Modularity Metrics for Conditional Compilation Software Product Lines

Waldemar Pires Ferreira Neto and Sérgio Soares
Informatics Center, Federal University of Pernambuco, Recife, Brasil 50670-901
Email: {wpfn, scbs}@cin.ufpe.br

Abstract—Software product lines (SPLs) enable modular large-scale reuse through a software architecture addressing features commonalities and variabilities. Several works used metrics to measure and evaluate SPLs. Many metrics have been proposed and adapted leading to different degrees of success in many SPL techniques. In this position paper, we propose a set of metrics to measure the conditional code modularity of SPL based on conditional compilation. Moreover, we present some examples of their application and how those metrics can be combined together.

I. INTRODUCTION

A software product line (SPL) is a set of related software products that are generated from reusable assets. They share a common core of features (commonalities) and differ by the variations each product has that are implemented by these reusable assets [1]. There are many techniques to promote the asset reuse, one the most remarkable is the Conditional Compilation (CC) [2]. This technique enables control over the code segments to be included or excluded from a program compilation being responsible to manage the SPL variability. Directives mark the varying locations in the code stating which part of the code is associated to certain tags. In a SPL, these tags are associated with features, by selecting features of a product, the associated tags are enable to generate the code of the product in question.

Any SPL has many features, and it is impossible to develop a large SPL, where each SPL’s asset is responsible for only one specific feature. Usually, some features are spread over many asset. In this context, the feature modularity is a key weapon in reducing with complexity in a large SPL.

In spite of all the benefits of Conditional Compilation in SPLs [2], this variability management technique suffers from lack of legibility and difficulty to determine the scope of each conditional compilation. In this sense, ensuring modularity mitigates these limitations. Therefore, if each SPL feature has a modular definition, we can easily identify the scope of it.

Many real life SPLs adopt classic code metrics to ensure modularity [1]. However, the problem with some classic metrics is isolation of assets, ignoring the relationship among them. Some works [3], [4] took into account the relationship among assets, however these works focus on other techniques, and cannot be fully applicable to Conditional Compilation.

Despite CC is not the best way to modularize features, this technique is used very often [2]. In the light of this fact, in this work we adapted some modularity metrics to the context of Conditional Compilation in SPLs. Thus these addition metrics can help developers to find features that need to be better modularized. The metrics presented in this work are adaptation of the metrics adopted by Figueiredo et at. [3].

II. METRICS FOR SOFTWARE PRODUCT LINES

In the previous section we presented a general definition of SPL based on conditional compilation (SPL-CC). More precisely in this work a SPL-CC have to be composed of three parts: feature model, a tree model that describes the available features; configuration knowledge (CK), a map that identifies for each feature, its corresponding assets; Assets, all the artifacts (eg. classes, aspects, libraries, etc.) that implement the behavior of a feature in the final product. Figure 1 presents an example of an SPL with a feature to publish comments. In SPL-CC some code snippets at an asset can be changed in compilation time, in order to implement a feature (Figure 2).

Fig. 1. Software Product Line Example.

Since the foundations of SPL, metrics has been used to measure and evaluate SPLs [1], Junior et al. [5] argue about the usage of some code metrics to assert the complexity and efficiency of the SPLs. Moreover, the authors propose and adapt several other metrics to measure the separation of concerns in the SPL context. The metrics used by the authors were tailored to SPL based on components. Figueiredo et al. [3] evaluate several characteristics of a specific SPL based on a set of metrics. The authors proposed 3 metrics to evaluate SPL modularity, theses metrics are focused on SPL based on aspects. SantAnna et at. [4] conducts a depth discussion about metrics to evaluate SPLs based on aspects. Even the Return of
Investment (ROI) for SPL based on metrics has been proposed [6]. In fact, an SPL-CC can have some features implemented via components or aspects, however its core characteristic are the conditional code snippets, for short only code snippets. In this work we distinguished a code snippets from a template. A template is an asset that can have some code snippet in it. For instance, the asset Publication.java (Fig. 2) is a template, and it has 3 distinct code snippets.

In order to measure the conditional code snippet modularity of an SPL-CC, we present 3 metrics. In the following we detail each of them.

**Features Diffusion over Asset (FDoA):** This metric counts, for each feature, the number of assets that have a conditional code snippets corresponding to the feature. A specialization of this metric can either identify the set of assets. Considering the SPL-CC in Figure 1, the FDoA of feature “twitter” is 1 asset and the FDoA set is \{Publication.java\}. This metric was adapted from Concern Diffusion over Components (CDC) [3]. If we applied CDC under the feature “twitter”, it would be is 2 assets and the FDC set would be \{Publication.java, TwitterTool.java\}.

**Features Diffusion over Conditional Code (FDoCC):** similar to previous metric, this metric counts, for each feature, the number of conditional code snippets spread over all assets. Considering the SPL-CC in Figure 1, the FDoCC to the feature “twitter” is 2 conditional code snippets (corresponding to the code snippet “#if Twitter ” and “#if( Twitter & Facebook ” at the asset Publication.java). This metric was adapted from Concern Diffusion over Operations (CDO) [3]. Since, we focus on conditional compilation modularity, the conditional code snippets has more meaning that the related operations.

**Features Diffusion over Lines of Code (FDoLoC):** this metric sums the number of line of code at all conditional code snippets required to implement a specific feature. Comments and black lines are ignored. Beside, even if an asset is identified as mandatory at the CK for a specific feature, only the code at the conditional code snippets is included (if it exists). Considering the SPL-CC in Figure 1, the FDoLoC to the feature “twitter” is 3 lines of code. This metric was adapted from Concern Diffusion over Lines of Code (CDLOC) [3]. If we applied CDLOC under the feature “twitter”, it would include all the lines of code in Publication.java and TwitterTool.java.

The metrics presented before can aim to measure the modularity of SPL-CC. Exploiting the combination of those metrics, more useful information can be extracted from the SPL-CCs. For instance:

**FDoA vs FDoCC:** a low FDoA and a high FDoCC can indicate that the implementation of the feature is scattered over the asset. Then, maybe the conditional code snippets (or part of them) can be implemented in an aspect, becoming the SPL-CC more modular.

**FDoCC vs FDoLoC:** a low FDoCC and a high FDoLoC can indicate that the conditional code snippets, for the specific feature, is too complexity. Then, some refactoring can be required.

### III. Conclusion

In this work we presented three new metrics to measure the conditional code snippet modularity of an SPL-CC. Besides, we presented some examples of their application and how those metrics can be combined in order to provide more information about the current SPL-CC status.

As future work, we plan to apply the proposed metric in a real SPL-CC, in order to evaluate the soundness of our metric. Besides, we planed to implement a tool to automatically collect this metric based on SPL-CC artifacts. If the SPL-CC does not clearly distinct the feature model, CK, and assets, the tool has to abstract them.

### Acknowledgment

The Brazilian National Council for Scientific and Technological Development - CNPq (Project. 141.066/2013-0) supported this project.

### References


