

Capture the Feature Flag: Detecting Feature Flags in Open-Source

Jens Meinicke, ♦ Juan Hoyos, ♦ Bogdan Vasilescu, ♦ Christian Kästner ♦
♦Carnegie Mellon University, USA ♦ Universidad Nacional de Colombia, Colombia

ABSTRACT
Feature flags (a.k.a feature toggles) are a mechanism to keep new features hidden behind a boolean option during development. Flags are used for many purposes, such as A/B testing and turning off a feature more easily in case of failures. While software engineering research on feature flags is burgeoning, examples of software projects using flags rarely come from outside commercial and private projects, stifling academic progress. To address this gap, in this paper we present a novel semi-automated mining software repositories approach to detect feature flags in open-source projects, based on analyzing the projects' commit messages and other project characteristics. With our approach we search over all open-source GitHub projects, finding multiple thousand plausible and active candidate feature flagging projects. We manually validate projects and assemble a dataset of 100 confirmed feature flagging projects. To demonstrate the benefits of our detection technique, we report on an initial analysis of feature flags in the validated sample of 100 projects, investigating practices that correlate with shorter flag lifespans (typically desirable to reduce technical debt), such as using the issue tracker and having a flag owner.

1 INTRODUCTION
Feature flags (aka. feature toggles) are becoming an increasingly important, but also controversial software-engineering practice with the advent of continuous deployment and delivery. Technically, feature flags are a design pattern to conditionally enable a code path (e.g., an if-statement controlled by a boolean flag), where the decision is typically controlled by an external configuration mechanism. Feature flags enable development of new features in the same branch, which can speed up releases while avoiding large merges [4, 8]. Flags are also used for experimentation in production [1, 14] and canary releases [12, 13]. Feature flags are widely discussed in blog posts and at practitioner conferences, and there are multiple competing startups offering tool support. Feature flags can be controversial and seen as a cause of technical debt, because they are easy to introduce, create additional complexity in a system, and are often hard to remove [6, 7, 9].

Unfortunately, there is little research on feature flagging practices, except for a handful of studies of Chromium [9, 10] and a few commercial projects [6, 7]. Unlike many other software engineering practices, where open-source software enabled a wealth of large-scale empirical research, studies of feature flags in open-source are rare. In part, this might be explained by feature flagging being a rather commercial practice, often used for A/B testing commercial web and mobile products. Another explanation is that feature flag use is non trivial to identify systematically, leaving uncertain how widespread and how diverse the practice is in open-source. Indeed, feature flag implementations can range from custom solutions to commercial software libraries, e.g., from Split.io or LaunchDarkly, to simple Boolean options in code or configuration files; Listing 1 shows an example of the latter—feature flags in Automattic/wp-calypso are defined as a map in a json file.

To enable and encourage empirical research on feature flagging practices, in this paper we propose a novel mining software repositories technique to identifying open-source projects using feature flags, based on textual analysis of commit messages for patterns such as "feature flag" combined with a series of filters to remove likely false positives (§3). Applying our technique to all public repositories on GitHub uncovers 3 237 candidate feature flagging projects. To assemble a starting dataset for future research, we manually confirm 100 projects that actually use feature flagging, which together account for 7 593 different flags in total. We demonstrate the value of this dataset with a small preliminary study on feature flagging practices and changes to the feature flags over time (introductions and removals) (§4). Among others, our study reveals that high ownership (i.e., the author who introduces a flag also removes it later) is statistically significantly associated with shorter flag lifespans, suggesting that ownership can be an effective practice to keep technical debt in check. In short, the goal of this work is to identify open-source repositories that use feature flagging and to provide a dataset, to help researchers study feature flagging practices on a variety of open-source projects.

In summary, the contributions of this paper are:

* We propose a novel mining software repositories technique to
identify repositories that likely use feature flags, resulting in
3,237 plausible candidates on GitHub.
• We provide a dataset of 100 projects and their 7,593 feature flags,
including information on each flag’s lifetime.
• We report on an initial analysis of this sample, showing that
ownership of feature flags correlates with shorter flag lifespans.

Our regular expressions for identifying feature flags, together
with the two lists of plausible candidate and manually verified
projects, are all available publicly at

2 RELATED WORK ON FEATURE FLAGS
Feature flags are a topic frequently discussed by practitioners in
blog posts and at practitioner conferences (e.g., [2, 3], Rahman
et al. [9, 10] and Mahdavi-Hezaveh et al. [6] provide an extensive
overview of grey literature on feature flags). According to an
interview study with practitioners [7] there are three common use
cases for feature flags: (a) parallel trunk-based development, where
multiple features guarded by feature flags are developed simultane-
ously in the same branch, (b) canary releases, where features are
released incrementally to different users, and (c) experimentation in
production (A/B testing), where features are selectively activated to
measure their impact on business objectives. In addition, they found
that the boundary between feature flags (intended to be temporary)
and configuration options (intended to be permanent) [11] is often
fuzzy, and often the same mechanisms are used for both.

Academic research on feature flags is rather sparse. Meinicke
practitioners about their feature flagging practices, finding among
others than feature flag removal is a key pain point, that testing
is rarely conducted systematically across feature flags and their
interactions, and that concerns about technical debt abound. To
the best of our knowledge, the only study on feature flags using
mining software repository techniques is the analysis of feature
flags in Chromium, the open-source implementation behind Google
Mouse, by Rahman et al. [9, 10]; they found that feature flags are
often long lived, confirming concerns expressed by practitioners.

3 CAPTURING FEATURE FLAGS
Finding open-source projects that use feature flags was a surpris-
ingly challenging task. Many of our initial attempts (described
briefly below) found barely any projects or overwhelmed us with
false positives, such that we doubted whether feature flags would
be used in open-source at all.2 We incrementally developed and
refined a method to identify and filter open-source feature-flagging
projects at scale on GitHub.

Our approach is semi-automated, using various heuristics to
identify promising candidate projects which we then manually vali-
date. Our goal here is not to identify all feature-flagging projects on
GitHub, but to build a substantial dataset of active feature-flagging
projects that can be used (and expanded) in future research.

Heuristics and Initial Sampling. We initially explored many
different strategies to identify open-source projects using feature
flags, including (a) finding projects importing feature flagging li-
braries (finds only few open-source projects, often without serious

2At some point we even had a bet among team members whether we would ever find
more than 20 open-source projects with serious feature flag use.
However, we did not manually inspect all 3237 projects. Instead, to guide our inspection process to focus on more likely candidates first, we developed the following process. First, we inspected the 50 projects with the largest numbers of commits matching our search terms. This revealed a significant number (about 50%) of false positives among those projects, i.e., projects that use flags for other purposes than feature flagging, such as compiler flags, preprocessor flags, or configuration options. We then analyzed what distinguished feature flagging projects from false positives in this initial set, developing two further prioritization heuristics:

- We compared the fraction of commit messages that match individual search terms used in H₁ and H₂ above, in the first 50 manually classified projects. A closer inspection of the matched terms reveals that non-feature-flagging projects tend to have a high percentage of messages matching flag removal, i.e., “remove flag” and “delete flag” (H₂) compared to feature-flagging projects, which have higher percentages for “feature flag” and “feature toggle” (H₁). Feature-flagging projects also have commit messages matching removal (H₂), however in a lower proportion; it is possible that these projects contain more feature-flagging-related tasks, such as adding new flags or changing the flag values. Figure 1 shows how the analyzed projects cluster with respect to their number of feature-flagging commits and the percentage of commits matching H₁. Note how the manually confirmed non-flagging projects (labeled ’Denied’) are on the bottom, while the manually confirmed feature flagging projects are on the top, with high percentages of commits matching H₁. Based on this insight we used the ratio of flag removal mentions to all flag mentions as a prioritization heuristic to aid our manual validation effort (H₃).

- We inspected the changes made in feature-flagging commits (H₁ and H₂) with the goal of identifying the files that define the feature flags. Such files commonly store the flag in a single field or as an entry in a map-like data structure. Thus, changes to feature flags in these files, such as adding or removing a flag, tend to only involve a few lines of code. We ranked the files touched by feature-flagging commits by median number of line changes and number of commits touching them. Analyzing the manually classified projects we found that repositories with changes to file names containing ‘feature’ and either ‘flag’ or ‘toggle’ are very likely to be actual feature-flagging projects. Based on this insight we used the names of the files defining the flags as a prioritization heuristic to aid our manual validation effort (H₄).

These prioritization heuristics (H₃ and H₄) were effective in guiding the rest of our inspection process, having few false positives among the highly ranked projects. Using H₃ and H₄ we continued to validate candidate projects until reaching 100 confirmed feature-flagging projects total. Note that H₄ uncovered 185 additional likely feature-flagging projects (labeled ’Likely’ in Figure 1), but we have yet to manually validate these. If more feature flagging projects are needed, we are confident that, with further manual analysis, one can find many more among the remaining candidate projects.

The Dataset. Both our candidate set of 3237 projects as well as our validated set of 100 projects, the latter also including the exact names of the feature flags used, are part of our dataset.

In Figure 2 (b–e), we characterize the validated set of 100 feature flagging projects in our dataset. The projects are mostly large, active, and popular, and written in a large number of different programming languages, showing that feature flags are indeed used in practice in open-source. Although we cannot be certain that our dataset is representative of all feature flag use in open-source, our data suggests that feature flagging is most common in web applications—this is expected, as websites (in contrast to say desktop applications) allow to roll-out features to subsets of users and for A/B testing on the server side.

4 PRELIMINARY FEATURE-FLAG STUDY

Our dataset of 100 open-source projects confirmed to use feature flags, which contains many large and active projects, will be a valuable resource to study feature-flag practices in public repositories. To demonstrate the potential, we report on a preliminary study of these 100 projects, that analyzes some aspects of feature-flag use.

Flag Data Collection. Informed by the previous manual inspection of each project, we collect data about individual feature flags, including how flags are used, introduced, and removed. Most of the projects define their flags in a single file, usually either as field, or in a data structure (e.g., a map). We developed tooling, such that we only need to define the file path(s) to files storing flags, and a regular expression that we use to identify the flags’ names and locations. We manually created these flag specifications for all 100 projects, which we release as part of our dataset. We show one such definition in Listing 2.

We then mined the git history of each project to record when flags are added and removed and by whom. In total, we collected data for 7593 feature flags in the 100 projects of our dataset (distribution in Figure 2a). The median number of flags per project is 34 and Chromium has the maximum number of flags with 1790.

Preliminary Analysis and Results. We analyzed the lifespan of feature flags to observe how frequently flags are cleaned up in open-source projects. Figure 3 shows the Kaplan-Meier estimate of
the survival probability for flags in our sample, accounting for right censoring (flags that have been introduced recently, that may or not be removed in the future). The estimator suggests that most flags are expected to be eventually removed, that half of the flags are removed within 15 months, but also that 25% of flags remain for a very long time or forever (more than 8 years).

Overall, we found that cleanup practices differ across projects; most projects clean up most flags, but few projects keep flags alive for a long time. Note though that the distinction of temporary feature flags and permanent configuration options is not always clear from the outside and a flag initially introduced as temporary might evolve into a permanent option in practice, without the ability of external observers to distinguish this in the implementation [3, 7, 9].

We further analyzed how ownership and tracking flags in issues associates with the lifespan of feature flags, for those flags that get removed (4 032 out of 7 593). First, flag ownership is a suggested practice to encourage the cleanup of flags by expecting the creator (i.e., the owner) of the flag to remove the flag after the feature is completed [6]. We identify all flags as having ownership if they were removed by the same Git user who introduced them in an earlier commit. Second, another suggested practice is to create a “cleanup” issue per feature flag, stating that the flag should be removed when the feature is finished, and assign the issue with a future date to a team or developer as a reminder [6]. We identified flags linked to issues by looking for patterns “(#nr)” in the commit messages that introduce or remove the flag.

We then estimated a mixed-effects linear regression, modeling flag age (days) as a function of the presence of an associated cleanup issue (boolean) and flag ownership (boolean), while controlling for project size (number of commits) and with a random effect for project, to account for the hierarchical nature of our data (flags nested within projects). The model has a total explanatory power (conditional R2) of 63%; the fixed effects alone explain 36% of the variance (marginal R2). Within this model, the effect of flag ownership is significant (beta = -189.13, SE = 10.14, 95% CI [-209.02, -169.25], t(3964) = -18.65, p < .001) and can be considered as medium (std. beta = -0.55, std. SE = 0.03). The effect of cleanup issues is negligible. Results suggest that flags removed by their owner have, on average, a 189 days shorter lifespan than flags removed by others, which suggests that ownership could be an effective practice to keep the project clean of stale flags and thus reduce technical debt. The weak correlation between cleanup issues and the lifetime of flags warrants more in depth future analyses.

5 CONCLUSION AND OPEN QUESTIONS

Research on feature flagging is sparse and focuses on interviews with practitioners. Beyond the Chromium study by Rahman et al. [9, 10], we are not aware of any mining software repository studies of feature flags, possibly because it is difficult to identify projects using the practice, as there are no obvious identifiers. In this work, we describe our semi-automated process to assemble a dataset of 100 open-source feature flagging projects, and we additionally collect data about the individual flags used in these projects, and their lifecycle. Our preliminary analysis demonstrates that this dataset provides promising research opportunities; among others we found that flag ownership is a promising practice which correlates with faster flag cleanup and may help control technical debt pending confirmation through experimental methods. There are many more practices recommended by practitioners (see grey literature overviews [6, 9]), and we hope that our dataset can be used to collect evidence about which practices are effective. Also follow up qualitative studies of the implementations or interviews with open-source practitioners using feature flags can be facilitated with our dataset. We hope that our work contributes to fighting technical debt in projects using feature flags and to evidence-based guidelines about effective practices.

Acknowledgements. Meinicke and Kästner have been supported in part by the NSF (awards 1552944, 1717022, 1813598), AFRL, and DARPA (FA8750-16-2-0042). Vasiilescu has been supported in part by the NSF (awards 1717415, 1901311).
REFERENCES


