Feature Models in Linux – From Symbols to Semantics*

Valentin Rothberg
Friedrich-Alexander-Universität Erlangen-Nürnberg
rothberg@cs.fau.de

Nicolas Dintzner
Delft University of Technology
N.J.R.Dintzner@tudelft.nl

Andreas Ziegler
Friedrich-Alexander-Universität Erlangen-Nürnberg
ziegler@cs.fau.de

Daniel Lohmann
Friedrich-Alexander-Universität Erlangen-Nürnberg
lohmann@cs.fau.de

ABSTRACT

Linux is a highly configurable operating-system kernel which has been widely studied in the context of software product lines over the past years. Understanding the challenges and perils of evolving and maintaining feature models of the size of Linux is crucial to provide the right tools for development today and to direct future research. Unfortunately, previous studies show contradictory observations when analyzing the evolution of Linux feature models. We explain how peculiarities of the feature models of the Linux kernel lead to those differing observations, and show how the results can be re-aligned. Moreover, our findings also demonstrate that symbolic differencing on feature models used by researchers so far has limited value, depending on the use case. We show how the limitations can be addressed by means of semantic differencing, and ironically invalidate the results we sought to re-align.

CCS Concepts

•**General and reference** → Experimentation; Reliability; Management; •**Software and its engineering** → Software configuration management and version control systems;

Keywords

Configurability, Linux, Kconfig, Feature Models, CADOS

1. INTRODUCTION

The open-source code base of Linux attracted the contributions of hundreds of companies and thousands of individuals over the past 24 years and lead to a feature-rich and open operating system (OS). With more than 15,200 configurable features in version 4.2, Linux is able to run in literally countless configurations on numerous hardware architectures and platforms – from small mobile devices to extremely tailored high-speed trading systems. Feature models (FMs) of such size present an important real-world case study to variability management and to software product lines (SPLs) in general.

Managing the variability of feature-rich software systems such as Linux is difficult due to the number of features and the complexity of the relationships among them. Mistakes when updating features can lead to various kinds of errors that may manifest in inconsistencies between the feature model and the implemented variability (e.g., #ifdefs, build system, etc.) at compile time [5, 6, 9], or in complex bugs that can only be revealed during execution of the kernel [10, 11]. As those studies have shown, it is important to understand the FM of Linux in order to *properly* evolve the system and to develop tools for researchers and developers, for instance, to detect dead code [5]. Thorough tool support is especially needed for extracting variability information from the FMs, which is barely possible by hand, even for experienced developers [13] . As part of the CADOS[1] project we have been developing tools, such as undertaker,[2] to detect and analyze inconsistencies between the configuration space (i.e., configuration files) and the implementation space (i.e., source files, headers and build system). In cooperation with kernel developers, we contribute parts of our tools upstream[3] and run daily consistency checks with undertaker-checkpatch on newly integrated git commits in linux-next (i.e., the Linux git repository for continuous integration) and accordingly report detected issues to the responsible developers and maintainers. Until now, our daily checks include the detection of dead code in source files [5], and of finding variability bugs in and induced by the build system [6]. We plan to extend the functionality of undertaker-checkpatch to also analyze changes applied to the Kconfig feature model of Linux. For this purpose, we require a fine-grained diff of feature models in order to map them to changes of Kconfig configuration files. Having fine-grained FM diffs allows us to analyze the causes of variability bugs in greater detail, for instance, when

*This work was partly supported by the German Research Council (DFG) under grant no. LO1719/3-1

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

VaMoS '16, January 27-29, 2016, Salvador, Brazil
© 2016 ACM. ISBN 978-1-4503-4019-9/16/01…$15.00
DOI: http://dx.doi.org/10.1145/2866614.2866624

[1] https://cados.cs.fau.de
[2] https://undertaker.cs.fau.de
[3] scripts/checkkconfigsymbols.py

certain changes to configuration files cause contradictory #ifdef conditions in the code [5].

A tool to generate fine-grained diffs of feature models in Linux has first been proposed by Dintzner et al. [12] in the form of FMDiff.[4] The authors were especially interested in the question which architectures are affected by changes of the FM. Since Kconfig configuration files in Linux are hierarchically organized by architectures, a change applied to such files may only affect some but not all architectures. Analyzing the impact of a change helps to know exactly which architectures need to be (build) tested, and can thereby improve performance of current testing frameworks of the Linux community since only affected and not all architectures will be checked. By reading the related papers [12, 15] about FMDiff, we stumbled across a seemingly contradictory observation with previous work from our group. On one hand, Dietrich et al. [7] state that more than 60 percent of all features are shared among the FMs. On the other hand, Dintzner et al. [15] discovered when diffing the FMs of Linux over a range of Linux versions, that changes to shared features affect the FMs in the same way but only 37 percent of changes affect all FMs. Since changes to code are proportional to its size [18], the results actually seem to be inverted. Besides a general research interest, it is important for us to reveal the causes of those inconsistent results since either previous research, FMDiff or the FMs of Linux suffer from *undiscussed* issues – our fine-grained consistency checks of git commits rely on their correctness.

Contributions

In this paper, we investigate the inconsistent observations of previous studies, and show that even small peculiarities of Kconfig and of configuration files can alter the underlying feature models and conclusions drawn from it to a high degree. By taking our observations and additional information from kernel developer Greg Kroah-Hartman into account, we can re-align and explain those results. Our findings also reveal certain properties of the FMs of Linux limiting the applicability of those models without further reasoning. In particular, we raise the issue of *feature selectability* in feature models: the symbolical presence of a feature in a given architecture-specific FM does not imply that the feature can actually be selected for the given architecture. The selectability of features has considerable impact on the interpretation and use of the results of previous studies. We thereby invalidate the common and often cited assumption that Linux architectures share more than two thirds of features and that they evolve similarly. Ironically, the motivation of this paper was to prove, not to invalidate this assumption – an unexpected journey. Furthermore, we propose a methodology based on SAT solving to address the issue of feature selectability in the context of feature-change classification.

The paper is structured as follows. We first give an overview of FMDiff and describe its workflow in Section 2. In Section 3, we explain the inconsistent results of previous studies, show where they stem from and analyze their causes. In Section 4, we raise the issue of feature selectability and discuss its impact on previous and future studies. We further show how the issue of feature selectablility can be addressed in the context of feature-change classifications.

[4]https://github.com/NZR/Software-Product-Line-Research

2. BACKGROUND

Supporting and maintaining the evolution of large scale software product lines offers challenges to both, industry and research. The Linux kernel, in particular, is under heavy development with 8,600 lines added, 5,800 lines removed and 2,100 lines modified – every day [18]. With nearly 4,000 active developers and 430 involved companies, Linux is the largest collaborative software project we know of [18]. The feature model of Linux shows remarkable growth rates as well. Given that Linux v2.6.12 (2005) was shipped with nearly 5,000 configurable features, the feature model of Linux has tripled in size hitting the 15,000 features mark with Linux v4.1 (2015).
To analyze the evolution of feature models and to examine changes applied to configuration files on a fine-granular level, Dintzner et al. [15] proposed an approach to diff the feature models of Linux in the form of FMDiff.[4] Taking two feature models as input, FMDiff compares both models in order to detect and classify changes of features. Figure 1 illustrates the workflow of FMDiff which can briefly be split into three steps:

(1) *Model Reconstruction*: A required step before comparing feature models is to translate the Kconfig configuration files into a textual representation that is easy to diff. FMDiff employs the Kconfig extractor *dumpconf* from the undertaker[2] suite, which parses Kconfig files and accordingly generates one feature model for each architecture in the Rigi Standard Format (RSF).
Selecting an architecture is the first mandatory step when configuring the Linux kernel. Due to the hierarchical organization of Kconfig files, choosing an architecture constrains which Kconfig files will be parsed and hence which Kconfig configuration options (i.e., features) are included in the FM of the architecture. Consequently, there is one FM and hence one RSF file per architecture. Moreover, undertaker-kconfigdump is based on the parser of Kconfig, which extracts and transforms the FM but does not evaluate any condition or expression. Hence, some parsed features may not be selectable since their constraints cannot be satisfied; a RSF file represents the FM of a given Linux architecture symbolically but not semantically.

(2) *Model Comparison*: As soon as the models are reconstructed in the RSF format, they can be compared. Taking two versions of one architecture-specific RSF model, FMDiff compares the attributes of each feature as defined by the Kconfig language. Therefore, FMDiff ships a meta model describing the Kconfig language and uses the Eclipse Modeling Framework to perform the differencing algorithm on both models.

(3) *Change Classification*: FMDiff uses a three level classification scheme [15] of feature changes, namely *change category* (i.e., feature modification, addition and deletion), *change sub-category* (e.g., modification of dependencies), and the *change type* describing the actually performed change. An exemplary diff of feature models is depicted in Figure 1. The change of the depends statement is classified as a modification of the feature BAR, while feature FOO2 is added to and feature FOO1 is removed from the FM. FMDiff stores

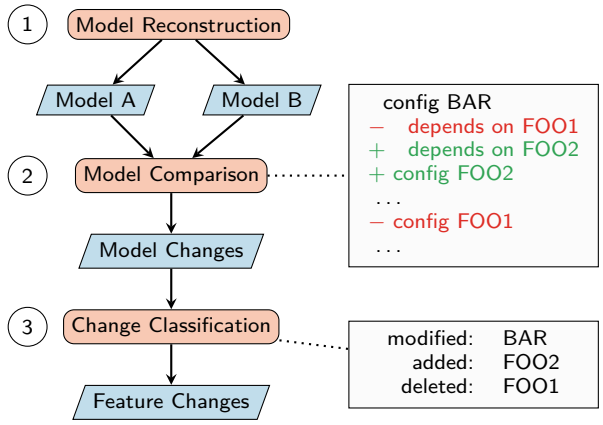

Figure 1: Diffing feature models with FMDiff.

the changes in a separate data base that can later be queried for further analysis.

Detailed descriptions of FMDiff and the Kconfig language can be found in [12, 15]. Note that in the context of this study, we reimplemented the functionality of FMDiff in Python, since we want to integrate the differencing classification into the undertaker suite. We compared our implementation with FMDiff, which is based on the Eclipse Modeling Framework (EMF), and receive almost equal results. When comparing boolean formulas, the EMF framework seems to be inaccurate in the detection of logical equivalence such that it sees a difference in formula ϕ_A (`A && B`) and formula ϕ_B (`B && A`). Our implementation uses limboole[5] to check if two given formulas are logically equivalent, and correctly detects boolean equivalence such as in the case of formula ϕ_A and ϕ_B.

3. FEATURE-MODEL EVOLUTION

In this section we describe in detail the observations made with FMDiff. We answer the question how and why those observations conflict with previous research, and highlight properties of the feature models of Linux that limit their usability for feature-change classifications.

3.1 Classifying Feature Changes

When diffing the architecture-specific feature models of Linux over a range of versions (v2.6.39 to v3.14) Dintzner et al. [15] discovered that "relatively few feature changes affect all architecture-specific FMs of the Linux kernel". In other words, if a feature is changed, the change is likely not to be visible in all feature models. This observation does not align with previous results, such that feature models of the Linux kernel evolve similarly with comparable growth rates [2] and that Linux architectures share more than 60 percent of all features [7]. By having a closer look at the feature distribution of Linux v4.2, we see that 67 percent of all features are shared among all feature models. Since changes to common features affect the feature models in the same way [15] and since the number of changes is directly proportional to the size of source code [18], one would expect that the majority of changes to configuration files – statistically around two thirds – affects all feature models and not only a subset of models. Hence, it is confusing that when using FMDiff between Linux v2.6.39 and v3.14, one observes that most changes (i.e., 63%) *do not* affect all FMs. In fact, the expected result is nearly inverted. This raises the question: *How do architecture-specific FMs really evolve?*

To answer this question, we must find out why the observations of previous studies differ. Let us first look at how FMDiff classifies changes applied to Kconfig configuration files with respect to affected architecture-specific FMs. The authors define an *architecture-generic change* as a change that occurs in *all* feature models, otherwise it is classified as an *architecture-specific change* [15]. Let us consider a change adding Kconfig option *FOO* to Kconfig. This change is classified as architecture-generic iff the change affects all architecture-specific FMs (i.e., *FOO* is added to all FMs). Otherwise it is classified as architecture-specific. Under the assumption that a generic change affects *all* feature models, the change classification of FMDiff is perfectly sound, but the results presented by Dintzner et al. [15] clearly indicate that there are more architecture-specific changes than architecture-generic changes. Hence, let us further investigate the analyzed data, the feature models.

Table 1 shows the *distribution* of features *over architectures* from Linux v2.6.39 (May 2011) to Linux v3.11 (September 2013), and hence shows how many features are shared among how many architectures. The table is to be read as follows: Linux v2.6.39 has 24 different architectures, 3,989 distinct features are included in only one architecture, 182 features are included in two architectures, 50 features are included in three architectures, 2,617 are included in all 24 architectures, etc. We can see that starting with Linux v3.11 the majority of features (i.e., 66.59%) is included in all architectures, which has not been the case in previous versions. Before version v3.11 there are clusters – displayed in dark gray – of cardinality three (v2.6.39 to v3.1) and two (v3.2 to v3.10) indicating that from version v2.6.39 until v3.10 the majority of features was present in *almost* all architecture-specific feature models. Thereby, the clusters in Table 1 are strong indicators for the dominance of architecture-specific changes as observed with FMDiff. Between Linux v2.6.39 and v3.10 only 22 percent to 34 percent of features were present in all feature models. Since changes are equally distributed, and since most features are not generic, most changes potentially could not affect all FMs resulting in more architecture-specific changes than architecture-generic changes as observed by Dintzner et al. [15]. By analyzing the features included in the clusters of Table 1 we make the following observations:

(a) The features in the clusters are included in all architecture-specific FMs except in the architectures um (v2.6.39 to v3.2), cris and h8300 (v2.6.39 to v3.11).

(b) 12 percent of those features are defined in the subsystem *sound*, 88 percent are defined in the *drivers* subsystem.

We find that in the analyzed Linux versions, only the architectures *um*, *cris* and *h8300* did not include the main configuration file of drivers (i.e., via "`source drivers/Kconfig`"). In 2003, `drivers/Kconfig` has been introduced to reduce duplication in Kconfig, since "[it was a] pain for architecture maintainers to keep the top-level Kconfig files in sync".[6] How-

[5] http://fmv.jku.at/limboole/

[6] git commit `31f42cec798d` from Linux history-git tree

Table 1: Distribution of features from Linux v2.6.39 to v3.11. How many features are shared among how many architectures?

Version	#arches	1	2	3	4–19	20	21	22	23	24	25	26	2	28	29	30
v2.6.39	24	3989	182	50	209	1	2293	944	1189	2617						
v3.0	24	3990	183	53	227	1	2345	968	1211	2637						
v3.1	25	4026	184	52	228	0	6	2440	968	1155	2667					
v3.2	26	4028	181	57	233	0	5	2	1	2788	512	4054				
v3.3	27	4077	180	51	232	5	0	6	1	1	2837	512	4133			
v3.4	27	4087	183	51	242	5	0	6	1	1	2907	520	4184			
v3.5	27	4129	179	50	243	0	0	6	1	2	3001	520	4265			
v3.6	27	4158	184	51	244	0	0	5	2	2	3098	527	4298			
v3.7	28	4139	183	50	248	0	0	0	6	2	1	3173	539	4384		
v3.8	28	4148	178	35	255	13	0	0	5	2	3	3269	548	4399		
v3.9	30	4269	177	36	261	14	0	0	0	1	6	0	3	3403	581	4413
v3.10	30	4280	173	35	273	13	1	0	0	1	6	0	3	3447	577	4460
v3.11	30	4270	178	33	288	16	1	0	0	1	6	0	2	0	0	8654

ever, um, cris and h8300 had not been converted to use this Kconfig file. Instead of including the generic driver Kconfig file, all three architectures included only such sub-directory configuration files that were really needed. As a consequence, many features of drivers were missing in either the FM of um, cris, h8300 or all of them. All features of the sound subsystem were affected as well, since `drivers/Kconfig` is the only place to include `sound/Kconfig`. Hence, the lack of um, cris and h8300 including the main driver Kconfig file explains the observation (a) and (b). By analyzing the git history of Linux, we find that um[7] has been converted with Linux v3.2, and that cris[8] and h8300[9] have been converted with Linux v3.11, which explains the change of the cluster's cardinality from three to two (v3.2) and from two to zero (v3.11) in Table 1.

3.2 Re-Aligning Results

The delayed conversion of `drivers/Kconfig`, explains the observation of Dintzner et al. [15] with FMDiff. Since most driver features and all features of sound have not been included in all architecture-specific FMs, there are more architecture-specific changes than generic changes between Linux v2.6.39 and v3.11. Still, this does not explain the inconsistent results with Dietrich et al. [7], who analyzed the distribution of features in Linux v2.6.35 and found that 62 percent of features are shared among the *main* architectures of the Linux kernel. However, Dietrich et al. excluded the architectures um, cris and h8300 from their analysis, which may explain why those studies produce seemingly inconsistent results – one included the other excluded um, cris and h8300. In order to investigate if the inclusion and exclusion of those architectures causes the different results, we apply the feature differencing algorithm of FMDiff on the range of previously analyzed [15] Linux versions (i.e., v2.6.39 to v3.14) by including and excluding um, cris and h8300 from analysis.

Table 2 compares the results. We applied the FM differencing as proposed by FMDiff, and classified changes once by including and once by excluding um, cris and h8300. Note, that we exclude those architectures only in versions where they are not yet converted to use `drivers/Kconfig` (i.e., um prior to v3.3, cris/h8300 prior to v3.12). In Table 2, we

[7]git commit `3369465ed1a6`
[8]git commit `acf836301e4b`
[9]git commit `9ac6adbcab11`

Table 2: Comparison of detected architecture-generic changes by including and excluding the architectures cris, um and h8300.

Version	Excluding	Including
v2.6.39	44.83%	14.99%
v3.0	72.66%	10.71%
v3.1	66.96%	11.25%
v3.2	73.36%	15.39%
v3.3	63.29%	23.67%
v3.4	59.81%	33.03%
v3.5	65.93%	40.24%
v3.6	72.79%	35.21%
v3.7	78.39%	28.73%
v3.8	53.91%	28.72%
v3.9	78.50%	53.31%
v3.10	55.33%	32.99%
v3.11	58.00%	12.54%
v3.12	47.93%	47.93%
v3.13	52.99%	52.99%
v3.14	53.46%	53.46%
Average	62.38%	30.94%

can see that excluding um, cris and h8300 reveals consistently more generic changes than including them in versions prior to Linux v3.12. After, the results remain equal. Those observations can be explained with the fact that prior to v3.12, the architectures um, cris and h8330 have not yet been converted to use `drivers/Kconfig`. Most features of drivers and all sound features have been missing in the FMs of those architectures, causing more architecture-specific than generic changes. The table also shows expected results for classifications when excluding those architectures. Since around 60 percent of features are shared among architectures [7] and thus are generic to the FMs, statistically around 60 percent of changes affect all FMs – 62.38 percent on average in our data. Hence, the table shows that excluding those architectures from analysis with FMDiff re-aligns the observations of Dietrich and Dintzner.

However, even though we can re-align previously inconsistent results with our data, previous work does *not* explain the observed data for um, cris and h8300. Hence, we are interested in the question why those three architectures have not been

converted for a considerable period of time. We analyzed the previously detected git commits that converted um, cris and h8300, but we could not detect any technical reason for those architectures not to include `drivers/Kconfig` before. Personal correspondence with Greg Kroah-Hartman, the maintainer of the driver core among many other subsystems of the Linux kernel, validates our conclusion that there was no technical reasons not to convert um, cris and h8300. In fact, he points out a much simpler reason: "people being lazy". The conversion to use the generic driver Kconfig file has first been applied to the x86 architecture, so that "it was up to the [remaining] arch maintainers to do the conversion, and given that there was no active maintainers for those arches, they never did it (or did it late as was the case with um)". Hence, the conversion of um, cris and h8300 had been delayed for a considerable amount of time – nearly 10 years – since those architectures had not been actively maintained. We argue that both approaches, including and excluding them from analysis, are valid since the inclusion compares the actual state of all architectures, whereas the exclusion of such reveals more similarities of actively maintained architectures.

To conclude, when analyzing the evolution of Linux FMs between Linux v2.6.39 and v3.14, Dintzner et al. [15] observed more changes being generic to some but not all architecture-specific FMs since the architectures um, cris and h8300 did not include the generic drivers configuration `drivers/Kconfig` in the analyzed range of Linux versions. Hence, most driver features and all sound features have been missing in the FMs of those architectures. On the other hand, Dietrich et al. [7] found that 62 percent of features are shared between the FMs of Linux, but they excluded um, cris and h8300 from analysis which explains the seemingly different observations. Personal correspondence with Greg Kroah-Hartman revealed that the conversion of um, cris and h8300 had been delayed since they remained unmaintained in the range of analyzed Linux versions.

However, the data presented in Table 1 and the exchange with Greg Kroah-Hartman emphasized another, yet more fundamental problem when it comes to the FMs of Linux: the pure symbolic presence of a feature in a shared, generic Kconfig file *does not* imply that this feature can be selected on all architectures. Hence, when using current techniques to know if a feature change affects one or more architecture-specific FMs, we ignore the fact that the architecture-specific FMs of Linux reflect Kconfig and its constraints *symbolically*, but *not semantically*. For instance, when considering drivers, only half of the available architectures support PCI so that all PCI drivers are selectable only for those architectures. As a consequence, we expect that many as architecture-generic classified changes do not affect all architectures semantically, since the changed feature may only be selectable for some but not all architectures. We discuss this issue in the following section and show how falsely classified changes can be detected by means of semantic filtering.

4. SYMBOLS VERSUS SEMANTICS

In the context of a software product line such as Linux, it is desirable to reduce or even avoid code redundancy to enable code sharing among different users (i.e., architectures) and to make the code more maintainable. Hence, all subsystems of Linux are designed to be architecture-generic. However, there are means to restrict code parts to certain features or architectures. By means of the C preprocessor, source-code artifacts can be conditionally compiled via #ifdef blocks, and entire source files are conditionally included into compilation via build-rules declared in Makefiles [6]. Similar in- and exclusion applies to the feature model of Linux as defined in Kconfig files. In Linux v4.2, 10,080 of all 15,204 features are included in every architecture-specific FM. However, it is important to understand that the symbolic presence of a feature in a FM does not imply that the feature can actually be selected for the given architecture – feature constraints need to be taken into account. Consider the following Kconfig code:

```
497 config PATA_CS5535
498     tristate "CS5535 PATA support (Experimental)"
499     depends on PCI && X86_32
500     [...]
```

Listing 1: Kconfig snippet from `drivers/ata/Kconfig` (Linux v4.2)

The feature, in the Kconfig language denoted as *configuration option*, PATA_CS5535 is defined in `drivers/ata/Kconfig`. This Kconfig file is included only by `drivers/Kconfig`, so that PATA_CS5535 is included in all architecture-specific feature models. Even though PATA_CS5535 is included in all FMs, it can be selected only on the x86 architecture due the dependency in line 499. Such cases, where features are included in all but selectable only for some architectures, raise several questions: Does it make sense to classify a feature and changes applied to it as part of an architecture-specific model if the feature does not exist semantically? Do architecture-specific FMs evolve similarly and really share two thirds of features?

4.1 Distribution of Features

To give a first impression of the size of the semantic problem, we compare the number of shared features that are selectable only for some architectures by analyzing the FMs of Linux v4.2 as follows:

(1) Collect a set F_{sym} of features that are included in all and hence shared among all architecture-specific FMs.

(2) Iterate over F_{sym} and check the satisfiability of the constraints of each feature by means of a SAT solver. If the feature can be selected on each architecture-specific FM, it is semantically generic.

We used undertaker-kconfigdump to generate the feature models in the RSF format, and employed PicoSAT [1] as a SAT solver. We find that only 2,523 of 10,080 symbolically shared features are actually selectable on all architectures. This means that architecture-specific feature models of the Linux kernel have *75 percent less* generic features as they may symbolically indicate.

We conclude that feature models of Linux *do not* share most features. In fact, only 17 percent (2,523) of all features (15,204) are semantically generic in Linux v4.2. In the following we measure the impact of feature selectability on the

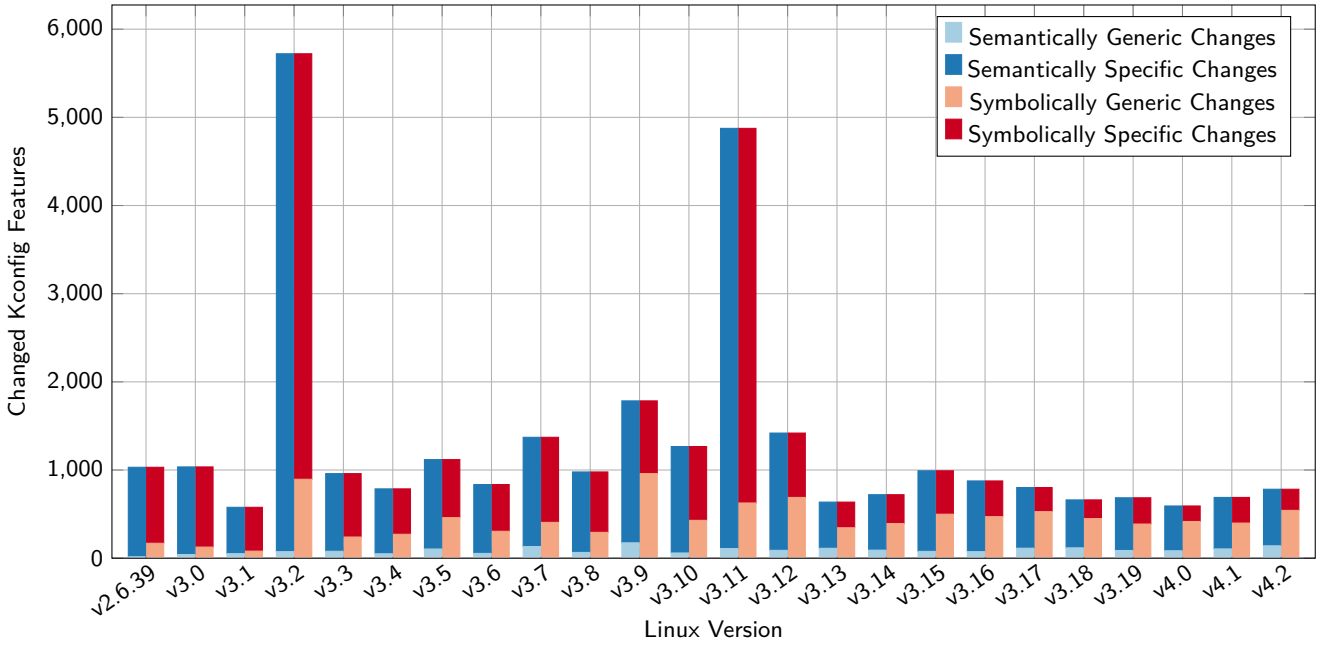

Figure 2: Comparison of semantic and symbolic feature-change classification.

evolution of FMs to answer the question if feature models of Linux evolve similarly.

4.2 Semantic FM Evolution

A major use case of FMDiff is described as determining the impact of a change to Kconfig in terms of build testing [15]. If a Kconfig option is changed, it is desirable to re-build only those architectures that are affected by the change. The Intel Open Source Technology Center, for instance, is build-testing over 180 different Linux development repositories, every single day [8]. Knowing which changes affect which architectures can greatly improve performance of such test-frameworks, since it avoids unnecessary builds of unaffected architectures. The gained time can be used for additional builds which thereby increases the amount of tested commits on a daily basis. However, current feature differencing techniques, as used by FMDiff, must be considered with care. If a change is classified to affect a given architecture, we must be sure that the changed feature can actually be selected on this architecture. Without this check, more architectures are build-tested than necessary, having a counter-productive effect on the purpose of enhancing current build-testing frameworks. Similar to the approach in Section 4.1, such false positives can be filtered by means of a SAT solver:

- A changed feature f affects architecture *arch* only if the feature constraints of f are satisfiable on the FM of *arch* (i.e., there is at least one valid configuration of *arch* containing feature f).
- Changes that do not affect the FM of a given architecture semantically need to be filtered.

We applied such *semantic filtering* of changes over a range of Linux versions, from v2.6.39 to v4.2. Figure 2 compares the results of architecture-generic and architecture-specific changes with and without semantic differencing. The results with semantic differencing are denoted as *Semantically Generic* and *Semantically Specific Changes*. The results without semantic differencing are denoted as *Symbolically Generic* and *Symbolically Specific Changes*. We can see that there are consistently more architecture-generic changes that impact the FMs symbolically than semantically. In total numbers, there are 1,689 changes between Linux v2.6.39 and v4.2 that semantically affect all FMs of Linux. From a symbolical point of view 9,968 changes affect all FMs.

On one hand, Figure 2 shows us that feature-change classifications without semantic differencing are highly imprecise: *93 percent* of architecture-generic changes are false positives. On the other hand, the figure also shows us that feature models of Linux evolve very differently. Only 5.8 percent of changes (i.e., 1,689 of 29,091) affect all FMs semantically. Thereby, we show that Linux architectures evolve very differently. Note, the peaks at Linux v3.2 and v3.11 are caused by the conversion of um, and cris and h8300 respectively to use the generic `drivers/Kconfig` configuration file. Those conversions caused the additions of all remaining drivers and sound features to the affected FMs which are thus classified as architecture-specific feature changes. We further conclude that current feature-differencing techniques do not work as expected in the context of build-testing. Since those techniques diff on a symbolic level, more architectures will be build-tested than necessary. Those false positives can be avoided by semantic filtering in order to re-assure that changes really affect a given architecture or not.

4.3 Discussion

Understanding the difference between the symbolical presence of a feature and its semantic selectability is crucial for any kind of study based on the feature models of Linux. Our results show that 2,523 of all 15,204 features in Linux

v4.2 are semantically generic. Furthermore, less than six percent of changes affect all architectures of Linux. We thereby show that feature models of Linux are *inherently different* from a technical point of view. How can a virtualized kernel architecture, such as User Mode Linux, share 60 percent of features with a bare-metal x86 kernel that supports thousands of different hardware devices? We want to make clear that for any research that is about *facts* of variability models in Linux, only a semantic approach makes sense. Otherwise, data is being measured that does not reflect reality and that – depending on the context – has no practical impact.

Our approach aims at avoiding false classifications by means of semantic filtering and can also be applied to generate *semantic models* (i.e., FMs that only include selectable features). By means of a SAT solver, the satisfiability of the feature constraints of each feature has to be tested. Hence, a model can be transformed into a semantic model by checking the satisfiability of each feature and by removing those that remain unsatisfiable. Semantic models can be interesting for qualitative studies on the feature models of a given architecture. However, our study and the presented approach is limited by the accuracy of the models extracted from Kconfig configuration files, which we discuss in the following section.

5. THREATS TO VALIDITY

The precision of our study is constrained by the accuracy and correctness of the extracted models, which we take as an input for diffing feature models. On one hand, inconsistent models may lead to false classifications (e.g., when dependencies are falsely propagated). On the other hand, an accurate translation of the semantics is crucial for reliably checking the satisfiability of feature constraints with respect to a given architecture.

El-Sharkawy et al. [16] have shown that the models generated by undertaker's dumpconf, among other tools, do not accurately reflect the semantics of Kconfig. The authors investigate the semantics of Kconfig in great detail and formulate several corner cases which are not translated correctly in the models we used for our analysis. Inaccuracies in the semantics of the model are especially unsatisfactory when solving a SAT problem, as in our case. However, at the current state the impact of the models' inaccuracies on analysis as such performed in this study is unclear; some unsupported cases affect only certain versions of the Linux kernel or do not practically impact its models (e.g., the `option module` case [16]). Most of the mentioned corner cases affect choices (i.e., mutually exclusive selects) in the Kconfig language. Since choices are barely changed in Linux [15], we do not expect major impacts on the presented results of our study.

Nonetheless, more accurate models can be generated with kconfigreader [17], developed by Kästner, which should be preferred over dumpconf in future Kconfig-based analysis and especially in production systems to improve accuracy.

6. RELATED WORK

Linux, among other software systems, has been intensively studied in the context of configurability and variability management. In the following, we want to list related work and similar studies.

Passos et al. [19] performed and extensive study extracting variability-coevolution patterns capturing changes in the variability model of the Linux kernel with subsequent changes to related artifacts (i.e., Makefiles and C source code). The authors created a catalog of (common) patterns describing how certain classes of changes affect different code artifacts, the frequency of such changes and how they are used in development.

Passos et al. [20] further investigated the scattering of driver features in Linux in a quantitative and qualitative study over 8 years of Linux development history (i.e., from Linux v2.6.12 to v3.19). The authors consider a feature to be scattered when it is not implemented in a modular way, but distributed over multiple source-code locations (e.g., multiple #ifdef blocks reference a specific feature or configuration option). Their results show that a majority of 82 percent of driver features can be introduced without causing scattering, and the amount of scattered features remains at a rather constant level of 20 percent. The authors conclude that scattering using pre-processor directives is a natural mechanism when dealing with system software of the size of Linux, although the usage yields potential maintenance efforts.

In [3], She et al. present a quantitative study of the Linux kernel, investigating certain properties of its variability model (e.g., characterization of features, code-granularity of features, and the model's hierarchy). The authors conclude that the variability model of the Linux kernel should be considered as a real-world benchmark for tool designers.

Nadi et al. [13] proposed an automated approach to (re)construct variability models from C code by extracting configuration constraints from the code base. Although the developed heuristics show a considerably high accuracy of 77 percent, the overall approach can recover only 19 percent of all model constraints. The authors further conclude that manual extraction of technical constraints is barely possible, even for experienced developers.

Among others [4, 9, 11], our group aims to support the aforementioned efforts of maintaining variable software by developing methods and tools to detect variability-related bugs, such as identifying dead #ifdef code [5], extending variability models with information from build systems [6] and developing a systematic approach to reveal configurability related build issues [14]. Our study aims to extend those bug-detection techniques to reliably analyze and locate variability bugs and their causes in git commits, which requires detailed information from Kconfig when feature models are being changed.

7. CONCLUSION

This paper was initially motivated by integrating fine-grained FM diffs into our tools. We require such diffs to determine the impact of changes applied to Kconfig files on the feature models in order to improve the analysis of dead features and dead #ifdef blocks. In course of the paper, we have shown that current feature models of Kconfig, as extracted by undertaker-kconfigdump, suffer the limitation that they do not reflect Kconfig semantically. In order to overcome this limitation and to make feature-model diffs applicable for our use case, any observed change affecting an architecture-specific FM must be validated by means of a SAT solver.

Acknowledgements

We want to thank Greg Kroah-Hartman for taking the time to discuss our issues and for sharing his knowledge and data publicly on github [18].

References

[1] Armin Biere. "PicoSAT essentials". In: *Journal on Satisfiability, Boolean Modeling and Computation* 4 (2008), pp. 75–97.

[2] Rafael Lotufo, Steven She, Thorsten Berger, Krzysztof Czarnecki, and Andrzej Wasowski. "Evolution of the Linux Kernel Variability Model". In: *Proceedings of the 14th International Conference on Software Product Lines: Going Beyond*. SPLC'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 136–150.

[3] Steven She, Rafael Lotufo, Thorsten Berger, Andrzej Wasowski, and Krzysztof Czarnecki. "The Variability Model of The Linux Kernel." In: *VaMoS* 10 (2010), pp. 45–51.

[4] Christian Kästner, Paolo G. Giarrusso, Tillmann Rendel, Sebastian Erdweg, Klaus Ostermann, and Thorsten Berger. "Variability-aware Parsing in the Presence of Lexical Macros and Conditional Compilation". In: *Proceedings of the 2011 ACM International Conference on Object Oriented Programming Systems Languages and Applications*. OOPSLA '11. Portland, Oregon, USA: ACM, 2011, pp. 805–824. ISBN: 978-1-4503-0940-0.

[5] Reinhard Tartler, Daniel Lohmann, Julio Sincero, and Wolfgang Schröder-Preikschat. "Feature Consistency in Compile-Time Configurable System Software". In: *Proceedings of the EuroSys 2011 Conference (EuroSys '11)*. Salzburg, 2011, pp. 47–60.

[6] Christian Dietrich, Reinhard Tartler, Wolfgang Schröder-Preikschat, and Daniel Lohmann. "A Robust Approach for Variability Extraction from the Linux Build System". In: *Proceedings of the 16th International Software Product Line Conference*. Ed. by ACM Press. Vol. 1. Salvador - Brazil, 2012, pp. 21–30.

[7] Christian Dietrich, Reinhard Tartler, Wolfgang Schröder-Preikschat, and Daniel Lohmann. "Understanding Linux Feature Distribution". In: *Proceedings of the 2012 Workshop on Modularity in Systems Software*. MISS '12. Potsdam, Germany: ACM, 2012, pp. 15–20.

[8] Michael Kerrisk. *Kernel build/boot testing*. 2012. URL: https://lwn.net/Articles/514278/.

[9] Sarah Nadi, Christian Dietrich, Reinhard Tartler, Ric Holt, and Daniel Lohmann. "Linux Variability Anomalies: What Causes Them and How Do They Get Fixed?" In: *Proceedings of the 10th Working Conference on Mining Software Repositories*. San Francisco, CA, USA, 2013, pp. 111–120.

[10] Iago Abal, Claus Brabrand, and Andrzej Wasowski. *40 Variability Bugs in the Linux Kernel*. Tech. rep. TR-2014-180. IT University of Copenhagen, 2014.

[11] Iago Abal, Claus Brabrand, and Andrzej Wasowski. "42 variability bugs in the linux kernel: a qualitative analysis". In: *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. ACM. 2014, pp. 421–432.

[12] Nicolas Dintzner, Arie Van Deursen, and Martin Pinzger. "Extracting feature model changes from the Linux kernel using FMDiff". In: *Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems*. ACM. 2014, p. 22.

[13] Sarah Nadi, Thorsten Berger, Christian Kästner, and Krzysztof Czarnecki. "Mining configuration constraints: Static analyses and empirical results". In: *Proceedings of the 36th International Conference on Software Engineering*. ACM. 2014, pp. 140–151.

[14] Reinhard Tartler, Christian Dietrich, Julio Sincero, Wolfgang Schröder-Preikschat, and Daniel Lohmann. "Static analysis of variability in system software: The 90,000# ifdefs issue". In: *Proc. USENIX Conf* (2014), pp. 421–432.

[15] Nicolas Dintzner, Arie van Deursen, and Martin Pinzger. "Analysing the Linux kernel feature model changes using FMDiff". In: *Software & Systems Modeling* (2015), pp. 1–22. ISSN: 1619-1366.

[16] Sascha El-Sharkawy, Adam Krafczyk, and Klaus Schmid. "Analysing the Kconfig Semantics and Its Analysis Tools". In: *Proceedings of the 2015 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*. GPCE 2015. Pittsburgh, PA, USA: ACM, 2015, pp. 45–54.

[17] C. Kästner. *kconfigreader*. 2015. URL: https://github.com/ckaestne/kconfigreader (visited on 12/15/2015).

[18] Greg Kroah-Hartman. *Kernel Development*. 2015. URL: https://github.com/gregkh/kernel-development (visited on 09/22/2015).

[19] Leonardo Passos, Leopoldo Teixeira, Nicolas Dintzner, Sven Apel, Andrzej Wąsowski, Krzysztof Czarnecki, Paulo Borba, and Jianmei Guo. "Coevolution of variability models and related software artifacts". In: *Empirical Software Engineering* (2015), pp. 1–50.

[20] Leonardo Passos, Jesus Padilla, Thorsten Berger, Sven Apel, Krzysztof Czarnecki, and Marco Tulio Valente. "Feature Scattering in the Large: A Longitudinal Study of Linux Kernel Device Drivers". In: *14th International Conference on Modularity (MODULARITY)*. 2015.