Feature Flag Fails Can Cost Millions

https://media.giphy.com/media/EimNpKJpihLY4/giphy.gif

Most people keep their flagging disasters under wraps. But it's a lucky day today, because we're going to break down some scenarios of **feature flag** accidents and explore the insights that we can learn from them.

Reused Flag Names

Imagine a situation when someone names a flag like this: **brand_new_flag**. Then think about the havoc this name will cause for both **back-end** and **front-end** teams.

Each of these crews could use the same flag and step on each other's toes by touching the same switches.

To avoid that, come up with a naming convention where each toggle has a verbose name like this:

this_is_a_lenghty_flag_name_created_for_this_newsletter. Also, think about including its purpose, e.g. by including a prefix of “Release,” “Experimental,” and others.

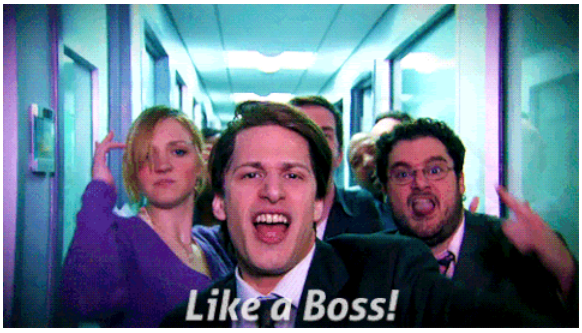
Holding on to Outdated Toggles

Another sad situation is when short-term switches contribute to increased **technical debt** which is not to be confused with **long-lived toggles**.

The longevity of a toggle refers to how long it is expected to be in use. For example, release toggles are needed in a **codebase** for a limited period of time. *In contrast,* circuit breakers and permission toggles are long-lived. They provide control for years after the release of a particular feature.

With that said, **fleeting feature flags** should be disposed of as soon as they have served their purpose.

Thus, **LinkedIn** and some other giants have a proven record of sticking with **end-of-life** toggles. Linkedin users once had an issue navigating the platform, because the site ran a release with all flags switched on.



https://media.giphy.com/media/Af7ap9r7NzJJe/giphy.gif

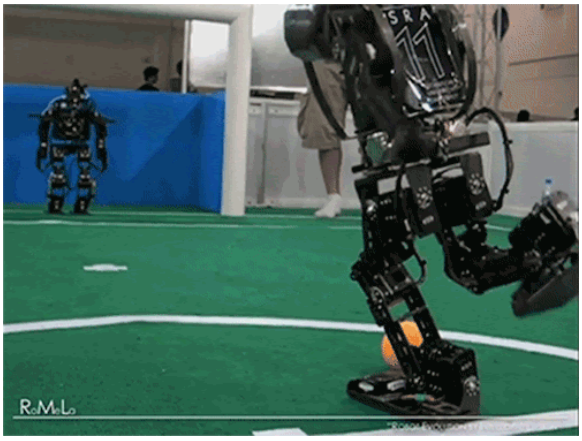
Bonus: You can avoid this sorry state of affairs by implementing a naming convention. Try prefixing your short-lived feature flags with ***temp-*.**

Logging to Prevent Forgetting

The **logging practice** is a mandate for feature flagging. Once you log all the changes to a feature flag, you can track who made modifications and at what time. So if anything goes sideways, you'll always have someone accountable for the issue. It also means that you can fix the bug without wasting time on finger-pointing.

With that said, here's a cautionary tale with a whole host of *don'ts*:

Have you heard about Knight Capital losing more than $400 mln due to a trading glitch? The glitch set off turmoil to the company *by selling all the stocks it **accidentally** bought Wednesday morning*.



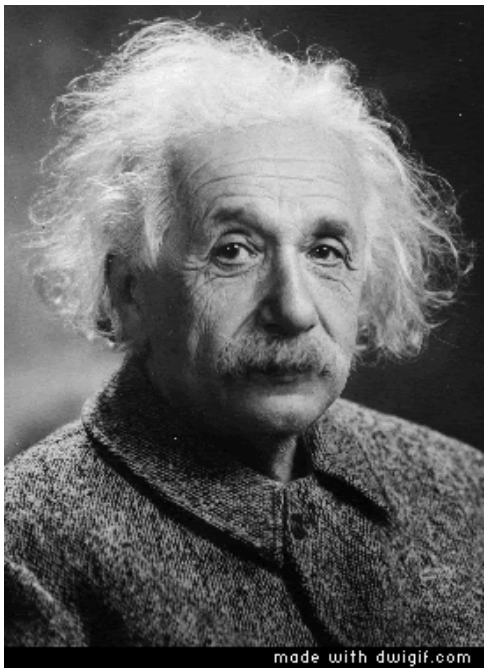
https://media.giphy.com/media/EizPK3InQbrNK/giphy.gif

In short, they left the "old" version and reused a toggle (yiykes). When the flag was flipped, the company reverted to the obsolete code still running on the eighth server and reinstalled it on the other **seven** versions.

As it turned out, this caused a domino effect, since all eight servers then had the defective code triggered by the reused RLP toggle and executing in full throttle.

The Moral of the Story

The Peter Parker principle says that with great power there must also come great responsibility. Although he knew nothing about feature flags, the DevOps teams can still relate to that.



https://media.giphy.com/media/3ohc1a4MRy0vQr7ww8/giphy.gif

Switches introduce complexity. We can keep tabs on this complexity by using time-tested feature flagging practices and appropriate tools to manage them.

Join us today to thank our newsletter sponsor, LaunchDarkly. LaunchDarkly is the #1 platform for managing Feature Flags. LaunchDarkly allows users to test new products safely, release new updates confidently, control the internal process dynamically, and measure the impact of their initiatives. Feature Flagging is an industry best-practice that allows for the faster deployment of code. This means more new features from your platform delivered exceptionally fast.

***

Got a tech story to share with our readers? Everything you've ever wanted to know about how to get published on Hacker Noon - **get it here**