Introduction to Feature toggles - How are they used

Pontus Persman

April 2022

1 Introduction

Feature toggles add conditional logic to the execution of an application. The toggles which are also commonly referred to as flags are simply if statements with environmental variables which flags what part of the codebase to execute. Feature toggles can be thought of as an answer to the difficulty of merging long lived feature branches as they allow for trunk based development, i.e having the collaborative effort be confined within one branch.

Several techniques which are associated with continuous development/deployment such as dark release, canary release, A/B testing etc. are often used in conjunction with feature toggles. A testament to its relevance can be seen as the biggest tech companies like Spotify, Facebook, Google, Microsoft etc. use them.

However, there also exists a lack of scientific evidence regarding both the benefits but also the drawbacks of feature toggles as the scientific knowledge is largely based on outspoken industrial leaders [1, 2]. Due to this lack of scientific evidence this essay aims help guide the decision of whether to use feature toggles. This essay first presents agile applications of feature toggles, advise from practitioners as well as what has been previously found in the literature, and finally a conclusion .

2 Background

Feature toggles facilities continuous deployment since features which are not yet completed, is in need of further testing or are experimental can safely be integrated in the main branch since the feature can be hidden from the users [3]. Thus, it allows for developers to continuously monitor user behavior, traffic as well as rollback to the old application in case the feature did not work as intended.

Figure 1 shows a basic version of how feature flags might be used in a program. While the implementation might seem quite simple as will be seen later managing them can still be tricky.

```
1: procedure USING FEATURE TOGGLE
2:     if featureFlag['NewFeature'] then
3:         Render NewFeature
4:     else
5:     .   Render Old Version
```

Figure 1: An example of how a feature toggle might be used.

2.1 Continuous experimentation

Continuous experimentation can have several benefits for a software company such as, better support for development decisions, reduced development time and deeper customer insight [4]. Some of the most common methods are, A/B testing, dark release and canary release. There are different implementation techniques being used for these methods, however, feature toggles is the most common. Feature toggles are also mostly used in web applications.

2.1.1 A/B testing

A/B testing is a rather simple experimental setup where different version are compared. Usually the included components are, a control version, i.e the existing system and a treatment version which would include the feature to test [5]. For most statistical power a division of the users to the different version of 50% should be used, but any given split in percentage can be used. It is also common to be expand to several layers. That is to say, to A/B/n.

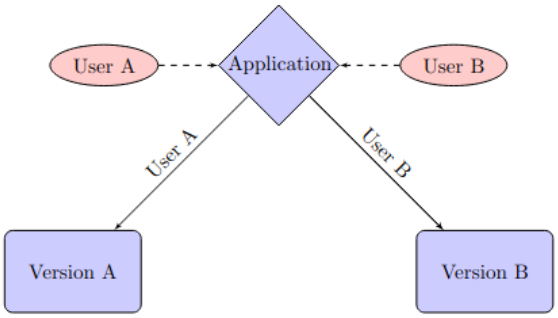

Figure 2: An example of A/B testing where certain version only are visible to certain users.

Microsoft's Bing is an example of an avid user of continuous experimentation as they have over 200 concurrent experiment running any given day [6]. The test can for example include usability and performance for which improvement in any leads to increased revenues. Even a minor improvement means a big revenue increase for engine as large as Bing, with a 1% increase in revenue translating to $10M annual increase.

Another big tech company which uses A/B testing is Netflix [7]. They mention that having a structured way of testing is an essential part of innovation as well as how A/B testing leads to data-driven judgement which thus reduces the risk the HiPPO (Highest Paid Person's Opinion) problem.

2.1.2 Canary release

Canary release is where only a small subset of users see a new version or feature, while having the majority of users still uses the old version [4]. In case the new implementation encounters problems this effectively limits the scope of it. While on a technical level similar to A/B testing it has been found that their usage differs in the experimental type. Canary releases are used for regression-driven experiments, i.e for testing technical problems such as, bugs, scalability or performance, whilst A/B testing are rather used for business-driven experimentation such as testing user satisfaction with changes.

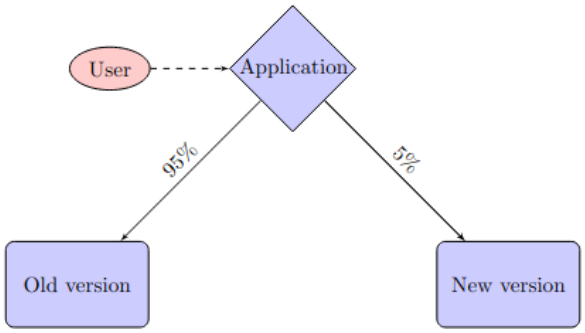

Figure 3: An example of how a canary release might look. The percentages refers to how many users are exposed to a certain version.

2.1.3 Gradual rollouts

Are mostly used in combination with A/B testing or Canary release [4]. As its name implies the new version or feature is gradually deployed to a larger subset of users until either the old version is replaced or a certain threshold is met.

2.1.4 Dark launch

Dark launchs are to test functionality when facing production like traffic without impacting any users, which is achieved by being hidden for users [8]. All regular traffic to the application is duplicated to the "dark" version, which allows for insight into behavior of new functionality without the drawbacks of impacting users of the application. Not only is dark launch as a concept pioneered by Facebook but it is also mostly used by them as well [4].

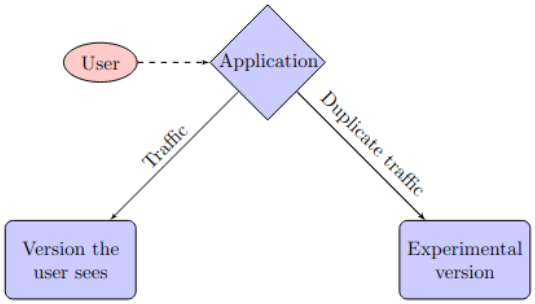

Figure 4: An example of how traffic is flowing in a dark release.

2.2 Trunk based development

Trunk based development is where developers merge small changes and frequently to the "trunk" or main branch [9]. This development method can be seen as a necessity for CI/CD workflows. The other development method mostly used is Gitflow, where feature branches are more long-lived and only gets merged after the feature is complete. Trunk based development as opposed to Gitflow has a lower risk introducing conflicting merges.

2.3 Circuit breaker / Kill switch

When applications grows in size and complexity it becomes increasingly difficult to fix potential failures within the system [10]. Circuit breaker or kill switch is a way for developers to have a centralised system of which they can control components of the application is being run as a response to failures in an application.

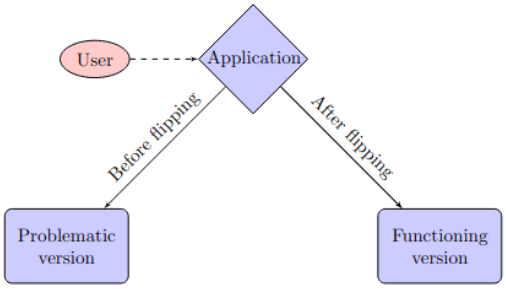

Figure 5: An example of using a kill switch can easily switch between different version of an application.

3 The potential and drawbacks of Feature toggles

The potential of feature toggles can be linked to the many different application where it is the primary implementation technique and the potential within those. However, their effectiveness will also be based on its use case and as such should be used with care.

3.1 According to Practitioners

As the scientific literature regarding feature toggles are somewhat limited it may be the fact that the practitioners are the most knowledgeable regarding their use case. As such their advise should be seen as useful and relevant.

3.1.1 Dead code

The most cited disadvantage with feature toggles is in regard to dead code [11]. Dead code can either be from feature toggles originally designed to be temporary but turned into business toggles instead or code related to a scraped feature but left with its toggle disabled [11]. This should by no means be overlooked as this is what lead to Knight Capital Group losing about $460 million, as they mistakenly turned on such a toggle which lead to different code being run inconsistently with both old and new code being executed for different users.

3.1.2 Compared to Long-lived branches

Feature toggles reduces complications with merges of long-lived feature branches and are thus well suited for companies who releases often [11]. Rather than having to deal with strenuous merge conflicts which can last months feature flags allows for trunk based development and it is likely this advantage which causes big tech companies who follow a CD pipeline to use them. That is to say, feature toggles are a well suited choice for implementing new features while adhering to the continuous integration infrastructure.

Another big advantage is the use of feature toggles in conjunction with gradual roll-out, especially for big tech companies. Since scalability issues can not be simulated with in-house deployment and reverting to an older version results in too much downtime, feature toggles ends up being an effective solution.

3.1.3 Design patterns

Feature toggles do also need to be taken into account when designing components [11]. Components with feature toggles should be designed in a way which avoid intersection since adding such a dependency would add unwanted complexity to the application. Enforcing such a design pattern in the source code may not unwanted however. Since doing this results in the functioning of components to be independent of each other, thus making testing and therefore long term maintenance of the code easier.

3.1.4 Trunk based development

As the feature toggles facilitates for trunk based development and the fact that normally extensive testing is carried out of the main branch having hundreds or thousand of branches in code leads to an explosion of paths seemingly needed to be tested. One solution to this problem could be to limit which combination of flags to test to the ones actually being expected to be used when deployed.

3.2 What is said in the literature

Trunk based development with feature toggles should in theory make merging easier. There exists evidence that while the adaptation of feature toggles may not reduce the total amount of merges it does reduce the effort associated with merges significantly [12].

Using dark launch in combination with feature flags have been reported by Microsoft to avoid having to deal with integration issues as well as maintaining long-lived feature branches [13]. However, the approach is not without its issues. Feature flags also come with extra configuration and having multiple flags can confuse developers into entering invalid configuration states. It has been reported that there is a lack of rigours testing and analysis of configuration changes. Removing feature flags is something that often leads to technical debt, i.e they cause a need for refactoring later on.

Further evidence that technical debt and the added complexity feature toggles brings to a system are a major drawback has been reported from multiple sources [11, 14]. It has been pointed out that long-lived feature toggles can lead to major failures in systems [10]. Not all feature toggles are meant to removed however, for instance, kill switches may be designed to not be removed or stay for a long time. Long-term business toggles which can be used to show different features to different users is another example of toggles which can be quite long-lived.

Based on an analysis of an official spreadsheet by Chrome developers it was found that the removal of feature toggles can be more difficult in practice than what one might expect. Since in theory release toggles should be fairly easy to remove since they can simply be removed whenever a feature is made permanent. Even though toggles were marked as "to remove" in the spreadsheet 44 out of 84 were still in the source code as technical debt [11]. Furthermore, 2 of 11 toggles which were marked as "removed" still existed in the source code.

Further empirical evidence of feature toggles lingering in the source code longer than intended has been reported elsewhere [10]. One article suggests that feature toggles are technical debt as soon as they are added and that ideally they should be short-lived as otherwise they can become costly to maintain.

4 Conclusion

Feature toggles can create a lot of value and can help facilitate the process of creating new feature and innovating. However, they should also not be carelessly implemented and following strict procedures for removing and refactoring may be a necessity. The potential of feature toggles may depend on the company in question and its use cases as they can differ quite a bit.

References

[1] Ron Kohavi et al. "Online Controlled Experiments at Large Scale". In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '13. Chicago, Illinois, USA: Association for Computing Machinery, 2013, pp. 1168–1176. ISBN: 9781450321747. DOI: 10.1145/2487575.2488217. URL: https://doi.org/10.1145/2487575.2488217.

[2] Diane Tang et al. "Overlapping Experiment Infrastructure: More, Better, Faster Experimentation". In: *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '10. Washington, DC, USA: Association for Computing Machinery, 2010, pp. 17–26. ISBN: 9781450300551. DOI: 10.1145/1835804.1835810. URL: https://doi.org/10.1145/1835804.1835810.

[3] Ian Buchanan. "Feature flags". In: (). URL: https://www.atlassian.com/continuous-delivery/principles/feature-flags.

[4] Sezin Gizem Yaman et al. "Introducing continuous experimentation in large software-intensive product and service organisations". In: *Journal of Systems and Software* 133 (2017), pp. 195–211. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2017.07.009. URL: https://www.sciencedirect.com/science/article/pii/S0164121217301474.

[5] Ron Kohavi and Roger Longbotham. "Online Controlled Experiments and A/B Testing". In: Jan. 2017, pp. 922–929. DOI: 10.1007/978-1-4899-7687-1_891.

[6] Ron Kohavi et al. "Online Controlled Experiments at Large Scale". In: Aug. 2013. DOI: 10.1145/2487575.2488217.

[7] Nirmal Govind. "A/B Testing and Beyond: Improving the Netflix Streaming Experience with Experimentation and Data". In: (2017). URL: https://netflixtechblog.com/a-b-testing-and-beyond-improving-the-netflix-streaming-experience-with-experimentation-and-data-5b0ae9295bdf.

[8] Chunqiang Tang et al. “Holistic Configuration Management at Facebook”. In: *Proceedings of the 25th Symposium on Operating Systems Principles*. SOSP ’15. Monterey, California: Association for Computing Machinery, 2015, pp. 328–343. ISBN: 9781450338349. DOI: 10.1145/2815400.2815401. URL: https://doi.org/10.1145/2815400.2815401.

[9] Kev Zettler. “Trunk-based development”. In: (). URL: https://www.atlassian.com/continuous-delivery/continuous-integration/trunk-based-development.

[10] Juan Hoyos et al. “On the Removal of Feature Toggles”. In: *Empir Software Eng 26, 15* (2021). DOI: 10.1007/s10664-020-09902-y.

[11] Md Tajmilur Rahman et al. “Feature Toggles: Practitioner Practices and a Case Study”. In: MSR ’16. Austin, Texas: Association for Computing Machinery, 2016, pp. 201–211. ISBN: 9781450341868. DOI: 10.1145/2901739.2901745. URL: https://doi.org/10.1145/2901739.2901745.

[12] Eduardo Smil Prutchi, Heleno de S. Campos Junior, and Leonardo Gresta Paulino Murta. “How the adoption of feature toggles correlates with branch merges and defects in open [U+2010] source projects?” In: *Software: Practice and Experience* 52 (2022), pp. 506–536.

[13] Chris Parnin et al. “The Top 10 Adages in Continuous Deployment”. In: *IEEE Software* 34.3 (2017), pp. 86–95. DOI: 10.1109/MS.2017.86.

[14] Philippe Kruchten, Robert L. Nord, and Ipek Ozkaya. “Technical Debt: From Metaphor to Theory and Practice”. In: *IEEE Software* 29.6 (2012), pp. 18–21. DOI: 10.1109/MS.2012.167.