Insights on Software Product Line Extraction Processes: ArgoUML to ArgoUML-SPL Revisited

Jabier Martinez
Tecnalia, Basque Research and Technology Alliance (BRTA)
Derio, Spain
jabier.martinez@tecnalia.com

Daniele Wolfart
PPGComp, Western Paraná State University (UNIOESTE)
Cascavel, Brazil
danielewolfart@gmail.com

Wesley K. G. Assunção
COTSI, Federal University of Technology - Paraná (UTFPR)
Toledo, Brazil
wesleyk@utfpr.edu.br

Eduardo Figueiredo
DCC, Federal University of Minas Gerais (UFMG)
Belo Horizonte, Brazil
figueiredo@dcc.ufmg.br

ABSTRACT

Software Product Lines (SPLs) are rarely developed from scratch. Commonly, they emerge from monolithic architectures when there is a need to create tailored variants, or from existing variants created in an ad-hoc way once their separated maintenance and evolution become challenging. Despite the vast literature about re-engineering systems into SPLs and related technical approaches, there is a lack of detailed analysis about the process itself and the effort that is involved. We provide and analyze empirical data of an existing SPL extraction process: the ArgoUML monolithic architecture transition to ArgoUML-SPL. The analysis relies on information mined from the version control history of the source-code repository and the discussion with developers that took part in the process. The contribution of this study is an in-depth characterization of the process compared to previous works that focused only on the structural results of the final SPL. We made publicly available the dataset and the analysis scripts to be used as baseline for extractive SPL adoption research and practice.

CCS CONCEPTS

• **Software and its engineering** → **Software reverse engineering**; **Software product lines**.

KEYWORDS

Software Product Line Architecture, Re-engineering, Mining Software Repositories, ArgoUML

ACM Reference Format:
Jabier Martinez, Daniele Wolfart, Wesley K. G. Assunção, and Eduardo Figueiredo. 2020. Insights on Software Product Line Extraction Processes: ArgoUML to ArgoUML-SPL Revisited. In *24th ACM International Systems and Software Product Line Conference (SPLC '20), October 19–23, 2020, MONTREAL, QC, Canada.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3382025.3414971

1 INTRODUCTION

Software Product Lines (SPL) [34] are not always developed from scratch but emerge from an existing system or from existing variants [7]. This kind of adoption is known as extractive adoption [26], which is undertaken by a re-engineering process. These re-engineering processes to extract SPLs have been analysed from different perspectives: industrial cases presented lessons learned regarding organizational and technical aspects [30], researchers are trying to increase the level of automation of the re-engineering steps [4], and practitioners have risk evaluation methods and cost models to measure the return on investment [2].

In this work, we focus on the process and the effort required for extractive SPL adoption and we fill the gap on the lack of comparable and reproducible research on the topic given the confidentiality constraints of existing industrial cases reported in the literature [30]. We can argue that there is still no evidence about the cost-effectiveness of using automatic approaches for the different re-engineering activities, and that ways to estimate the re-engineering effort beforehand could be improved. We consider that the main reason is because we lack baselines for comparison and empirical data providing clues for estimation.

As in other software engineering research fields, Open-Source Software provides valuable assets and information for extractive SPL adoption research [30, 39]. ArgoUML[1] is probably the most representative system that can be comparable to industrial practices [4, 30]. The transition from ArgoUML to ArgoUML-SPL was an academic project, but it was focused on industry-strength practices [8, 9]. The system was mainly selected because of its relevant size and the cross-cutting features it contained. ArgoUML-SPL was extensively used for validating automatic or semi-automatic techniques for feature location in families of systems [1, 4, 11–14, 28, 30–32, 37, 42], extracting SPLs or compositing new product variants [15, 23, 43], and its artifacts were source of information for reverse engineering studies [29, 33]. However, among these studies there is no translation from these results to actual human effort on the re-engineering process. In the literature, we can find some pieces of work describing the effort related to the duration [3, 10, 25, 40, 41] and number of developers [16, 35, 36] to conduct the re-engineering process. However, only brief or no discussions are provided.

ArgoUML is mostly used in SPL research considering the result of the re-engineering, not the actual process of obtaining an SPL. In the seminal publication where ArgoUML-SPL was presented [9],

[1]http://argouml.tigris.org , SPL http://argouml-spl.tigris.org. Moved (July 2020): https://github.com/argouml-tigris-org, https://github.com/argouml-tigris-org/argouml-spl

the focus was on an in-depth characterization of the extracted SPL in terms of size metrics, crosscutting metrics, crosscutting behavior analysis, granularity metrics and analysis about where in the code structure the annotations were inserted with more frequency. Contrary to the characterization of *the structure* of the extracted SPL, in this work we contribute with a missing analysis and characterization on *the process* of creating ArgoUML-SPL from the ArgoUML monolithic architecture.

The contributions of this work are:

- A characterization of the process where ArgoUML was transformed into ArgoUML-SPL using practices from the mining software repositories field [17]. The insights of this characterization can be a source of reference for companies and practitioners willing to re-engineer an existing monolithic system into an SPL. Practitioners can then be familiar with a re-engineering process and how it was conducted.
- A publicly available dataset and analysis scripts[2] to be used as baseline for extractive SPL adoption research, and as an illustrative and comprehensible example for industry.

2 BACKGROUND

Annotative SPLs: SPLs can be implemented with many different compositional or annotative approaches [19]. The annotative approach used to implement ArgoUML-SPL is a well-known technique for handling software variability [6]. It has long been used in programming languages like C (commonly known as `ifdef` preprocessor directives), and it can also be used in object-oriented languages, such as C++ [18] and Java (e.g., the Java Preprocessor[3] functionality used in ArgoUML-SPL). In this approach, annotative directives indicate pieces of code that should be compiled/included or not based on the value of variables. The pieces of code can be marked at granularity of a single line of code or to a whole file.

The Re-engineering Process: The process of extracting an SPL from legacy system variants is covered in a mapping study [4]. This process covers some specific phases that are designed to deal with commonalities and variabilities among the system variants. The main motivation of migrating to an SPL is to mitigate engineering problems related to maintenance and evolution. Another scenario is the migration of a single monolithic system to an SPL. In such situation, the motivation is to allow different configurations of products from a common base of artifacts, enabling systematic reuse. Through an extractive adoption of an SPL [26], a company can reduce time-to-market for new products, growing its portfolio of software products, and better fulfilling client demands. The extraction should be conducted with the support of the available artifacts. Some studies highlight the importance of high-level artifacts, such as feature models and product line architectures [5, 24]. Other studies describe the process based on source-code [4, 27].

ArgoUML in a Nutshell: ArgoUML is an open-source tool that supports the edition of UML 1.4 diagrams and other related functionalities such as code-generation from UML diagrams or reverse engineering UML diagrams from source code. Its initial release was in 1999 and the latest release was v0.34 in 2011 with 120 KLoC of Java code. Along its history, over 150 developers contributed to its development counting close to 20 thousand commits. More than 6 thousand issues were reported from which around 5 thousand were closed[4]. Relevant fixes were made around 2014 when the activity on its development was abandoned.

In 2010, Marcus Couto, supervised by two software engineering researchers, initiated the re-engineering process of ArgoUML (v0.28.1) to ArgoUML-SPL as part of his master thesis [8]. Figure 1 presents an overview of the re-engineering steps. The developers used as input the ArgoUML CookBook[5] and source-code for conducting four steps, represented by the gray rectangle in Figure 1.

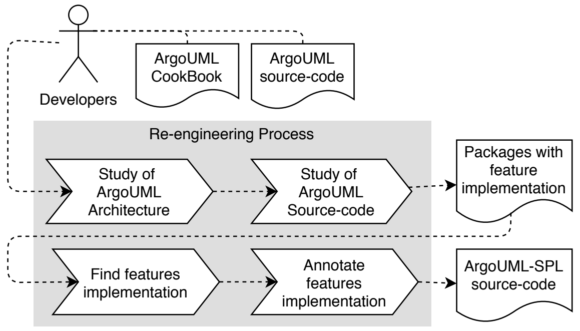

Developers
ArgoUML CookBook
ArgoUML source-code
Re-engineering Process
Study of ArgoUML Architecture
Study of ArgoUML Source-code
Packages with feature implementation
Find features implementation
Annotate features implementation
ArgoUML-SPL source-code

Figure 1: Re-engineering process to extract ArgoUML-SPL from a monolithic version of ArgoUML, adapted from [8].

At the end of this process eight features were made optional by introducing Java annotations. Six features were related to diagrams and two were representative cross-cutting features, namely `Logging` and `Cognitive` support. The latter is responsible for providing critics and warnings regarding UML model instances. This re-engineering was made in 156 commits (from now on, we call them *revisions* following the revision control terminology of SVN[6]) with contributions from three other developers. The latest revisions in 2014 consists only of fixes on the Java comments related to the granularity types of the variability annotations (meta-data on the variability annotations). Those revisions were made by Benjamin Klatt for the case study of his PhD dissertation on consolidating variants in an SPL [22].

3 CASE STUDY DESIGN

We rely on ArgoUML-SPL, mining data from the SVN repository that was used as version control system during the re-engineering process. The design of our case study is based on the Goal/Question/Metric (GQM) approach [38], as described below.

3.1 Study Goal, Questions, and Metrics

Study goal: *understanding architectural, technical, and organizational aspects of the ArgoUML-SPL re-engineering process.*
Research questions (RQ): Guided by our study goal, we derived the following research questions.

[2]https://github.com/jabiercoding/argouml-spl-adoption-dataset.
[3]javapp: http://www.slashdev.ca/javapp

[4]http://argouml.tigris.org/project_bugs.html. Stored (July 2020): https://github.com/argouml-tigris-org/argouml/releases/tag/GITHUB_IMPORT
[5]https://argouml-tigris-org.github.io/tigris/wiki-argouml/wiki
[6]https://subversion.apache.org/

- *RQ1: How has the ArgoUML source code changed during the re-engineering process?* We discuss the modifications for turning a monolithic architecture into an SPL.
  - *Metric:* LoC of each feature, percentage of LoC changes in the architecture, and interactions among features.
- *RQ2: How was the version control system used to conduct the re-engineering process?* Since our source of information is the SVN repository, commonly used in industrial settings, in this question we aim to investigate if the features were extracted all in the same trunk, if there were branches, and the number of revisions in the process. The answer of this question can help practitioners planning to conduct the re-engineering process.
  - *Metric:* number of branches vs number of features (branching strategy), and number of revisions per feature.
- *RQ3: What were the organizational aspects associated to the ArgoUML re-engineering process?* The organizational aspects are related to the time, effort, and team involved in the extraction of ArgoUML-SPL. The goal is to provide insights for those ones willing to move from monolithic systems to SPL architectures.
  - *Metric:* number of months for each feature extraction, number of people, and initial/final revisions.
- *RQ4: What were the problems in the ArgoUML re-engineering process?* Since the re-engineering process is a complex intellectual activity, it is an error-prone task. Here we investigate which and how bugs were found after the SPL extraction, and barriers found by the collaborators during the process.
  - *Metric:* bugs found and inconsistencies detected.

3.2 Repository Data Extraction

A set of scripts were implemented to create a dataset and to automate the analysis[2]. From the 156 revisions of the re-engineering process, we automatically identified the SVN revisions with Java code changes (51 revisions) and we automatically downloaded them. The rest of the revisions were mainly updates of the website content and SVN structure (e.g., empty folders creation). We then extended the ground-truth extractor of the ArgoUML-SPL feature location benchmark [31] to provide metrics on the LoC of each feature and we ran it for each of the revisions. We also identified, for each LoC, whether it was included as part of a feature or as part of the `Core` (the common part).

The initial revisions mention that the source code of the v0.28.1 release was used to extract the ArgoUML-SPL. However, until revision 41, the repository does not have all the source code, but only Java packages and classes where the annotations were included. Starting from revision 41 the source code is complete (both Core and parts with optional code) to build the software. To conduct our analyses in equal conditions for each revision, for revisions prior 41, we included the missing Java classes from v0.28.1.

4 RESULTS AND ANALYSIS

This section presents the results collected from the SVN repository and the analysis in order to answer the posed research questions.

4.1 Making the Architecture Variable

The evolution during the re-engineering process can be mainly analysed by checking the changes in the repository. Figure 2 shows

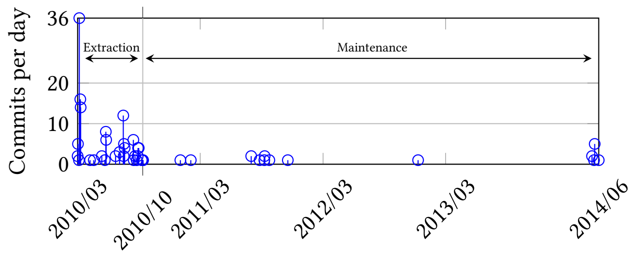


Figure 2: Commits per day in ArgoUML-SPL extraction SVN

the activity in the repository from the day of the first revision until the latest. A vertical line, in October 2010 (9/10/2010) separates two periods that we manually identified analysing the revisions: the SPL extraction itself with significant activity (135 revisions) and the SPL maintenance with sporadic commits (21 revisions).

Figure 3 shows the evolution of the source code while transitioning to a variability-rich system. In this case, the horizontal axis represents the different revisions where Java code changes were made to make the architecture variable; e.g., revision 12 (R12) is the first one with Java changes. Thus, Figure 3 shows time progression in terms of the relevant revisions. The vertical axis corresponds to the percentage of the total LoC of ArgoUML and this way we observe how the code base size evolves in terms of the Core feature and other optional features. A vertical line (R135) also separates the two periods and we can observe how the variable parts did not significantly change during the SPL maintenance period.

In the first revision, the code base corresponds almost completely to the core feature (non-optional part of ArgoUML). As the re-engineering process advances, new optional parts start to appear and grow. The final ArgoUML-SPL has eight optional features and 17 feature interactions, that are presented grouped in Figure 3 as "Interactions". We can observe in Figure 3 that after the last revision, the re-engineering process had impact on 22.78% of the source code, which corresponds to the features that became optional. The core/common implementation of the ArgoUML-SPL represents 77.22% of the code. Measures on the LoC of each feature and its interactions are reported later in Table 1.

4.2 Branching Strategy

Analysing the SVN, we were able to analyse whether the approach consisted on "one feature at a time" process or if a different approach was used. Figure 4 shows the revision history graph regarding the created branches. For four features (`SequenceDiagram`, `UseCaseDiagram`, `CollaborationDiagram`, and `DeploymentDiagram`), feature-specific branches were used before merging. Also, for those branches, features were extracted in parallel (`SequenceDiagram` in parallel with `UseCaseDiagram`, and `CollaborationDiagram` with `DeploymentDiagram`).

The extraction of `ActivityDiagram` started in R12, `Logging` in R17, and `Cognitive` and `StateDiagram` both in R33. According to the feedback from Marcus Couto, the decision of creating branches per feature was taken since the beginning. However, we did not find evidences in the repository about branches for the first four features. Hence, it was probably made without adding the branches in the SVN.

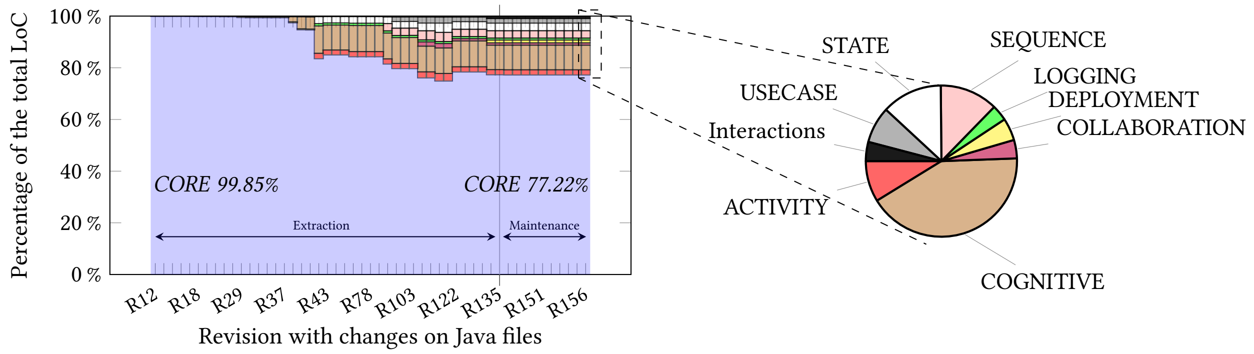


Figure 3: Evolution of the common Core and variable part during the ArgoUML-SPL extraction.

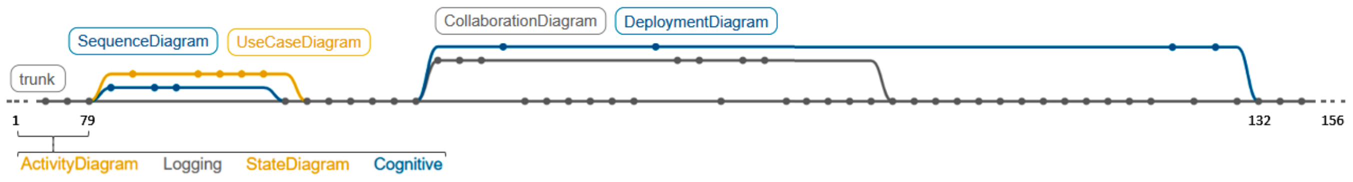


Figure 4: Revision history graph of ArgoUML-SPL SVN. The eight extracted features were extracted in a separated branch and then merged in the trunk. This figure illustrates four branches from where information is available.

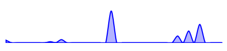
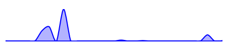
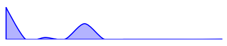
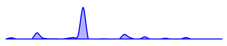
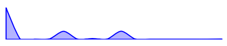
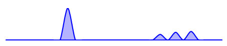
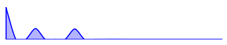
4.3 Organization and extraction effort

The team involved in ArgoUML-SPL extraction was small. Marcus Couto was in charge of all the re-engineering process, except the `UseCaseDiagram` and `SequenceDiagram` branches which were extracted by Camilo Ribeiro. Eduardo Figueiredo and Marco Tulio Valente were two experts supervising the extraction, acting like project managers, but not interacting directly with the repository. Finally, Christian Kästner reported variability-related bug fixes that were solved in R143, and Benjamin Klatt improved the meta-data of the Java variability-related annotations.

To analyse how time-consuming was the extraction of ArgoUML-SPL we measure time from two dimensions. First, the range of revisions in the repository since a feature started to be extracted until its last change or revision to one of its feature interactions (i.e., until the feature was completely stable). Second, the dates from this revision range are used to calculate the number of months. Both time dimensions, together, provide a more complete overview of the time involved in the extraction of each feature. Table 1 summarizes this information including an analysis of the activity on the trunk for the feature under-study (i.e., LoC changes related to this feature or one of their feature interactions) during the revision range.

We can observe interesting facts in Table 1. For instance, `DeploymentDiagram` was the quickest extracted (only 1.53 months, 34 revisions). All the revisions were made in a dedicated branch and, after its merge into the trunk (see Figure 4), no more changes were made to this feature nor to any of its feature interactions. Contrary to this, `Logging` was the feature that took more time to be globally stable. This cross-cutting feature had a sustained activity in the trunk. The features `CollaborationDiagram`, `SequenceDiagram`, and `UseCaseDiagram` had significant activity after their branches merge, that means that their extraction was not completed between merging the branch and the start of the SPL maintenance period.

Table 1: Duration between the start of the extraction until the feature and their corresponding feature interactions got stable. Results are provided for the SPL extraction and, separately, for the whole history including maintenance (global). Also, activity in the trunk in terms of LoC changes.

Feature F. core LoC Interact. LoC	Revisions (global) −branch−	Dates dd/mm/yyyy (global)	Months (global)	Feature Trunk activity Revs with Java changes
Activity 5,950 208	12 ⟶ 132	03/04/2010 27/09/2010	5.9	
Cognitive 28,380 2,069	33 ⟶ 132 (33 ⟶ 143)	03/04/2010 27/09/2010 (07/10/2011)	5.9 (18.4)	
Collaboration 2,676 199	95 ⟶ 132 (95 ⟶ 148) −95 ⟶ 115−	12/08/2010 27/09/2010 (08/06/2014)	1.53 (46.53)	
Deployment 3,243 1,708	98 ⟶ 132 −98 ⟶ 132−	12/08/2010 27/09/2010	1.53	
Logging 2,312 690	17 ⟶ 132 (17 ⟶ 152)	03/04/2010 27/09/2010 (16/06/2014)	5.9 (51.16)	
Sequence 8,437 420	80 ⟶ 123 (80 ⟶ 148) −80 ⟶ 88−	21/06/2010 25/09/2010 (08/06/2014)	3.2 (48.26)	
State 8,801 207	33 ⟶ 132	03/04/2010 27/09/2010	5.9	
UseCase 5,294 60	81 ⟶ 132 (81 ⟶ 156) −81 ⟶ 89−	21/06/2010 27/09/2010 (28/06/2014)	3.26 (48.93)	
		Average:	4.14	

Regarding feature interactions, in the last revision we could observe 17 different types of interactions. Among these interactions, nine are related to the cross-cutting feature `Logging` interaction with some diagram type, three involves the cross-cutting feature `Cognitive` and a diagram type, and five are interactions only between diagrams. On average, each feature interaction has 2.64 revisions, commonly spread along the re-engineering process. The size of these interactions varies, ranging from interactions implemented with only one or two LoC, to one interaction with 1,653 LoC.

4.4 Error-proneness of the Process

In the context of re-engineering ArgoUML into an SPL, we could observe some problems during the conduction of the process.

Incomplete extraction. As mentioned before, around one year after finishing the ArgoUML-SPL re-engineering process, a revision was made with a message describing variability-related bugs pointed out by Christian Kästner. These bugs were found by a tool called LEADT (Location, Expansion, And Documentation Tool), proposed by Kästner et al. [20, 21]. ArgoUML-SPL was used as a subject system to evaluate this tool, finding the bugs that were reported and fixed. This is a lesson learned about how, as in other software engineering projects, unnoticed bugs might appear. As mentioned in Section 4.3, some issues have been treated as well after merging a feature branch in the trunk during the extraction period.

Lack of tool support for variability consistency checks. Feature extraction is an error-prone task. We can see how a typo `SEQUENCEIAGRAM` (missing `D`), introduced inconsistencies in the system in R88 and R89. In addition, we observe evidences of feature name refactorings, e.g., `UMLSTATEDIAGRAM` to `STATEDIAGRAM` in R119, and how it affects also its feature interactions. More advanced tool support for checking the consistency of variability information and for variability-aware refactoring will be desired.

5 DISCUSSION AND THREATS TO VALIDITY

Next the RQs presented in Section 3 are answered.

RQ1: How has the ArgoUML architecture changed during the re-engineering process? Starting from a monolithic architecture, incrementally 135 revisions were needed to achieve an initial SPL architecture which was then maintained until 156 revisions. This final SPL architecture has 77.22% of core/common code, and 22.78% of code belonging to features turned into optional. The final architecture is composed of eight optional features. In addition, there are 17 feature interactions. Analysing the history of revisions, we can observe that many revisions are needed to make the SPL stable.

RQ2: How was the version control system used to conduct the re-engineering process? The developers started using feature-specific branches to perform modifications and then merging them into the trunk. By using the branching strategy, more people could work in parallel during the re-engineering process, keeping a better organization of the code base. Regarding the number of revisions per feature, all features were implemented in many revisions, but spread in different intervals of time (see RQ3). Once the branches were merged, the features continued to evolve in the trunk (except from one feature that did not need more changes), and no branches were created for bug-fixing or improvements.

RQ3: What were the organizational aspects associated to the ArgoUML re-engineering process? There is variation in the time and number of revisions to extract a feature. For example, `DeploymentDiagram` was extracted in 34 revisions and took only 1.53 months. On the other hand, four features took 5.9 months. As a "rule of thumb" extracted from the data in this specific case, the average is 4.14 months per feature and each variable KLoC takes around 0.7 months to be extracted. Taking into account the team, only one collaborator was in charge of the re-engineering process, with sporadic help of a second person. Two software engineering researchers have a role of project managers, supervising the process. Finally, two external people (researches from other institutions) were responsible to bug identification and improving the quality of variability-related meta-data during the SPL maintenance period.

RQ4: What were the problems in the ArgoUML re-engineering process? Some bugs related to variability annotations were found by a tool called LEADT, which automatically verify the source code. This kind of bugs found by LEADT were the result of the manual process of re-engineering a system into an SPL. Therefore, automated support should be used. Another problem related to the extracting of features from an existing system is about the need of naming consistency in the features' annotations. We could observe in the ArgoUML case study that a typo made by one of the developers lead the SPL behaves incorrectly. A tool to avoid these problems is fundamental to support the re-engineering process.

Threats to Validity. The need to create variants of ArgoUML was not an industrial request, but an academic Master thesis. In fact, as far as we know, the SPL was not used beyond research interests. One can argue if it is representative of a re-engineering process in industrial settings. We admit that each case is different and vary in terms of the number of features, their characteristics, the presence of different types of artefacts (not only source code), different binding times, etc. However, the size of ArgoUML is not negligible and, therefore, it cannot be considered a toy or an academic case study. In fact, ArgoUML has features with cross-cutting behavior, one of the most problematic cases usually present in industrial settings. Another threat is the information that is not explicitly available in the repository such as meetings to discuss and exchange domain knowledge. We tried to minimize this threat with feedback from everyone involved in ArgoUML-SPL.

6 CONCLUSIONS

We revisited the ArgoUML monolithic architecture transition to ArgoUML-SPL to provide insights on the process. Based on data mined from the SVN repository used during the re-engineering, we analysed details on how the process was conducted, the source code impact of the SPL extraction, the effort required, and the problems found. The results show that almost one fourth of the ArgoUML source code was changed to tune features to optional, the branching strategy to manage the feature extracting is recommended, the effort to extract an optional feature depends on their size and interactions with other features, and the use of automated tools can help during the process. One case study is not enough to obtain representative effort estimations that can be used in other cases. As further work, we aim to compare the results with other SPLs extracted from monolithic architectures with the goal of aggregating them towards more accurate effort and cost estimation means.

Acknowledgments. We thank ArgoUML-SPL committers M. Couto, M. T. Valente, B. Klatt and C. Kästner for their feedback.

REFERENCES

[1] Ra'Fat AL-Msie'deen, Abdelhak Seriai, Marianne Huchard, Christelle Urtado, Sylvain Vauttier, and Hamzeh Eyal Salman. 2013. Feature Location in a Collection of Software Product Variants Using Formal Concept Analysis. In *Safe and Secure Software Reuse*. Springer Berlin Heidelberg, Berlin, Heidelberg, 302–307.

[2] Muhammad Sarmad Ali, Muhammad Ali Babar, and Klaus Schmid. 2009. A Comparative Survey of Economic Models for Software Product Lines. In *2009 35th Euromicro Conference on Software Engineering and Advanced Applications*. 275–278.

[3] Nasir Ali, Wei Wu, Giuliano Antoniol, Massimiliano Di Penta, Yann-Gaël Guéhéneuc, and Jane Huffman Hayes. 2011. MoMS: Multi-objective miniaturization of software. In *27th IEEE Int. Conf. on Software Maintenance (ICSM)*. 153–162.

[4] Wesley K. G. Assunção, Roberto E. Lopez-Herrejon, Lukas Linsbauer, Silvia R. Vergilio, and Alexander Egyed. 2017. Reengineering legacy applications into software product lines: a systematic mapping. *Empirical Software Engineering* 22, 6 (2017), 2972–3016.

[5] Wesley K.G. Assunção, Silvia R. Vergilio, and Roberto E. Lopez-Herrejon. 2020. Automatic extraction of product line architecture and feature models from UML class diagram variants. *Information and Software Technology* 117 (2020), 106198.

[6] Muhammad Ali Babar, Lianping Chen, and Forrest Shull. 2010. Managing Variability in Software Product Lines. *IEEE Software* 27 (2010), 89–91.

[7] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wasowski. 2013. A survey of variability modeling in industrial practice. In *The 7th Int. Workshop on Variability Modelling of Software-intensive Systems, VaMoS '13, Pisa , Italy, January 23 - 25, 2013*. ACM, 7:1–7:8.

[8] Marcus Couto. 2010. *Extracting Software Product Lines: A Case Study Using Conditional Compilation*. Master's thesis. Pontifical Catholic University of Minas Gerais. https://homepages.dcc.ufmg.br/~mtov/diss/2010_marcus.pdf

[9] Marcus Vinicius Couto, Marco Tulio Valente, and Eduardo Figueiredo. 2011. Extracting Software Product Lines: A Case Study Using Conditional Compilation. In *15th European Conference on Software Maintenance and Reengineering, CSMR 2011, 1-4 March 2011, Oldenburg, Germany*. IEEE Computer Society, 191–200.

[10] Deepak Dhungana, Paul Grünbacher, and Rick Rabiser. 2011. The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study. *Automated Software Engineering* 18, 1 (01 Mar 2011), 77–114.

[11] Hamzeh Eyal-Salman, Abdelhak-Djamel Seriai, and Christophe Dony. 2013. Feature-to-code traceability in a collection of software variants: Combining formal concept analysis and information retrieval. In *IEEE 14th International Conference on Information Reuse Integration (IRI)*. 209–216.

[12] Hamzeh Eyal-Salman, Abdelhak-Djamel Seriai, and Chistophe Dony. 2013. Feature-to-Code Traceability in Legacy Software Variants. In *39th Euromicro Conference on Software Engineering and Advanced Applications*. 57–61.

[13] Hamzeh Eyal-Salman, Abdelhak-Djamel Seriai, and Christophe Dony. 2014. Feature Location in a Collection of Product Variants: Combining Information Retrieval and Hierarchical Clustering. In *SEKE: Software Engineering and Knowledge Engineering*. Vancouver, Canada, 426–430.

[14] Hamzeh Eyal-Salman, Abdelhak-Djamel Seriai, Christophe Dony, and Ra'Fat Ahmad Al-Msie'Deen. 2013. Identifying Traceability Links between Product Variants and Their Features. In *REVE: Reverse Variability Engineering*. Genova, Italy, 17–22.

[15] Stefan Fischer, Lukas Linsbauer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. 2014. Enhancing Clone-and-Own with Systematic Reuse for Developing Software Variants. In *IEEE International Conference on Software Maintenance and Evolution*. 391–400.

[16] Negar Hariri, Carlos Castro-Herrera, Mehdi Mirakhorli, Jane Cleland-Huang, and Bamshad Mobasher. 2013. Supporting Domain Analysis through Mining and Recommending Features from Online Product Listings. *IEEE Transactions on Software Engineering* 39, 12 (2013), 1736–1752.

[17] Ahmed E Hassan. 2008. The road ahead for mining software repositories. In *2008 Frontiers of Software Maintenance*. IEEE, 48–57.

[18] Ying Hu, E. Merlo, M. Dagenais, and B. Lague. 2000. C/C++ Conditional Compilation Analysis using Symbolic Execution. In *30th Int. Conference on Software Maintenance (ICSM '00)*. ACM, New York, NY, USA, 10.

[19] Christian Kästner, Sven Apel, and Martin Kuhlemann. 2008. Granularity in Software Product Lines. In *30th Int. Conference on Software Engineering* (Leipzig, Germany) *(ICSE '08)*. ACM, New York, NY, USA, 311–320.

[20] Christian Kästner, Alexander Dreiling, and Klaus Ostermann. 2011. *Variability Mining with LEADT*. Technical Report 01/2011. Department of Mathematics and Computer Science, Philipps University Marburg, Marburg, Germany. http://www.uni-marburg.de/fb12/forschung/berichte/berichteinformtk

[21] Christian Kästner, Alexander Dreiling, and Klaus Ostermann. 2014. Variability Mining: Consistent Semi-automatic Detection of Product-Line Features. *IEEE Trans. Software Eng.* 40, 1 (2014), 67–82.

[22] Benjamin Klatt. 2014. *Consolidation of Customized Product Copies into Software Product Lines*. Ph.D. Dissertation. Karlsruhe Institute of Technology, Germany. http://digbib.ubka.uni-karlsruhe.de/volltexte/1000043687

[23] Benjamin Klatt, Klaus Krogmann, and Christoph Seidl. 2014. Program Dependency Analysis for Consolidating Customized Product Copies. In *IEEE International Conference on Software Maintenance and Evolution*. 496–500.

[24] Jens Knodel and Dirk Muthig. 2005. Analyzing the product line adequacy of existing components. In *1st Int. Workshop on Reengineering towards Product Lines. R2PL*. 21–25.

[25] Ronny Kolb, Dirk Muthig, Thomas Patzke, and Kazuyuki Yamauchi. 2006. Refactoring a legacy component for reuse in a software product line: a case study. *Journal of Software Maintenance and Evolution: Research and Practice* 18, 2 (2006), 109–132.

[26] Charles W. Krueger. 2001. Easing the Transition to Software Mass Customization. In *Software Product-Family Engineering, 4th Int. Workshop, PFE 2001, Bilbao, Spain, October 3-5, 2001, Revised Papers (Lecture Notes in Computer Science)*, Vol. 2290. Springer, 282–293.

[27] Miguel A. Laguna and Yania Crespo. 2013. A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. *Science of Computer Programming* 78, 8 (2013), 1010–1034. Special section on software evolution, adaptability, and maintenance - Special section on the Brazilian Symposium on Programming Languages.

[28] Lukas Linsbauer, E. Roberto Lopez-Herrejon, and Alexander Egyed. 2013. Recovering Traceability Between Features and Code in Product Variants. In *17th International Software Product Line Conference* (Tokyo, Japan) *(SPLC '13)*. ACM, New York, NY, USA, 131–140.

[29] Lukas Linsbauer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. 2014. Feature Model Synthesis with Genetic Programming. In *Search-Based Software Engineering*. Springer International Publishing, Cham, 153–167.

[30] Jabier Martinez, Wesley K. G. Assunção, and Tewfik Ziadi. 2017. ESPLA: A Catalog of Extractive SPL Adoption Case Studies. In *Proceedings of the 21st Int. Systems and Software Product Line Conference, SPLC 2017, Volume B, Sevilla, Spain, September 25-29, 2017*. ACM, 38–41.

[31] Jabier Martinez, Nicolas Ordoñez, Xhevahire Tërnava, Tewfik Ziadi, Jairo Aponte, Eduardo Figueiredo, and Marco Tulio Valente. 2018. Feature location benchmark with argoUML SPL. In *Proceeedings of the 22nd Int. Systems and Software Product Line Conference - Volume 1, SPLC 2018, Gothenburg, Sweden, September 10-14, 2018*. ACM, 257–263.

[32] Jabier Martinez, Tewfik Ziadi, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. Name Suggestions During Feature Identification: The Variclouds Approach. In *20th International Systems and Software Product Line Conference* (Beijing, China) *(SPLC '16)*. ACM, New York, NY, USA, 119–123.

[33] Jabier Martinez, Tewfik Ziadi, Tegawendé F. Bissyandé, Jacques Klein, and Yves le Traon. 2015. Automating the Extraction of Model-Based Software Product Lines from Model Variants (T). In *30th IEEE/ACM International Conference on Automated Software Engineering*. 396–406.

[34] Linda M. Northrop and Paul C. Clements. 2012. A Framework for Software Product Line Practice Version 5.0. (2012). http://www.sei.cmu.edu/plp/framework.html

[35] Andrzej Olszak and Bo Nørregaard Jørgensen. 2012. Remodularizing Java programs for improved locality of feature implementations in source code. *Science of Computer Programming* 77, 3 (2012), 131–151. Feature-Oriented Software Development (FOSD 2009).

[36] Jun Otsuka, Kouichi Kawarabata, Takashi Iwasaki, Makoto Uchiba, Tsuneo Nakanishi, and Kenji Hisazumi. 2011. Small Inexpensive Core Asset Construction for Large Gainful Product Line Development: Developing a Communication System Firmware Product Line. In *15th Int. Software Product Line Conference, Volume 2* (Munich, Germany) *(SPLC '11)*. ACM, New York, NY, USA, 20:1–20:5.

[37] Daniel Strüber, Mukelabai Mukelabai, Jacob Krüger, Stefan Fischer, Lukas Linsbauer, Jabier Martinez, and Thorsten Berger. 2019. Facing the truth: benchmarking the techniques for the evolution of variant-rich systems. In *Proceedings of the 23rd Int. Systems and Software Product Line Conference, SPLC 2019, Volume A, Paris, France, September 9-13, 2019*. ACM, 26:1–26:12.

[38] Rini van Solingen, Vic Basili, Gianluigi Caldiera, and H. Dieter Rombach. 2002. *Goal Question Metric (GQM) Approach*. John Wiley Sons, Inc.

[39] Daniele Wolfart, Wesley Klewerton Guez Assunçao, and Jabier Martinez. 2019. Open Source Software on the Research of Extractive Adoption of Software Product Lines. In *Proceeedings of Latin.Science Latinoware*.

[40] Yiming Yang, Xin Peng, and Wenyun Zhao. 2009. Domain Feature Model Recovery from Multiple Applications Using Data Access Semantics and Formal Concept Analysis. In *16th Working Conference on Reverse Engineering*. 215–224.

[41] Gang Zhang, Liwei Shen, Xin Peng, Zhenchang Xing, and Wenyun Zhao. 2011. Incremental and iterative reengineering towards Software Product Line: An industrial case study. In *27th IEEE Int. Conf. on Software Maintenance (ICSM)*. 418–427.

[42] Tewfik Ziadi, Luz Frias, Marcos Aurélio Almeida da Silva, and Mikal Ziane. 2012. Feature Identification from the Source Code of Product Variants. In *16th European Conference on Software Maintenance and Reengineering*. 417–422.

[43] Tewfik Ziadi and Lom Messan Hillah. 2018. Software Product Line Extraction from Bytecode Based Applications. In *23rd International Conference on Engineering of Complex Computer Systems (ICECCS)*. 221–225.