Bottom-Up Technologies for Reuse: Automated Extractive Adoption of Software Product Lines

Jabier Martinez, Tewfik Ziadi, Tegawendé Bissyandé, Jacques Klein, Yves Le Traon

Bottom-Up Technologies for Reuse: Automated Extractive Adoption of Software Product Lines

Jabier Martinez, Tewfik Ziadi
LIP6, Sorbonne Universités, UPMC Univ Paris 06, CNRS
Paris, France
(jabier.martinez tewfik.ziadi)@lip6.fr

Tegawendé F. Bissyandé, Jacques Klein, Yves Le Traon
SnT, University of Luxembourg
Luxembourg
(tegawende.bissyande jacques.klein yves.leatraon)@uni.lu

***Abstract*—Adopting Software Product Line (SPL) engineering principles demands a high up-front investment. Bottom-Up Technologies for Reuse (BUT4Reuse) is a generic and extensible tool aimed to leverage existing similar software products in order to help in extractive SPL adoption. The envisioned users are 1) SPL adopters and 2) Integrators of techniques and algorithms to provide automation in SPL adoption activities. We present the methodology it implies for both types of users and we present the validation studies that were already conducted. BUT4Reuse tool and source code are publicly available under the EPL license.**
Website: http://but4reuse.github.io
Video: https://www.youtube.com/watch?v=pa62Yc9LWyk

***Index Terms*—Software product line engineering; Extractive software product line adoption; Variability management; Reverse engineering**

I. Introduction

Software Product Line (SPL) has matured in recent years and is now a popular paradigm for variability management in software engineering. Thanks to SPL Engineering (SPLE), a company can create a family of related product configurations for a given domain and later automatically generate the associate product variants based on reusable assets. Among other benefits, SPLE helps to achieve large scale productivity gains, address time to market requirements and improve product quality [1].

It has been reported that more than 50% of industrial practitioners formally implement an SPL only after the instantiation of several similar product variants using ad-hoc reuse techniques [2] such as copy-paste-modify. These extractive processes are referred to as *bottom-up* or *extractive* approaches to implement systematic software reuse. In an extractive SPL adoption, an organization capitalizes on existing custom software systems by extracting the common and varying source code into a single SPL [3].

Although some tools aim to enhance clone-and-own reuse practices [4], complete SPL adoption remains challenging because of the lack of comprehensive and usable tools. We present BUT4Reuse, a tool-supported framework mainly dedicated to automate relevant tasks for extractive SPL adoption.

*BUT4Reuse is a tool to be used by companies with existing similar software products that aim to get the benefits claimed by SPL Engineering. We will refer to this first type of users as **SPL adopters**.*

BUT4Reuse is designed to be generic and extensible [5]. *Generic* by enabling its use in different scenarios with product variants of different software artefact types (e.g., source code in Java, C, models, requirements or plugin-based architectures), and *extensible* by allowing to add different concrete techniques or algorithms for the relevant activities of extractive SPL adoption (i.e., feature identification, feature location, mining feature constraints, extraction of reusable assets, feature model synthesis and visualisations). Several validation studies using BUT4Reuse for different software artefact types or with different extensions have already been published [5]–[11].

*BUT4Reuse is a tool to be used by **Integrators** to test concrete techniques for the different steps of extractive SPL adoption. The software engineering research community can integrate innovative techniques for comparison and benchmarking.*

This paper is structured as follows: Section II describes the methodology it implies for SPL adopters and Section III describes technical aspects for integrators. Section IV presents validation studies and Section V concludes this paper.

II. BUT4Reuse for SPL adopters

Figure 1 illustrates the methodology for SPL adopters. We present details of the numbered steps in the Figure.

1. Preparation: The use of BUT4Reuse assumes the existence of product variants (e.g., legacy artefacts created through ad-hoc reuse). SPL adopters *collect the artefact variants* and list them in a BUT4Reuse *artefact model*. This can be achieved by simply drag and drop the variants as shown in Figure 2. After this, SPL adopters, verify that BUT4Reuse supports the artefact types that they are targeting. In practice, that means that an *adapter* is available for this artefact type. 15 different adapters are currently supported [12] covering a wide range of artefact types. Considering that we selected a set of vending machine variants for analysis [7], Figure 3 shows that support for EMF models is provided [8] and this adapter is preselected among the available adapters. At this point, the SPL adopters can use the techniques, algorithms and visualisations selected by default, or they can change or configure them in the BUT4Reuse preferences.

2. Feature identification and location: In SPLE, a feature is defined as a prominent or distinctive characteristic, quality or user-visible aspect of a software system or systems [13].

Given the complexity of products, BUT4Reuse does not assume a complete upfront knowledge of the existing features throughout the artefact variants. Several domains of expertise are often required to build a product and different stakeholders are responsible for different functionalities. In this context, domain knowledge about the features of legacy variants can be scattered across the organization. If this is the case, *Feature identification* should be performed.

In feature identification SPL adopters aims to have an explicit list of the features within the scope of their input variants. This requires to analyse the automatically identified implementation blocks, manipulate them and give them a feature name. Figure 4 shows an example of identified implementation blocks and their distribution in a set of variants [7]. In Figure 5 we show support for feature naming using information retrieval techniques [6]. We can also visualise the elements of a set of variants and their distribution among the implementation blocks [8] as shown in Figure 6. On the contrary, if the features are known in advance, and their presence (and absence) in the artefact variants is known, *Feature location* can be performed directly after creating a *Feature list*. This Feature list model is an exhaustive list of the features with the references to the variants that have/implement each feature.

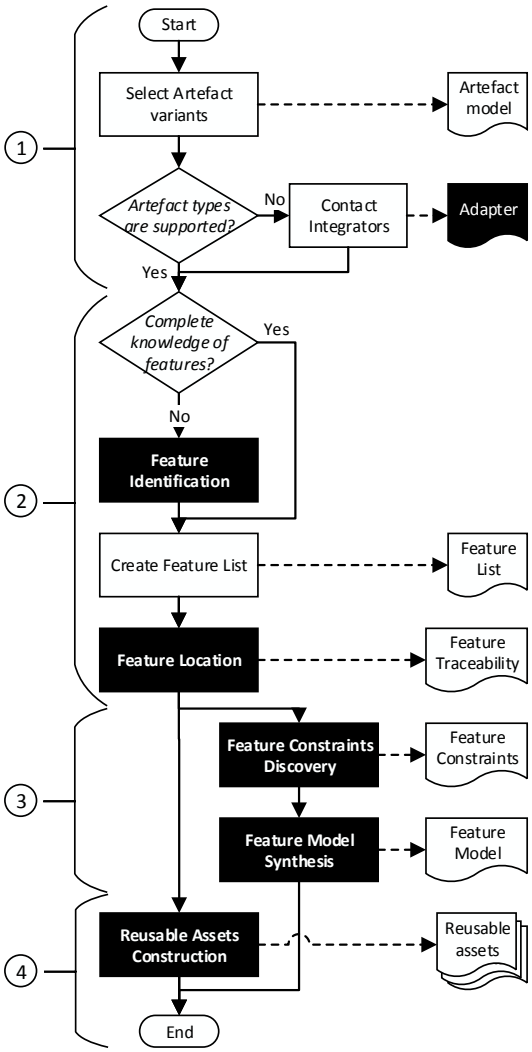

Fig. 1: Methodology for SPL adopters. Integrators can extend BUT4Reuse at the points with black background.

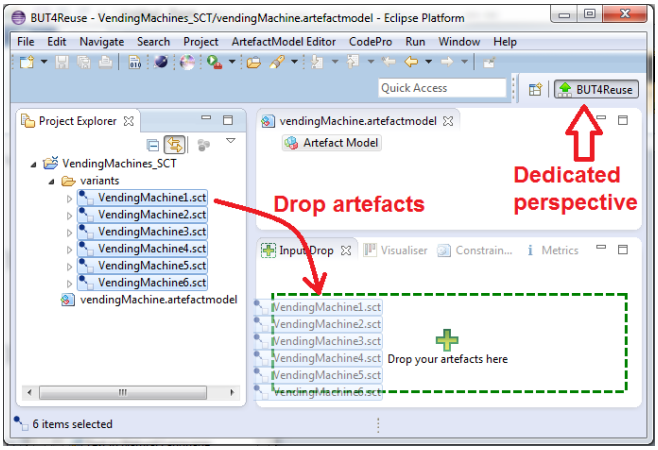

Fig. 2: Drag and drop artefact variants for their analysis.

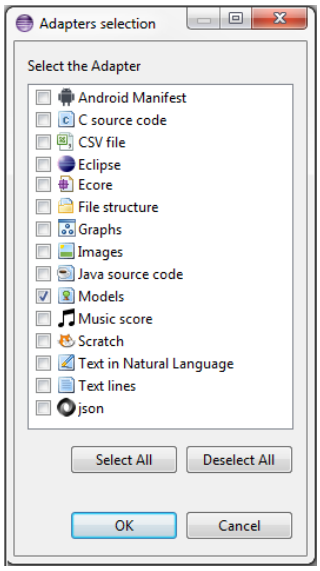

Fig. 3: Adapters selection for an artefact model.

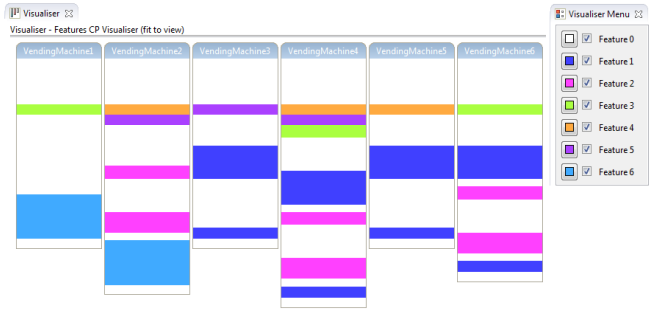

Fig. 4: Bars and stripes to visualise how a set of features are identified and located in a set of vending machines [7].

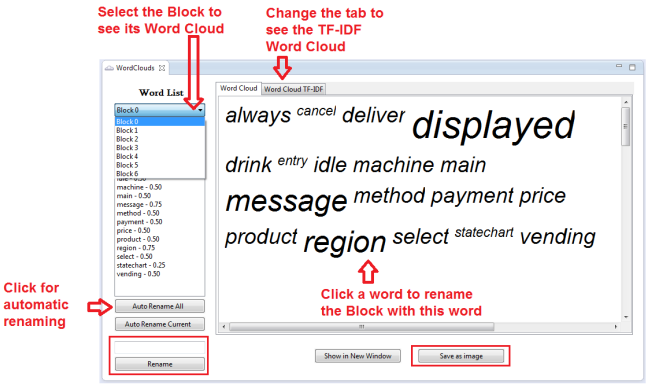

Fig. 5: Word clouds show relevant names for each implementation block during feature identification [6].

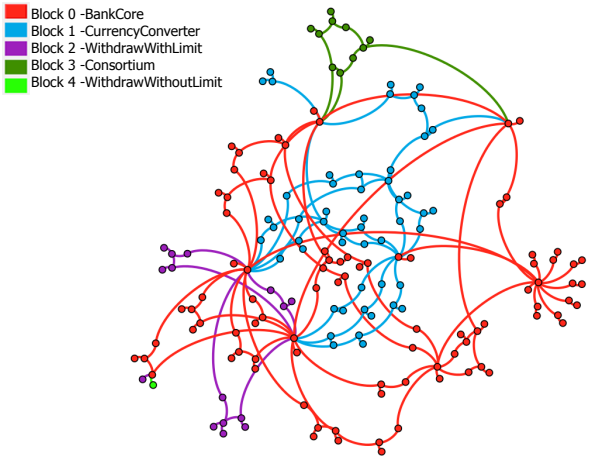

Fig. 6: Graph visualisation of all the elements of the Bank systems case study, their dependencies (clockwise-based directed graph) and their relation to the identified blocks [8].

3. Feature model creation: Constraints between features (e.g., one feature requires another feature, or one feature excludes another feature) are important domain knowledge that will guarantee the correctness of the product configurations in the SPL. The variants used in the extractive SPL process have correct feature configurations but, apart from these configurations, the derivation of new feature combinations will be desired. Because of this, *feature constraints discovery* (also known as mining feature constraints) is an important step. Normally this is performed after feature location because the implementation of the feature can provide evidence of structural dependencies between the features. However, other approaches could make use directly of the feature list, without the feature location, trying to infer constraints by for example mining rules from the feature configurations of the existing artefacts. Once the constraints discovery process is finished, the *feature model synthesis* process creates a structured *feature model* taking into account the constraints information and, optionally, other available information from the domain.

4. Reusable assets extraction: Once feature location was performed, the traceability between the features and their implementation has been calculated. At this point, the reusable assets construction aims to extract the implementation of each feature and prepare it to be used by an SPL compositional or annotative approach. The feature model and the reusable assets will make it possible to have an initial operative SPL.

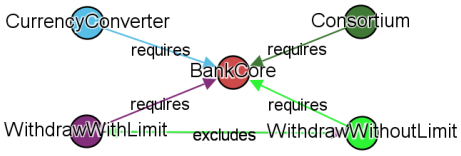

Fig. 7: Graph visualisation of the discovered constraints in the Bank systems case study [8].

III. BUT4Reuse for Integrators

BUT4Reuse is built on Eclipse following its plugin-based architecture. In the current open sourced version, it consists of 47 plugins apart from the plugins dedicated to testing. Integrators can extend BUT4Reuse by providing adapters and implementing techniques for extractive SPL adoption activities. In Figure 1, we noted with black background the main interesting extensible parts for Integrators. In practice, BUT4Reuse follows the extension points mechanism provided by plugin development in Eclipse. For each extension point, integrators only need to declare the extension and to implement a predefined interface provided by BUT4Reuse.

The Adapters and the Elements are the interfaces that are at the core of the BUT4Reuse framework. In fact, the principles of BUT4Reuse are 1) A software artefact can be decomposed into distinct Elements. 2) Given a pair of Elements in a specific artefact type, a similarity metric can be computed for comparison purposes. 3) Given a set of Elements recovered from existing artefacts, a new artefact, or at least a part of it (which would be a reusable asset), can be constructed [5].

Based on our reported experiences [5], the design of an Adapter uses to take more time than its actual implementation. For the design, integrators need to define the elements that compose the artefacts, their granularity, their dependencies with other elements and how to calculate their similarity. Figure 8 illustrates some examples of adaptation and construction but complete details are available [5], [12].

IV. Validation Studies

This section presents works using BUT4Reuse. In particular we focus on adapter, visualisation and benchmark studies.

Adapter studies. *Source code:* We reproduced several case studies of feature identification [5] presented in previous works dealing with variants of source code in Java and C [14]. We used an extended version of the Java BUT4Reuse adapter to analyze several families of Android apps. The analysis served to identify app variants belonging to feature-oriented Android app generators, and to discuss about cases of device-driven and content-driven variability [11].

Models: We used BUT4Reuse for the identification of variability and commonality in model variants, as well as the extraction of a CVL-compliant Model-based Software Product Line (MSPL) from the features identified on these variants [8].

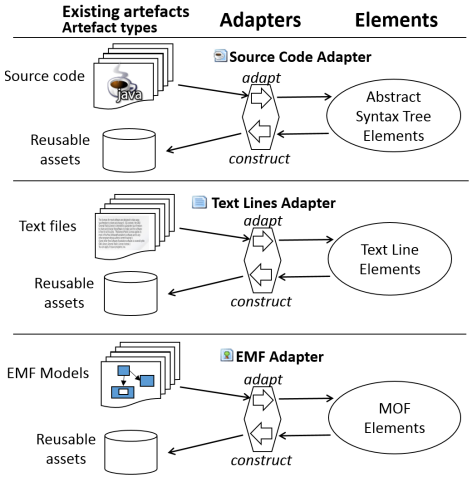

Fig. 8: Artefact types examples and Elements representation creation through the adapters [5].

The models adapter is suitable to any MOF-based model. We evaluated with large case studies to show how our feature identification with structural constraints discovery and the SPL adoption process are implemented to make the approach valid (i.e., the extracted software product line can be used to regenerate all variants considered) and sound (i.e., derived variants which did not exist are at least structurally valid).

Plugin-based architectures: We considered the SPL adoption of Eclipse variants which are plugin-based architectures targeting different development scenarios [5]. We discussed the learning curve and the lessons learnt from a group of master students in the design and implementation of the Eclipse adapter. Then, we used the adapter for analyzing the variants of an Eclipse release to discuss the results of the different phases of extractive SPL adoption in this case study.

Visualisation studies. *Bars and stripes:* We showed how bars and stripes can be used as visualisation paradigm for the identification of commonality and variability as it was already have been shown appropriate for clone detection [7]. We showed an example of this visualisation in Figure 4.

WordClouds: Word cloud visualisations can provide insights of the emerging vocabulary and variability from a set of variants. We designed VariClouds for helping domain experts in feature identification and naming [6]. We evaluated our approach by assessing its added-value to several previous works in the literature where no tool support was provided to domain experts to characterise features from software blocks. In these case studies we dealt with different artefact types to show its soundness and genericness. Figure 5 showed an example of this visualisation.

Feature Relations Graphs (FRoGs): FRoGs is an interactive visualisation paradigm to help domain experts and stakeholders to manage the challenges in maintaining the constraints among features [9]. The objective is to obtain a better understanding of feature constraints and potentially refine the existing feature model by uncovering and formalizing missing constraints (i.e., feature constraints discovery).

Graphs and Pruned Concept Hierarchy: Graphs are available, for example to show the relation of the elements of each variant, their assignation to blocks or the constraints between the blocks [8]. Figure 6 and 7 showed two graph examples. A special case of graphs is the Pruned concept hierarchy, also known as AOC-poset, also used in variants analysis.

Benchmark. Realistic, non-trivial, comparable, and reproducible settings are needed to compare techniques for extractive SPL adoption. In order to support research on feature location, the Eclipse Feature Location Benchmark (EFLBench) [10] was implemented and integrated in BUT4Reuse. The benchmark is based on the Eclipse releases and we provided example results of four different feature location approaches.

V. Conclusion

Extractive SPL adoption is a challenging task for SPL adopters. In addition, the research community on this field needs a common framework to test and compare their works on the different steps of the process. BUT4Reuse is a generic and extensible open source framework that aims to face these challenges and that already has a set of validation studies.

Acknowledgment

This work has been partially supported by the European project ITEA 3 15010 REVaMP^2.

References

[1] L. M. Northrop, P. C. Clements *et al.*, "A Framework for Software Product Line Practice, Version 5.0," www.sei.cmu.edu/productlines/framework.html, 2009.

[2] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. Wasowski, "A survey of variability modeling in industrial practice," in *VaMoS*, 2013.

[3] C. W. Krueger, "Easing the transition to software mass customization," in *PFE*, 2001.

[4] S. Fischer, L. Linsbauer, R. E. Lopez-Herrejon, and A. Egyed, "The ECCO tool: Extraction and composition for clone-and-own," in *ICSE, Volume 2*, 2015.

[5] J. Martinez, T. Ziadi, T. F. Bissyandé, J. Klein, and Y. L. Traon, "Bottom-up adoption of software product lines: a generic and extensible approach," in *SPLC*, 2015.

[6] ——, "Name Suggestions during Feature Identification: The VariClouds Approach," in *SPLC*, 2016.

[7] J. Martinez, T. Ziadi, J. Klein, and Y. L. Traon, "Identifying and visualising commonality and variability in model variants," in *ECMFA*, 2014.

[8] J. Martinez, T. Ziadi, T. F. Bissyandé, J. Klein, and Y. L. Traon, "Automating the extraction of model-based software product lines from model variants (T)," in *ASE*, 2015.

[9] J. Martinez, T. Ziadi, R. Mazo, T. F. Bissyandé, J. Klein, and Y. L. Traon, "Feature relations graphs: A visualisation paradigm for feature constraints in software product lines," in *VISSOFT*, 2014.

[10] J. Martinez, T. Ziadi, M. Papadakis, T. F. Bissyandé, J. Klein, and Y. L. Traon, "Feature location benchmark for software families using eclipse community releases," in *ICSR*, 2016.

[11] L. Li, J. Martinez, T. Ziadi, T. F. Bissyandé, J. Klein, and Y. L. Traon, "Mining Families of Android Applications for Extractive SPL Adoption," in *SPLC*, 2016.

[12] J. Martinez, "Mining software artefact variants for product line migration and analysis," Ph.D. dissertation, University of Luxembourg, 10 2016.

[13] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (foda) feasibility study," Carnegie-Mellon University Soft. Eng. Institute, Tech. Rep., 1990.

[14] T. Ziadi, C. Henard, M. Papadakis, M. Ziane, and Y. L. Traon, "Towards a language-independent approach for reverse-engineering of software product lines," in *SAC*, 2014.