Peter Conn

1) What kind of experiences do you have with different usages of feature toggles?

My main **use** of feature toggles is **for development**.
I'd start work on a **new feature that is hidden behind a toggle** (usually a command line flag). This means code can be committed to the main repository before the result is polished enough for users. An alternative to this would be developing the new feature on a different branch - although at the rate of Chrome's development we'd be dealing with so many merge conflicts when we wanted to finally commit the code.

This also means that we can decouple the technical process of launching with the product decision of "shall we launch this feature" - we can write the code whenever we want and then the feature can be turned on for users once it's gone through all of our feature reviews (eg, it's got the sign off from our product, UX, privacy, security, accessibility teams).

2) What problems have you encountered while using feature toggles?
Spread Toggle
During the course of working on a feature, you'll likely have multiple iterations and multiple sub-features. In order to be flexible and manage work across a team, it's nice to keep these sub-features independent when you can. This can lead to a problem where you've got 3 different sub-features all in development at the same time, and if you have each of them behind it's own feature toggle, you've now got 2^3 = 8 different possible configurations that the user can be in. This interaction between multiple new/in development features is a ripe area for bugs, and if you get a bug report, sometimes it can be hard to even figure out which combination is causing the bug.

3) Do you think it would be better if there were best-practice standards for feature toggles?

I could see there being some value in published best practices, although they definitely should be split up based on the different use cases for them (as Peter already mentioned).

Finner Thoranisson

1) What kind of experiences do you have with different usages of feature toggles?
I've added toggles for pretty much every category you've indicated - which I guess could be seen as a bias in my answer.

I'm probably in the same boat. I can't even remember what I used the toggles for anymore. :)

In a nutshell, my view is that there's a right tool for the job. This has two perspectives. First, intended usage:

- If a feature should be limited to internal or power-user behaviour, use a command line flag. (I.e. in the *_switches.cc files.)
- If a feature should be limited to technically savvy users, use either a base::Feature or a command line flag and list it on about://flags.
- If a feature should be used for experimentation or server-side configuration, e.g. a kill-switch, use a base::Feature.

Second, configurability. **Command line flags** *can* support values, but they **quickly become impossible to manage**.
Nested Toggle
Having a drop-down box with preconfigured options on about://flags is a much easier option there.
Or have multiple individual flags that depend on each other: I suspect you mean that with nested flags.

I think there would be value in having a best-practice standard for these toggles. I mean, just the paragraphs Peter wrote (between my comments) would help a lot each time people think of adding a toggle.

Then there is **applicability**. ***How do we use a flag?*** This is similar to what you've written in the paper, and in my view **depends on the rest of the code**.
Any branch in functionality creates a permutation in intended behavior, which has consequences on the complexity (and coverage) of tests, maintenance and often stability.

2) What problems have you encountered while using feature toggles?
Extending on that thought: **we avoid feature toggles from being long-lived**, and actually track that in our repository. **Long-lived toggles have a tendency to *rot*, creating unintentional regressions**. Similarly, in large projects, it might create the perception for

users that particular functionality is intended to stick around forever despite not being the default, which rarely is the case.

In scenarios where one is changing a pre-existing feature, it is very useful to have a toggle to switch between implementations, which avoids disrupting the user base with an implementation that is still work-in-progress. It is also very useful to tie that into an experiment where you compare the performance of the close-to-ready new implementation with the old implementation. But it can be a double-edged sword if not used properly.

For example, if the experiment takes a while to run (but shows good numbers) and QA, for whatever reasons, doesn't have time to give their green light until close to the deadline you are trying to make, it is very tempting to just expand the study to cover everyone (flip the flag from default-off-but-governed-by-study to default-on-but-with-kill-switch), thereby launching the feature. But since the feature has been code-complete for a while, perhaps the feature has made its way most (or even all) of the way to the Stable channel (albeit default-off). Enabling it would circumvent an important safety valve, which is the release channels -- unstable channels especially, which is what developers (both internal and external to Chromium) use to avoid being surprised by upcoming changes.

Enabling the feature via a study should at least come with some minimum-version guarantee, and that version should not be the version that contains the months-old last meaningful changelist that affected the feature but the version that comes out today/tomorrow (which will hit the unstable channels way first).

Peter Beverloo

Hi Harika,

Thank you for sharing! That's a fascinating outcome. Let me include Finnur and Peter C who may have opinions, as both of them have used plenty of feature toggles as well.

To your questions:

1) What kind of experiences do you have with different usages of feature toggles?
I've added toggles for pretty much every category you've indicated - which I guess could be seen as a bias in my answer.

In a nutshell, my view is that there's a right tool for the job. This has two perspectives. First, intended usage:

- If a feature should be limited to internal or power-user behaviour, use a command line flag. (I.e. in the *_switches.cc files.)
- If a feature should be limited to technically savvy users, use either a base::Feature or a command line flag and list it on about://flags.
- If a feature should be used for experimentation or server-side configuration, e.g. a kill-switch, use a base::Feature.

Second, configurability. Command line flags *can* support values, but they quickly become impossible to manage. Having a drop-down box with preconfigured options on about://flags is a much easier option there. Or have multiple individual flags that depend on each other: I suspect you mean that with nested flags.

Then there is applicability. *How do we use a flag?* This is similar to what you've written in the paper, and in my view depends on the rest of the code. Any branch in functionality creates a permutation in intended behaviour, which has consequences on the complexity (and coverage) of tests, maintenance and often stability.

2) What problems have you encountered while using feature toggles?
Extending on that thought: we avoid feature toggles from being long-lived, and actually track that in our repository. Long-lived toggles have a tendency to *rot*, creating unintentional regressions. Similarly, in large projects, it might create the perception for users that particular functionality is intended to stick around forever despite not being the default, which rarely is the case.

3) Do you think it would be better if there were best-practice standards for feature toggles?
Perhaps more of a "well-lit path". When a developer has a goal in mind, a particular sort of feature toggle could be recommended. However, whereas some projects are developed using a very homogenous architecture, Chromium is not, and there could be good reasons for a developer to derive from such best practices.

Thanks,
Peter