On the Interaction of Feature Toggles

Xhevahire Tërnava, Luc Lesoil, Georges Aaron Randrianaina, Djamel Eddine Khelladi, Mathieu Acher

On the Interaction of Feature Toggles

Xhevahire Tërnava, Luc Lesoil, Georges Aaron Randrianaina
Univ. Rennes 1, Inria, IRISA
Rennes, France
{firstname[-name2].lastname}@irisa.fr

Djamel Eddine Khelladi
CNRS, IRISA, Univ. Rennes 1
Rennes, France
djamel-eddine.khelladi@irisa.fr

Mathieu Acher
Univ. Rennes 1, IUF, Inria, IRISA
Rennes, France
mathieu.acher@irisa.fr

ABSTRACT

Feature toggling is a technique for enabling branching-in-code. It is increasingly used during continuous deployment to incrementally test and integrate new features before their release. In principle, feature toggles tend to be light, that is, they are defined as simple Boolean flags and used in conditional statements to condition the activation of some software features. However, there is a lack of knowledge on whether and how they may interact with each other, in that case their enabling and testing become complex. We argue that finding the interactions of feature toggles is valuable for developers to know which of them should be enabled at the same time, which are impacted by a removed toggle, and to avoid their mis-configurations. In this work, we mine feature toggles and their interactions in five open-source projects. We then analyse how they are realized and whether they tend to be multiplied over time. Our results show that 7% of feature toggles interact with each other, 33% of them interact with another code expression, and their interactions tend to increase over time (22%, on average). Further, their interactions are expressed by simple logical operators (*i.e.*, `and` and `or`) and nested `if` statements. We propose to model them into a Feature Toggle Model, and believe that our results are helpful towards robust management approaches of feature toggles.

CCS CONCEPTS

• **Software and its engineering → Maintaining software**.

KEYWORDS

feature flags, continuous deployment, interaction of feature toggles

ACM Reference Format:
Xhevahire Tërnava, Luc Lesoil, Georges Aaron Randrianaina, Djamel Eddine Khelladi, and Mathieu Acher. 2022. On the Interaction of Feature Toggles. In *Proceedings of the 16th International Working Conference on Variability Modelling of Software-Intensive Systems (VAMOS '22), February 23–25, 2022, Florence, Italy*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3510466.3510485

1 INTRODUCTION

Large companies, such as Google [22, 23], Microsoft [20], Hewlett-Packard [5, 8], and Facebook [6], among others [15], have recently adopted trunk-based development by employing feature toggles. Feature toggles [1] are recognized as a powerful technique for modifying the system behaviour without changing code [7]. Technically, a feature toggle is a binary condition that controls whether a feature appears in the system or not [14]. In other words, feature toggles make possible trunk-based development by branching directly in code. Thus, avoiding the burden of merging features branches [18]. They are particularly used for continuous deployment, such as for dark launches, canary releases, or A/B testing [6, 18, 20].

Several works study or share the knowledge regarding different aspects of feature toggles, namely their good, bad, ugly practices [5], removal [10, 24], common used practices by practitioners [13, 22], wide usage in public projects [15], or differences and commonalities with configuration options [16]. However, to the best of our knowledge, the literature lacks the evidence on the interaction complexity of feature toggles within a codebase.

Feature toggles are perceived as a simple and light technique, and testing all their added paths in code seems not to be an issue [18]. But, are they actually simple to deal with in real projects, especially over the time? For developers, the difficulty is to know which toggles should be enabled in production at the same time [18] and to automate their maintenance [24], especially their removal when they are ready for final release. A wrong enabling or a bad practice of feature toggles can cause the bankruptcy of the organization (*e.g.*, [26]). Then, as feature toggles usually have a short lifespan [7, 16], they can quickly lead to some technical debt [24]. Our hypothesis is that interactions among feature toggles can hinder even more their enabling and make troublesome their removal.

To better understand whether feature toggles interact within a codebase, how they are implemented, how they are documented, and whether they have a tendency to multiply over time, we mined feature toggles in five popular open-source projects (*cf.* Section 3). The obtained results are reported in Section 4, which reveal that 7% of feature toggles interact with each other and 33% of them interact with another code expression, that is, that is not a feature toggle. Knowing their interactions is beneficial especially to developers in order (1) to prevent any system mis-configuration, (2) to assist developers in charge of maintaining feature toggles, and (3) to enrich the documentation of feature toggles with their interactions. We believe that our results show the need to extend the feature toggles management frameworks (*e.g.*, [19]), and maybe also the variability management approaches (*e.g.*, [2, 11]), with the documentation of feature toggles and their interactions.

In summary, our contributions are: *(i)* to the best of our knowledge, a first empirical evaluation of interaction of feature toggles, *(ii)* the proposition of a Feature Toggle Model (`FTM`) to model their

[1] They are also often referred to as feature flags, bits, gates, or flippers.

VAMOS '22, February 23–25, 2022, Florence, Italy

ACM ISBN 978-1-4503-9604-2/22/02…$15.00
https://doi.org/10.1145/3510466.3510485

interactions, and *(iii)* the availability of our automated mining approach with the obtained data for five popular software systems.

```
var (   //The rest of 21 feature toggles are omitted
  TerraformJSON = new("TerraformJSON", Bool(false))
  TerraformManagedFiles = new("TerraformManagedFiles", Bool(true))
) // The rest of code is omitted
```

Listing 1: Defined: kops/pkg/featureflag/featureflag.go

```
1  func (t *TerraformTarget) Finish(taskMap map[string]fi.Task) error
       {
2    //The rest of code is omitted
3    if featureflag.TerraformJSON.Enabled() {
4      if featureflag.TerraformManagedFiles.Enabled() {
5        return errors.New("TerraformJSON cannot be
6        used with TerraformManagedFiles")
7      }
8      err = t.finishJSON()
9    } else {
10     err = t.finishHCL2()
11   }
12   return nil
13 }
```

Listing 2: Usage: ../upup/pkg/fi/cloudup/terraform/target.go

2 BACKGROUND AND MOTIVATION

Feature toggles allow the developers to decide when and for whom to enable or disable some features (*i.e.*, functionalities) in a given software. Most often, a feature toggle is a constant, that can be `true` or `false`. Its value is evaluated in an `if-else` statement, which is used to surround a block of code. Up to four different kinds of feature toggles are distinguished (*i.e.*, release, experiment, ops, and permission) [7], which may have different binding times (*i.e.*, compile, start-up, periodic, or activity) [10]. Hence, there is a fuzzy boundary between them and the concept of configuration options [16]. Their apparent difference is that an added feature toggle is expected to be removed after few hours, weeks, and rarely more, that is, as soon as the guarded feature becomes stable [7, 10]. Whereas, a configuration option is more permanent in the system.

As an illustration example, in Listings 1 and 2 are shown two feature toggles that are used in the *Kops* system. *Kops* is also among the used subjects later in this study (*cf.* Section 3.2). First, as in Listing 1, two feature toggles are defined, usually all of them within a single file for the whole system [13, 23]. Then, as in Listing 2, they are used in a *toggle point* (*i.e.*, `if-else` statement) to condition the activation of system features. For instance, whenever the feature toggles `TerraformJSON` and `TerraformManagedFiles` are enabled then the surrounded block of code in lines 5-6 in Listing 2 will be part of the *Kops* system, otherwise the code in line 10 will be included. But, as shown in Listing 1, `TerraformJSON` has by default a `false` value whereas `TerraformManagedFiles` has a `true` value. If both of them are enabled then the system will throw an error. This kind of error could be prevented beforehand, for example, by documenting the feature toggles interactions [2]. The interaction from the *block of code viewpoint* is that they should not be enabled at the same time, otherwise the system will be mis-configured. But, from the *feature toggles viewpoint*, in order to evaluate the feature toggle in line 4, the feature toggle in line 3 is required first to be evaluated, that is, `TerraformManagedFiles implies TerraformJSON`.

[2]It is also reported in a recent issue: https://github.com/kubernetes/kops/pull/12341.

The structure of feature toggles seems less simple than expected, which motivates our work in this study. First, knowing how a feature toggle is scattered in the codebase is important to manage it, for example, `TerraformManagedFiles` is also used in two other toggle points in *Kops*. Secondly, some feature toggles can be unused or deprecated, which can lead to technical debt [20, 24]. Thirdly, as shown in Listings 1 and 2, feature toggles may interact and knowing their interactions can be beneficial to developers. For example, `TerraformJSON` toggle is deprecated [3], but during its removal one needs to know that it interacts with `TerraformManagedFiles`. Currently their interaction is not documented [4]. Moreover, the feature-toggling in *Kops* is supported by the `featureflag` package, but it has no functionalities to show the feature toggles interactions.

Knowing the places and interactions of feature toggles, including the unused ones, can be helpful for developers during their testing, system configuration, and their removal when they are ready for final release.

3 STUDY DESIGN

In this section, we provide the study design and our approach.

3.1 Research questions

The goal of this study is to explore the presence of feature toggles interactions in a system from the developer's viewpoint.

RQ_1 : **Do feature toggles interact with each other in order to enable some system functionalities (*i.e.*, features)?** We want to explore whether the enabling of a feature toggle requires other feature toggles, or more than one of them is used to enable some system functionalities. To this end, we mine the feature toggle points in five systems and analyse them regarding the feature toggles interactions.

RQ_2 : **Do feature toggles and their interactions have a tendency to be multiplied over time?** Presuming that feature toggles interact in a codebase, we then analyse whether the interactions of feature toggles tend to multiply over the years and if their change is proportional to the number of feature toggles. To this end, we extract the number of feature toggles and their interactions in the first release for each of the last five years of five subject systems.

3.2 Subject systems

To answer the research questions, we analysed the feature toggles in five Go public projects shared in GitHub, namely in *Boulder*, *Juju*, *Kops*, *Kubernetes*, and *Loomchain*. In the Stack Overflow 2021 survey results [5], Go is one of the top-10 most loved and wanted programming languages among developers. *Boulder* is an implementation of an ACME-based Certificate Authority, *Juju* is a framework that improves the experience of running Kubernetes operators, *Kops* is used to set up Kubernetes clusters swiftly, then is the *Kubernetes* itself, a system for managing containerized applications across multiple hosts, and *Loomchain* is an interoperable blockchain engine.

[3]See https://github.com/kubernetes/kops/pull/12341

[4]See https://github.com/kubernetes/kops/blob/9916733b3187338acbdccbf80396c7e3ffde8637/docs/advanced/experimental.md

[5]https://insights.stackoverflow.com/survey/2021#technology

Table 1: The five subject systems with their resulting # of feature toggles (FTs), # of logical relations, and the scattering of FTs in files. Where, `Indep.` - FTs with no interactions, `Inter.` - FTs into interactions, and `No` - toggle points with no logical relations

System	Commit	#LOC	#Feature Toggles				# Toggle Points					#Files		
			Total	*Unused*	*Indep.*	*Inter.*	*Total*	*No*	*And*	*Or*	*Implies*	*Total*	*With FT*	*Range*
Boulder	f7c6fef	515,186	17	5	6	6	23	13	16	0	1	2,332	16	0-2
Juju	89faeee	1,117,797	13	1	9	3	37	16	11	14	1	6,602	49	0-12
Kops	5f1d95c	2,608,174	23	1	11	11	69	58	10	4	13	11,078	29	0-11
Kubernetes	16227cf	4,581,828	108	31	24	53	278	170	130	8	24	17,503	297	0-23
Loomchain	350994c	102,471	55	21	21	13	109	91	16	8	10	472	39	0-7

We have chosen them mostly because all of them use feature toggles [13, 15]. They are also popular projects with 4k, 1.9k, 13.4k, 18.2k and 136 stars, respectively, in their repositories[6]. In addition, all projects have a different number of feature toggles, between 13 and 108. In Table 1 are given their last analysed commit ID, number of feature toggles, lines of code (LoC), and number of files.

3.3 Our approach

We build an automated approach to analyse feature toggles and their interactions in five subject systems. It consists of three main steps: (1) a specific version of a given system is cloned, (2) it is statically analysed, by relying on the Go parser of *tree-sitter* [17] [7], and then (3) the toggle points in its codebase are mined and analysed.

In the step (3), we first extract the defined feature toggles in the system. Usually, they are defined in a single dedicated file for the entire system. For instance, as shown in Listing 1, all 23 feature toggles in *Kops* are defined in the `./pkg/featureflag/featureflag.go` file. Next, all toggle points are extracted. That is, we track all `if-else` statements and extract those that contain a feature toggle name in its condition. For example, out of Listing 2 we extract lines 3 and 4, that is, `featureflag.TerraformJSON.Enabled()` and `featureflag.TerraformManagedFiles.Enabled()`, which contain the two defined feature toggles given in Listing 1. Afterward, we analyse whether the feature toggles interact within a toggle point or among them. We differentiate and look for the three found types of interactions: `and`, `or`, and `implies`. In the case of Listing 2, we record the `implied` relation between `TerraformJSON` and `TerraformManagedFiles`. The extracted data during these steps are particularly used to answer the RQ_1. For the RQ_2, we count the number of feature toggles for all available versions of the project. Concretely, we track the changes in the feature toggles file by listing all of them, by using git. Next, we repeat steps (1), (2), and (3) for 5 different releases of each system (one per year, from 2017 to 2021). All other technical details, data, and procedures are also available at https://github.com/llesoil/poc_ftm.

4 RESULTS

We now report the results with regard to our research questions.

4.1 Feature toggles interactions (RQ1)

To answer the first research question, we analysed the defined feature toggles within a system, their structure in toggle points, and how they are scattered in files. We conducted this same analyses for a recent release in all five subject systems (*cf.* Commit in Table 1).

The obtained results are summarized in Table 1. Based on them, on average for all systems, 33% of feature toggles are always used alone in a toggle point (*i.e.*, to guard a block of code). Then, 27% of the defined feature toggles are never used in a toggle point, that is, they are unused. Whereas, 40% of them are in an interaction between themselves or with another expression in code. The smallest number of feature toggles that are involved in an interaction are in *Juju* (23%) and the largest number are in *Kubernetes* (49%). A deeper analysis revealed that most of these interactions are between a feature toggle and another code expression, for example, `if(featuretoggle`$_x$ `&& revokedby != 0){...}` in *Boulder*. Excluding these cases, then only 12% of feature toggles interact at least once with each other in *Boulder*, 15% in *Juju*, 22% in *Kops*, 2% in *Kubernetes*, and 7% in *Loomchain*. On average, only 7% of feature toggles interact with each other. In all these cases, the maximum number of feature toggles involved in an interaction is 2.

Apart from it, the majority of toggle points contain a single feature toggle (67%). In the rest of the toggles points, the most often used logical relation is the `and` relation (69%). Less are used the `or` and `implies` relations (13% and 18%, respectively). Further, from the last three columns in Table 1, it can also be noted that all feature toggles are scattered up to a certain number of files, namely, between 0.26% and 8.26% of files. On average for all systems, all feature toggles are contained in only 1.13% of a system's files. Yet, a single file contains from 0 up to 23 feature toggles in maximum.

As an answer to the RQ_1, a small percentage of feature toggles tend to interact with each other (7%). Most often they interact with the rest of the codebase (33%). Further, only up to 2 feature toggles are involved in an interaction. Then, a considerable number of feature toggles are unused (27%). which may lead to technical debt. Specifically, feature toggles are considered as just another source of technical debt, as they are easy to be added in the short term, but as longer as they remain in the code the more they will cost the organization [4]. Hence, the unused feature toggles need to be deleted from the codebase. Yet, all used feature toggles are concentrated in a few files (1.13%). These results reveal the structure of feature toggles which tend to be relatively simple to be managed.

4.2 Multiplication of interactions (RQ2)

To answer RQ_2, we first extracted the number of feature toggles since their first introduction in each of the five subject systems. Their varying number is shown in Figure 1. Each dot represents a change in the file with the defined feature toggles. It can be

[6] Based on our last check on 01/11/2021.
[7] See https://tree-sitter.github.io/

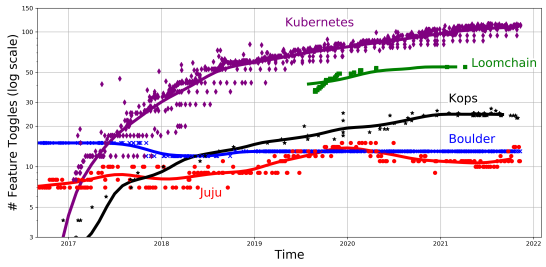

Figure 1: Evolution of feature toggles (y-axis, log scale) over time (x-axis) - Each point represents a change of toggles

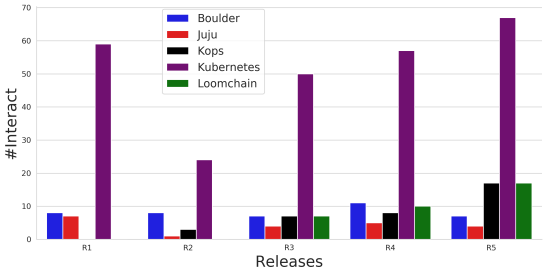

Figure 2: Evolution of interactions for the five systems

noted that the number of feature toggles in *Kops*, *Kubernetes*, and *Loomchain* tends to multiply over time. For the two other projects, *Boulder* and *Juju*, it varies but tends to remain the same in general.

Further, we extracted the number of feature toggles interactions for 5 releases (the first one of the last 5 years) of each subject. The obtained results are given in Figure 2. By comparing the last year with the average of previous years, results show that in *Boulder* and *Juju* the number of interactions tends to decrease by 18% and 6%, respectively. Whereas, they tend to be increased in *Kops* (65%), *Kubernetes* (29%), and *Loomchain* (39%). Based on the calculated Pearson correlation [3], the number of feature toggles interactions and the overall number of feature toggles is mostly correlated in *Kops* (0.90), *Kubernetes* (0.39), and *Loomchain* (0.88). In other systems, namely *Boulder* (-0.02) and *Juju* (0.04), they are less correlated. It should be noted that a given interaction fades away with the removal of its feature toggles. For example, when the `TerraformJSON` feature toggle is removed [8], so is the dependency in Listing 2.

As an answer to RQ_2, the feature toggles interactions tend to multiply over time (by 22%). Further, in 3 from 5 systems there is a close correlation between their number and the overall number of feature toggles in a codebase.

4.3 An extracted Feature Toggle Model

As a considerable number of feature toggles may interact in a codebase (40%) and their number tend to multiply over time (22%), we propose a representation of them, so the developers become aware of which and how feature toggles interact.

From our observations, all five systems use a framework or package to define, set, and evaluate the set value of feature toggles. But, in all cases, the interactions between feature toggles are realized using the two simple logical operators (*i.e.*, `or` and `and`) or nested `if` statements. Hence, to extract the interactions among feature toggles one has to analyse the structure of the code. In our knowledge, this task is not supported by the used feature toggles frameworks.

Therefore, by using our presented approach in Section 3.3, we mined the interactions of feature toggles in five subjects and propose to represent them in a Feature Toggle Model (`FTM`). In Figure 3 is shown an excerpt of the `FTM` of *Kops*, with 9 from its 23 feature toggles. It is a weighted graph where vertices represent the feature toggles, edges their interactions, edge weight the occurrence of an interaction, the node weight the independent occurrence of a feature toggle, and the node `Expr` marks any other expression in code.

[8]The commit with which this feature toggle in `Kops` is removed: e31dd98.

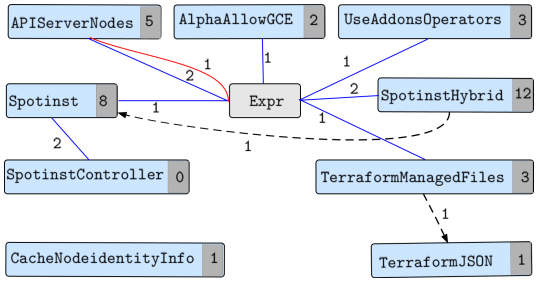

Figure 3: An excerpt from the Feature Toggles Model of *Kops*

Further, the blue edge marks an `and` interaction, the red edge the `or` interaction, and the dashed oriented edge the `implies` interaction. By using this graph representation, for example, one can easily spot the `implied` interaction between `TerraformManagedFiles` and `TerraformJSON` in *Kops*, which is presented in Listing 2. Furthermore, `TerraformManagedFiles` has another `and` interaction with a code expression, and is spread in 3 more places in code.

Having an `FTM` can be useful in particular to test the guarded features or to remove feature toggles. For instance, if a system feature is guarded by more than one feature toggle then the `FTM` can be helpful to figure out which ones may need to be enabled at the same time. Then, for instance, the responsible developer for `TerraformManagedFiles` toggle in *Kops* can be different from the one that is responsible for `TerraformJSON`. In this case, the `FTM` suggests that, because of the identified toggles interaction, both developers need to be notified during the removal of `TerraformJSON`.

By using our approach, we automatically extracted the `FTM` for each of the five subject systems. All of them are made available[9]. In addition, considering the short-lived nature of feature toggles, our approach can be used to extract their `FTM` whenever it is required.

4.4 Threats to validity

Internal threats. With our approach, we mined only those feature toggles that were evaluated in `if` statements. There are chances that they are evaluated also in `case` statements. Another threat is the mining of `implied` interactions in nested functional calls. During a manual analysis, we have encountered none of them. But, we noticed that sometimes a feature toggle is assigned to a variable

[9]Browse the pdfs in https://github.com/llesoil/poc_ftm/tree/main/results/FTM

which is after used within an `if` statement. We could have missed these cases with our mining approach, which does not apply data-flow or control-flow analysis. Still, we noticed only a few of them. Their absence has less impact on our results. Moreover, we focus on the mainstream usage of toggles, that is, in `if` statements.

External threats. Although our current approach is used to analyse only the Go projects, it can be easily extended to analyse the projects in other languages by using other tree-sitter parsers [10]. Further experiments are needed to make our results more conclusive and increase their generalizability.

5 RELATED WORK

Feature toggling is often discussed in grey literature by practitioners [5, 7, 9, 14, 27]. It is heavily used in industrial settings, for trunk-based development, and a considerable number of feature toggles management frameworks exist (*e.g.*, [19]). But, only recently it has received attention in academic research. Regardless, most of the works target only some aspects of feature toggles, such as their adaptation by finding their best practices [5, 13, 13, 18, 22, 23], their classification and implementation [7, 14], their removal [10, 24], and also their differences with configuration options [16, 21]. To our knowledge, none of the current works discuss the interactions of feature toggles. Hence, compared to them, we show how feature toggles interact and believe that their modeling can be useful for their management. Besides, there are approaches that analyse the complexity of preprocessor directives in C-based systems [12]. In contrast to them, we analyse the complexity of feature toggles.

On the other hand, there are abundant works on variability modeling (*e.g.*, into feature models) [2, 11], its extraction [25], and management [1]. But, adapting the existing variability-aware approaches [28] to analyze feature toggle points (an alternative form of presence conditions) can be valuable. Herein, we proposed a new extraction and modeling approach of feature toggles.

6 CONCLUSION AND FUTURE WORK

Feature toggles are perceived as a light technique for enabling trunk-based development. This work explores the interaction among them by analysing five open-source software systems. We provide an approach to extract the structure of feature toggles and show that they interact in 40% of the cases, but only 7% between each other. Still, we believe that they need to be modeled into a Feature Toggle Model, as their interactions tend to multiply over the time (22%).

As future work, we plan to extend this study *(i)* by analysing the interaction of feature toggles in more systems, *(ii)* to further improve the modeling of their interaction into a Feature Toggle Model (FTM), and *(iii)* possibly to integrate our approach in feature toggles management frameworks. In particular, we would like to empirically evaluate the realization of feature toggles by feature toggles management frameworks, including their interaction, and to further improve the construction and formalization of the FTM.

ACKNOWLEDGMENTS

This research was funded by the SLIMFAST N°20810047 and the ANR-17-CE25-0010-01 VaryVary projects.

[10] For example: https://tree-sitter.github.io/tree-sitter/#available-parsers

REFERENCES

[1] Nicolas Anquetil, Uirá Kulesza, Ralf Mitschke, Ana Moreira, Jean-Claude Royer, Andreas Rummler, and André Sousa. 2010. A model-driven traceability framework for software product lines. *Software & Systems Modeling* 9, 4 (2010), 427–451.
[2] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. 2016. *Feature-oriented software product lines.* Springer, Springer.
[3] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. 2009. Pearson correlation coefficient. In *Noise reduction in speech processing*. Springer, Spr, 1–4.
[4] Jim Bird. 2014. Feature Toggles are one of the Worst kinds of Technical Debt. https://dzone.com/articles/feature-toggles-are-one-worst.
[5] Andy Davies. 2018. Feature Toggles The Good, The Bad, and The Ugly with Andy Davies. https://www.youtube.com/watch?v=r7VI5x2XKXw.
[6] Dror G Feitelson, Eitan Frachtenberg, and Kent L Beck. 2013. Development and deployment at facebook. *IEEE Internet Computing* 17, 4 (2013), 8–17.
[7] Martin Fowler. 2021. Feature Toggles (aka Feature Flags). https://martinfowler.com/articles/feature-toggles.html.
[8] Gary Gruver. 2014. The Amazing DevOps Transformation Of The HP LaserJet Firmware Team. https://itrevolution.com/the-amazing-devops-transformation-of-the-hp-laserjet-firmware-team-gary-gruver.
[9] Santosh Hari. 2020. Feature flags: the toggle, the A/B test and the canary - NDC Oslo 2020. https://www.youtube.com/watch?v=FD5fX02QCmY.
[10] Juan Hoyos, Rabe Abdalkareem, Suhaib Mujahid, Emad Shihab, and Albeiro Espinosa Bedoya. 2021. On the Removal of Feature Toggles. *Empirical Software Engineering* 26, 2 (2021), 1–26.
[11] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. 1990. *Feature-oriented domain analysis (FODA) feasibility study.* Technical Report. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.
[12] Jörg Liebig, Sven Apel, Christian Lengauer, Christian Kästner, and Michael Schulze. 2010. An analysis of the variability in forty preprocessor-based software product lines. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. NY, USA, ACM, 105–114.
[13] Rezvan Mahdavi-Hezaveh, Jacob Dremann, and Laurie Williams. 2021. Software development with feature toggles: practices used by practitioners. *Empirical Software Engineering* 26, 1 (2021), 1–33.
[14] Mark McKenna and Josh Allen. 2016. Feature Toggles: Lunch & Learn. https://www.youtube.com/watch?v=gxm1C92XhCQ.
[15] Jens Meinicke, Juan Hoyos, Bogdan Vasilescu, and Christian Kästner. 2020. Capture the feature flag: Detecting feature flags in open-source. In *Proceedings of the 17th International Conf. on Mining Software Repositories*. ACM, NY, USA, 169–173.
[16] Jens Meinicke, Chu-Pan Wong, Bogdan Vasilescu, and Christian Kästner. 2020. Exploring differences and commonalities between feature flags and configuration options. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*. ACM, NY, USA, 233–242.
[17] Iulian Neamtiu, Jeffrey S Foster, and Michael Hicks. 2005. Understanding source code evolution using abstract syntax tree matching. In *Proceedings of the 2005 international workshop on Mining software repositories*. ACM, NY, USA, 1–5.
[18] Steve Neely and Steve Stolt. 2013. Continuous delivery? easy! just change everything (well, maybe it is not that easy). In *2013 Agile Conf.* IEEE, IEEE, 121–128.
[19] Roy Osherove. 2021. Feature Toggle Framework List. https://pipelinedriven.org/feature-toggle-frameworks-list/.
[20] Chris Parnin, Eric Helms, Chris Atlee, Harley Boughton, Mark Ghattas, Andy Glover, James Holman, John Micco, Brendan Murphy, Tony Savor, et al. 2017. The top 10 adages in continuous deployment. *IEEE Software* 34, 3 (2017), 86–95.
[21] Eduardo S Prutchi, Heleno de S. Campos Junior, and Leonardo GP Murta. 2021. How the adoption of feature toggles correlates with branch merges and defects in open-source projects? *Software: Practice and Experience* 52, 2 (2021), 506–536.
[22] Md Tajmilur Rahman, Louis-Philippe Querel, Peter C Rigby, and Bram Adams. 2016. Feature toggles: practitioner practices and a case study. In *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, NY, USA, 201–211.
[23] Md Tajmilur Rahman, Peter C Rigby, and Emad Shihab. 2019. The modular and feature toggle architectures of Google Chrome. *Empirical Software Engineering* 24, 2 (2019), 826–853.
[24] Murali Krishna Ramanathan, Lazaro Clapp, Rajkishore Barik, and Manu Sridharan. 2020. Piranha: Reducing feature flag debt at uber. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*. ACM, NY, USA, 221–230.
[25] Julia Rubin and Marsha Chechik. 2013. A survey of feature location techniques. In *Domain Engineering*. Springer, Springer, 29–58.
[26] Doug Seven. 2014. Knightmare: A DevOps Cautionary Tale. https://dougseven.com/2014/04/17/knightmare-a-devops-cautionary-tale/.
[27] Split. 2020. Feature Flag Maintenance. https://www.youtube.com/watch?v=qbVNbMSzy0.
[28] Alexander Von Rhein, Alexander Grebhahn, Sven Apel, Norbert Siegmund, Dirk Beyer, and Thorsten Berger. 2015. Presence-condition simplification in highly configurable systems. In *2015 IEEE/ACM ICSE*, Vol. 1. IEEE, IEEE, 178–188.