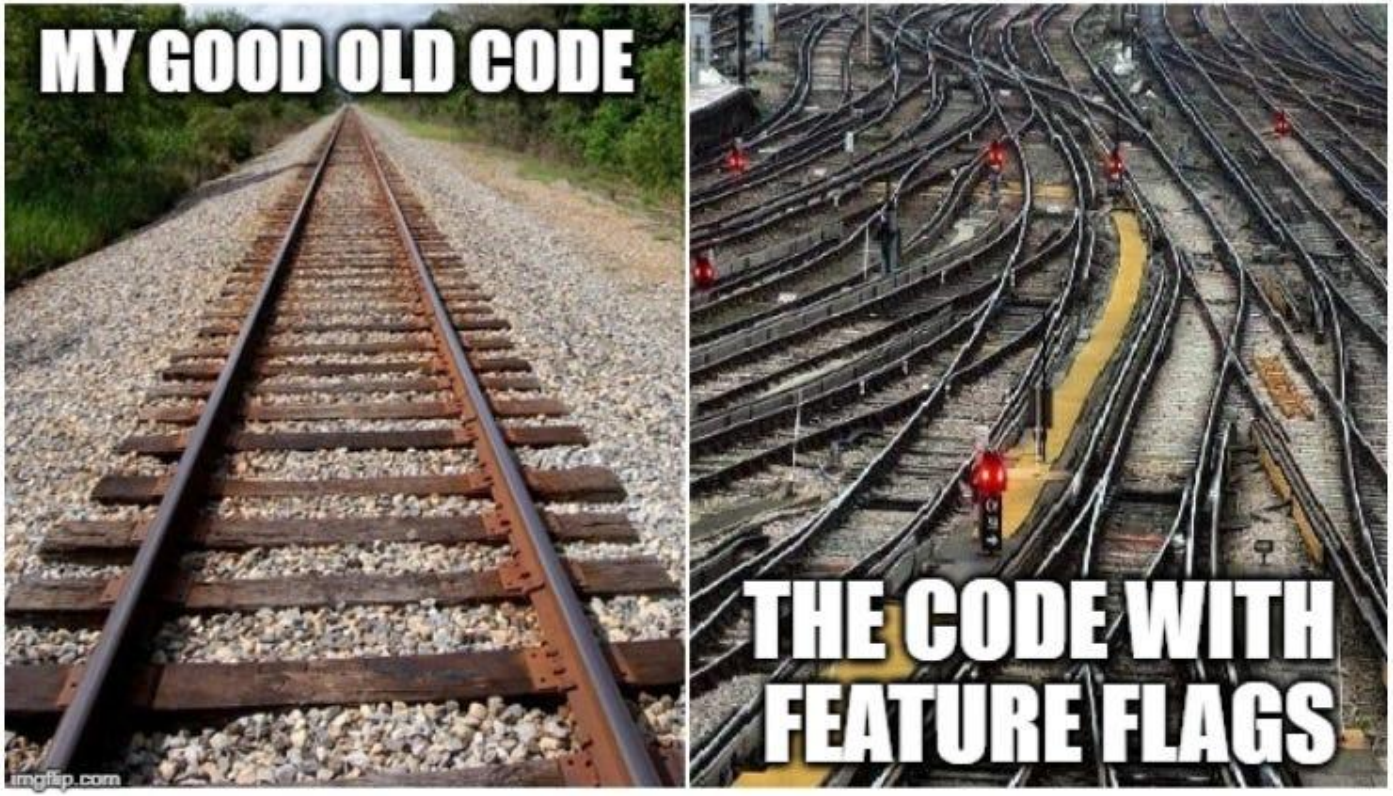
Tech debt accumulates if old logic paths are not cleaned up

- It’s difficult to debug a system with many logic branches
- It’s really difficult when it’s not obvious which branch is used in a given environment
- Designs of new logic need to consider both on and off states - do we build on both sides of the flag or does this flag now turn off two things?

Software systems are rarely *simple*






MY GOOD OLD CODE
THE CODE WITH FEATURE FLAGS
imgflip.com

Infinite loop

Combinations of logic branches overwhelms QA effort

Solution becomes brittle and difficult to debug

Design gets bogged down

Stability suffers

Delivery stalls

The purpose of a safety flag is to allow rapid rollback of changes in a state of ***crisis***

Safety Flags aren’t safe

Safety flags are a flawed approach to defect management

Defects not detected until production systems

End users affected

Require an embarrassing rollback


STAND BACK EVERYONE!
TESTING IN PRODUCTION
memegenerator.net

Solve the right problem

If you have bugs making it out to production significant enough to warrant rollback of an entire feature, you have a quality problem and *that* is the problem that needs solving

Safety flags optimise for defect rollback but we can get better outcomes if we optimise for successful release

Do we really want to be *that company* with all the issues on production?

A flag is just a flag

Feature flags solve a particular problem but are open to abuse

Common with software tools (AutoMapper anyone?)

People have found good and bad uses for almost every invention in history

If used as intended they are still a powerful and valuable tool

Fairly ubiquitous so we may as well use them properly!


DARK LAUNCH
LIGHTYEAR
imgflip.com

Swim between the flags

Feature flags are a great tool to decouple deployment from release

They are *not* a great tool for defect management

A handful of simple rules to make sure we get the value feature flags can bring while staying on top of the complexity they add

Not an exhaustive list but a good place to start!

Do you *really* need a flag?

The easiest flag to deal with is the one you never create

Is it worth making a song and dance about?

- Maybe some things are just better *evolved* into your platform

Can we just release it?

- Small enough to do in one go?
- Release in smaller features?

Can we sequence the work to achieve the same thing?

- Release the APIs first and release the UI when ready?
- Apis generally backward compatible so only flag the UI?

Align flags to product releases

Flags are about GTM activities rather than engineering deployment

End users should notice a release … they should not notice a deployment

Flags should be coarse grained and encompass a large feature *set*

- Release of a new profile page
- Release of your new AI assistant
- Something you would advertise on your blog

Encourages a short flag life cycle

Avoid nested or dependent flags

Danger Will Robinson Danger!

These are a special kind of ugly

They turn toggling a feature on/off a carefully orchestrated dance

Failure to adhere to the sequence can result in unpredictable behaviours

Testing the different combinations of flags soon becomes impossible

Automate the testing burden away

Now that it’s alive we have to water and feed it

This means we have to test two (at least) logic paths every time we make a change that affects toggled logic

You do not want to be doing this manually

Automated tests form a safety net that allows rapid change in a complex environment

Feature flags just made your environment complex

Clean up feature flags deliberately and early

You don’t want that old logic hanging around

You want to be working in a clean code base

Do you update dependencies used by toggled logic?

Do you fix vulnerabilities?

Static analysis warnings?

You’ve released the thing … it’s served its purpose … it’s just dead weight

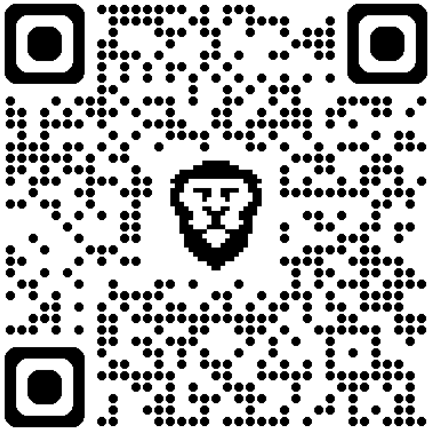
Isn’t this obvious?

The solution might be but the problem needing solving was not

These ramblings are based on my own experiences and there will be plenty that have alternative views

Further Reading

https://martinfowler.com/articles/feature-toggles.html



… you’ll be glad to know the Flag Marshalls lived happily ever after.

Questions?

https://jasonvella.com

https://www.linkedin.com/in/jvella/