Information and Software Technology 78 (2016) 27–52

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Assessing fine-grained feature dependencies

Iran Rodrigues[a,*], Márcio Ribeiro[a], Flávio Medeiros[c], Paulo Borba[b], Baldoino Fonseca[a], Rohit Gheyi[c]

[a] Federal University of Alagoas, Maceió, Brazil
[b] Federal University of Pernambuco, Recife, Brazil
[c] Federal University of Campina Grande,Campina Grande, Brazil

ARTICLE INFO

Article history:
Received 9 August 2015
Revised 18 May 2016
Accepted 23 May 2016
Available online 24 May 2016

Keywords:
Preprocessor
Software family
Feature dependency

ABSTRACT

Context: Maintaining software families is not a trivial task. Developers commonly introduce bugs when they do not consider existing dependencies among features. When such implementations share program elements, such as variables and functions, inadvertently using these elements may result in bugs. In this context, previous work focuses only on the occurrence of intraprocedural dependencies, that is, when features share program elements within a function. But at the same time, we still lack studies investigating dependencies that transcend the boundaries of a function, since these cases might cause bugs as well.

Objective: This work assesses to what extent feature dependencies exist in actual software families, answering research questions regarding the occurrence of intraprocedural, global, and interprocedural dependencies and their characteristics.

Method: We perform an empirical study covering 40 software families of different domains and sizes. We use a variability-aware parser to analyze families source code while retaining all variability information.

Results: Intraprocedural and interprocedural feature dependencies are common in the families we analyze: more than half of functions with preprocessor directives have intraprocedural dependencies, while over a quarter of all functions have interprocedural dependencies. The median depth of interprocedural dependencies is 9.

Conclusion: Given these dependencies are rather common, there is a need for tools and techniques to raise developers awareness in order to minimize or avoid problems when maintaining code in the presence of such dependencies. Problems regarding interprocedural dependencies with high depths might be harder to detect and fix.

1. Introduction

Developers commonly introduce errors when they fail to recognize dependencies among the software modules they are maintaining [1]. The same situation happens in configurable systems in terms of program families and product lines, where features share program elements such as variables and functions. This way, features might depend on each other and developers can miss such dependencies as well. Consequently, by maintaining one feature implementation, they might introduce problems to another, like when assigning a new value to a variable which is correct to the feature under maintenance, but incorrect to the one that uses this variable [2,3].

In this context, developers often use the C preprocessor to implement variability in software families [4–7]. The C preprocessor allows the use of directives to annotate the code, associating program elements with specific features. When a developer defines a variable in a feature and then uses it in another feature, we have a feature dependency. The same happens with functions.

Previous work [8] reports on how often feature dependencies occur in practice by considering 43 preprocessor-based families and product lines. However, the study focuses only on intraprocedural dependencies, that is, feature dependencies that occur exclusively within the function boundaries. Nevertheless, dependencies that go beyond function boundaries might be harder to detect.

* Corresponding author.
E-mail addresses: irgj@ic.ufal.br, iran@iranrodrigues.com.br (I. Rodrigues), marcio@ic.ufal.br (M. Ribeiro), flaviomedeiros@copin.ufcg.edu.br (F. Medeiros), phmb@cin.ufpe.br (P. Borba), baldoino@ic.ufal.br (B. Fonseca), rohit@dsc.ufcg.edu.br (R. Gheyi).

http://dx.doi.org/10.1016/j.infsof.2016.05.006
0950-5849/

Despite important, we still lack a study that takes other kinds of feature dependencies into account.

Therefore, to minimize these lack and better understand feature dependencies, in this work we perform an empirical study to assess to what extent feature dependencies occur in practice, identifying their characteristics and frequency. We also compare some of our results with results from previous work [8].

Before executing this study, as a first step, we arbitrarily analyze several bug reports from many open-source software families, like *GCC*,[1] *GNOME*,[2] and *Linux* kernel.[3] The idea of this first step is to learn how configuration-related bugs happen in such families and better prepare our study. After finding examples of bugs related to feature dependencies, we conduct an empirical study that complements previous work on this topic, in the sense that we take interprocedural dependencies into account. Notice that, during maintenance of preprocessor-based software, these dependencies are even harder to detect: one feature might use data from another and they are in different functions. Because in a typical system we have several method calls passing data, we also compute the depth of such dependencies (from the variable definition to its use). In addition, we consider dependencies based on global variables. We also compute the dependency direction, that is, mandatory-to-optional, optional-to-mandatory, and optional-to-optional. A mandatory-to-optional dependency, for instance, means that the definition of the program element (for instance, a global variable) happens in a mandatory feature—that is, no `#ifdef` encompassing the definition—and its use in an optional feature. In particular, we answer the following research questions: How often do program families contain intraprocedural dependencies? How often do program families contain global dependencies? How often do program families contain interprocedural dependencies? How often do dependencies of different directions occur in practice? What is the dependency depth distribution for interprocedural dependencies? How the results of the current study compare with the previous ones? Answering these questions is important to better understand feature dependencies and assess their occurrence in practice.

To answer our research questions, our study covers 40 C program families of different domains and sizes. We select these families inspired by previous work [6–11]. We rely on *TypeChef* [12], a variability-aware parser, to compute feature dependencies considering the entire configuration space of each source file of the families we analyze. To detect dependencies that span multiple files, we perform global analysis (instead of per-file analysis).

The data we collect in our empirical study reveal that the feature dependencies we consider in this work are reasonably common in practice, except the ones regarding global variables. Following the convention "average ± standard deviation", our results show that 51.44% ± 17.77% of functions with preprocessor directives have intraprocedural dependencies, 11.90% ± 12.20% of the functions which use global variables have global dependencies, while 25.98% ± 19.99% of all functions have interprocedural dependencies.

In summary, the main contributions of this paper are:

- data on feature dependency that reveal to what extent they are common in practice, complementing previous work by considering new types of dependencies;
- a strategy to compute feature dependencies based on the TypeChef variability parser.

We organize the remainder of this paper as follows. In Section 2 we introduce the concept of feature dependency. Next, in Section 3 we show motivating examples that illustrate variability bugs from industrial systems. Then, we present the empirical study settings in Section 4. After, in Section 5 we present and discuss the results. Later, in Section 6 we present some consequences of our work. In Section 7 we discuss the related work and in Section 8 we present the final considerations of this work.

This paper is an extension of our previous work [8] on feature dependency analysis. In this work we bring more evidence regarding bugs related to intraprocedural feature dependencies. Compared to the previous study, we analyze new families and use a different tool, improving its external validity. Moreover, we now take global and interprocedural dependencies into account, presenting bugs related to such types of dependencies and computing data regarding their presence in a set of industrial software families.

```
1.  #ifdef A
2.    int x;
3.  #endif
4.  …
5.  #ifdef B
6.    x++;
7.  #endif
```

Fig. 1. Example of a feature dependency regarding variable `x`.

2. Feature dependency

A program family consists of a set of programs that share a common core but also have distinguishing functionalities. These commonalities and variabilities are often modeled as features, each representing increments in functionality to the program. Each feature provides a potential configuration option, so developers can generate different programs tailored for specific tasks or platforms. When we consider program families written in C, developers often use the C preprocessor (*cpp*) to implement variability in those systems [4–7].

The C preprocessor allows the use of conditional compilation directives such as `#if` or `#ifdef` along with a *macro expression* to surround feature-specific fragments of code. Macro expressions might contain one or more macros as a boolean formula, as in `#if defined(A) && defined(B)`, which might refer to specific configuration options. The minimum subset of features in which a fragment of code is included in the conditional compilation is called *presence condition* [13]. Developers can use preprocessor directives to wrap from entire structures such as functions to part of a statement such as a single variable, allowing variability in different levels of granularity. This flexibility also allows code from a single feature to be scattered all over the program.

Often features communicate and collaborate with each other, so their implementations might share program elements and data. When different features refer to the same program element, such as a variable, we have a feature dependency. Following the classification proposed by Apel et al. [14], such feature dependencies we consider in this paper are operational feature interactions, since a feature pass data to another one.

To better explain this concept, we refer to the code snippet in Fig. 1. In the figure, the definition of variable `x` (see line 2) is inside an `#ifdef` block, associated to the macro expression `A` (see line 1). In practice, not all macros in a macro expression correspond to actual features in a broader sense. However, since feature models are not always available, and for the sake of simplicity, in this work we consider that each macro in a macro expression refers to a different feature. That said, we consider that the definition of `x` is in a *code fragment* of feature `A`. Likewise, `x` is later incremented (see line 6) in a *code fragment* of feature `B` (see line 5). This means that the definition of `x` will be included in the compilation if and only

[1] https://gcc.gnu.org/bugzilla/.
[2] https://bugzilla.gnome.org/.
[3] https://bugzilla.kernel.org/.