Analyzing the Impact of Refactoring Variants on Feature Location

Amine Benmerzoug, Lamia Yessad, Tewfik Ziadi

Analyzing the Impact of Refactoring Variants on Feature Location

Amine Benmerzoug[1][0000−0003−3770−8339]⋆, Lamia Yessad[1][0000−0001−5457−6425], and Tewfik Ziadi[2][0000−0001−9241−8276]

[1] Ecole Nationale Supérieure d'Informatique, LCSI, Algiers, Algeria
{a_benmerzoug,l_yessad}@esi.dz
[2] Sorbonne Université, LIP6, Paris, France
tewfik.ziadi@lip6.fr

Abstract. Due to the increasing importance of feature location process, several studies evaluate the performance of different techniques based on IR strategies and a set of software variants as input artifacts. The proposed techniques attempt to improve the results obtained but it is often a difficult task. None of the existing feature location techniques considers the changing nature of the input artifacts, which may undergo series of refactoring changes. In this paper, we investigate the impact of refactoring variants on the feature location techniques. We first evaluate the performance of two techniques through the ArgoUML SPL benchmark when the variants are refactored. We then discuss the degraded results and the possibility of restoring them. Finally, we outline a process of variant alignment that aims to preserve the performance of the feature location.

Keywords: Software Product Line · Feature Location · Refactoring.

1 Introduction

Software Product Lines (SPL) represent one of the most exciting paradigm shift in software development in the last two decades [14]. The main idea is to implement at the same time a family of similar applications rather than implementing a single system. The SPL engineering framework introduces into the general process of software development new activities related to software variability management and product derivation based mainly on the concept of *features* [1]. A feature refers to a specific functionality or characteristic of a software [10]. During these decades, many concrete approaches have been proposed (ex. Sven et al. [1]). However, adopting an SPL approach and designing SPL variability is still a major challenge and represents a risk for a company [12]. Berger et al. [3] showed, in a survey with industrial companies, which participated in industrial SPLE, that around 50% of them cannot adopt SPL proactively [3]. Indeed, instead of adopting an SPL, these companies clone an existing product

⋆ Corresponding author.

and modify it to fit customer requirements. This approach, called *clone-and-own*, is widely used because it is faster and more efficient to start with an already developed and tested set of variants. Thereby, the extractive approach for SPL adoption or SPL Reengineering is gaining ground. It consists in migrating, automatically or semi-automatically, the existing variants into an SPL. One of the main steps in the extractive approach is what is referred to as Feature Location (FL). FL is a traceability recovery task for identifying the implementation elements associated to each feature among the family variants that are created using clone-and-own [16].

However, all FL techniques are built on the same assumption that the input variants only differ in term of features and do not consider the situation where some changes are applied on some variants, without a complete propagation to all variants. This kind of evolution is only introduced to improve the quality of the source code without introducing any variation in terms of features. *Code Refactoring* [9] is an example of such evolution.

This paper presents a study to investigate the impact of the evolution related to code refactoring of the variants on the process of feature location and preform experiments to quantify the impacts introduced by refactorings. We particularly consider the following Research Questions:

- (RQ1): Does refactoring affect feature location results?
- (RQ2): How to cover the negative impact of refactoring on feature location techniques?
- (RQ3): What is the new vision to implement for preserving the performance of a feature location technique?

To answer these questions, we have conducted a study on two feature location techniques through the ArgoUML SPL benchmark [15]. The study consists of forty-two experiments:

- The first twenty-two experiments aim to observe and analyze if the two location techniques performed are impacted by refactorings (RQ1). Two hundred fifty refactorings are applied to analyse this impact.
- The rest of experiments aims to confirm the possibility of restoring performance of the same techniques when propagating the changes to all variants (RQ2 and RQ3). It means that unchanged variants must be modified according to the existing refactorings.

The contributions of this paper can be summarized as follows:

- We use the ArgoUML SPL benchmark to do further experiments measuring the negative impact of refactoring variants on feature location techniques.
- We identify that the negative impact represented by the distance we called *Degraded Degree* evolves linearly based on the number of the applied refactorings.
- We outline a preliminary process that aims to align the variants, i.e. propagate refactorings to all variants before performing the feature location.

The rest of the paper is organized as follows. Section 2 presents the feature location (FL) and refactoring techniques as well as the problem statement. Section 3 describes our study design and section 4 discusses the results. This section also outlines the main activities towards an alignment process. Finally, section 5 presents related work before concluding the paper in section 6.

2 Background and Problem Statement

2.1 Feature Location for Software Product Line Reengineering

As shown by Martinez et al. [16], feature location in the extractive approach of SPLE can be illustrated by the Fig. 1. It takes as input a set of variants created using clone-and-own. For each of the variants, there are implementation elements (represented as a set diamonds in Fig. 1) and the information of which features are implemented. For example, Variant 1 implements features F1, F2 and F3 whereas Variant 2 implements F1 and F3. In this context of feature-based variants, a specific FL technique takes as inputs the information of all the variants (features and implementation elements) and finds, for each feature, which are the associated implementation elements as shown at the bottom of Fig. 1.

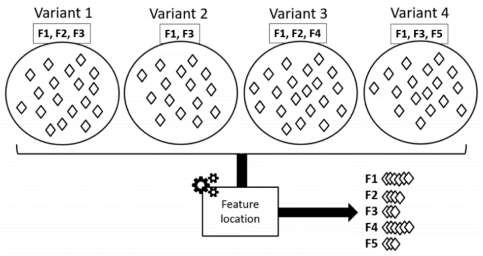

Fig. 1. Feature location in feature-based variants [16].

Rubin and Chechik [20] and Assunção et al. [2] conducted surveys about the state-of-the-art in FL domain with a large variety of approaches. For instance, many approaches use techniques from the field of Information Retrieval (IR) such as Formal Concept Analysis (FCA) [25], Document Vectors (DVs) [13] and Latent Semantic Indexing (LSI) [8].

2.2 Problem Statement

As mentioned above, feature location techniques for feature-based variants are built on the same assumption that the input variants are very similar, and they only differ in terms of features. However, the variants created using clone-and-own can evolve independently. Many evolutions that are not related to features

can be applied without a complete propagation to all variants. An example of such evolutions is related to code refactoring that can be introduced to improve the quality of the source code of a specific variant without adding or removing features [9]. Fowler [9] proposed a refactoring catalog[1] that contains from a simple *Rename field* to more complex operations such as *Extract method* or *Move method*.

To illustrate the impact of refactoring applied on variants created using clone-and-own, let us consider the example of the banking system [26]. Fig. 2 shows the source code of the *Account* class of two variants denoted `Product1Bank` and `Product2Bank`. `Product1Bank` implements the *Base* and *Limit* features whereas `Product2Bank` implements the *Base*, *Limit* and *CurrencyExchange* features. The *Base* feature represents the mandatory part in all possible variants and the *Limit* feature is mainly implemented with the `limit` property and its associated getter.

Fig. 2 (a) shows a simple type of refactoring called *Rename field*, where the `limit` field in *Account* class in `Product1Bank` is changed into a more meaningful name `limitOnAccount` (all the updated references are highlighted in green). Another example of refactoring illustrated in Fig. 2 (b) is *Extract method*, where the condition statement in *withdraw* method in `Product2Bank` is extracted into the new method *isSufficient*, which returns a boolean. Thus, the condition statement is replaced with the call to the new method (both the call and the new method are highlighted in yellow).

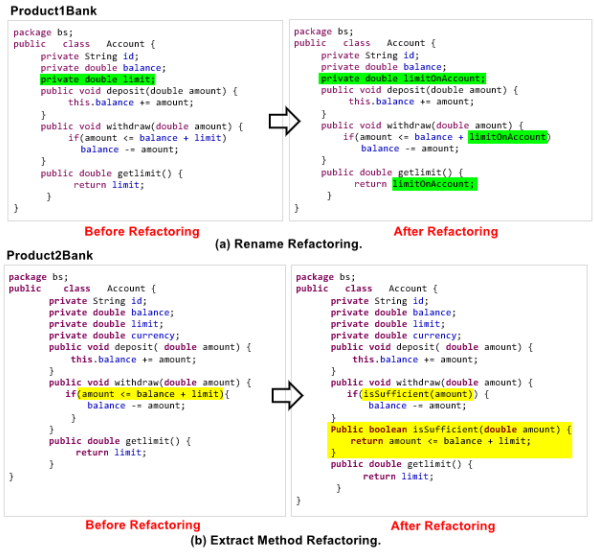

Fig. 2. Rename field and Extract method refactorings in the "Banking System".

[1] Refactoring.com [online]. [date of reference: July 3rd of 2019]. Available at: https://refactoring.com/catalog/

Applying refactoring by renaming the `limit` field only in `Product1Bank` may have an impact on FL. Indeed, the implementation of the *Limit* Feature only considers one identifier and not both. The extraction of the new method in `Product2Bank` may also impact the results of FL because the change is not propagated to `Product1Bank`.

In this simple example and even if we only applied two refactoring operations on a single class, FL results can be significantly impacted. This paper aims presenting a deeper study to evaluate systematically this impact according to the number of refactoring operations. The next sections present the design of this study using the ArgoUML SPL benchmark and highlight the results.

3 Study Design

3.1 ArgoUML SPL Benchmark for FL

To evaluate FL in feature-based variants, the ArgoUML SPL benchmark was proposed as a facto platform for evaluating FL techniques [15]. This benchmark is based on the ArgoUML SPL which is extracted as a product line from an open-source tool for UML modeling [4]. Fig. 3 presents the feature model of the ArgoUML SPL where variability is mainly related to the support of the different UML diagrams. For instance, the feature *State* is the functionality related to the UML state diagrams and it is defined as optional whereas the class diagram is mandatory. The implementation of ArgoUML SPL is coded in Java using an annotative approach based on the well-known `#ifdef` directives.

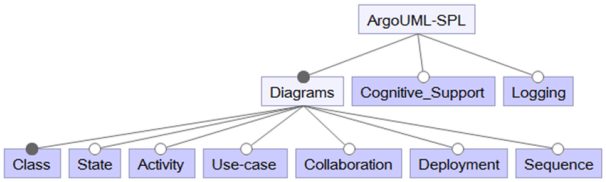

Fig. 3. ArgoUML SPL feature model [4].

The ArgoUML SPL benchmark was initially proposed as a challenge at System and Software Product line Conference (SPLC 2018) [15]. The idea is to propose for the research community a benchmark to implement and evaluate FL techniques. The benchmark provides scenarios, ground-truth, and metrics calculation:

- There are 15 scenarios helping to generate up to 256 different variants.
- The ground-truth contains 24 text files corresponding to each feature and its combinations. These files contain traceability information to classes, methods, and their refinements.

- The three traditional metrics *precision*, *recall*, and *F-score* are computed to evaluate the retrieval effectiveness of traceability information.

The results format of any feature location techniques that uses the benchmark should be adapted to the same format as the ground-truth where the text file name is the name of the features or their combination or feature negation.

3.2 Data Preparation Procedure

In this study, we used ArgoUML SPL to generate a data set of five variants using predefined Random scenario (one of the 15 scenarios). Our study is divided into three steps: Before Refactoring, After Refactoring and After Alignment.

Data Preparation Before Refactoring In this step, we consider the generated five variants without modification as input artifacts of the location technique. This latter locates the source code of each feature knowing that the set of features in each variant product is already identified.

We perform the experiment #1 using the DVs location technique and the experiment #2 with the LSI technique. The results are the same obtained by Cruz et al. [5] and are taken as a point of comparison.

Data Preparation After Refactoring In this step, we apply several types of refactoring on each variant. Here, it is important to consider the number of refactorings applied on variants, denoted `NREFACT`. For that, we perform 20 experiments (10 for each technique) starting with 25 refactorings of several types (*Rename class/method/field*, *Extract class/method*). Then, the number of refactorings is increased by 25 until it reaches 250 different refactorings (see Table 1).

Table 1. Refactorings variation

Experiment (DVs)	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
Experiment (LSI)	#13	#14	#15	#16	#17	#18	#19	#20	#21	#22
NREFACT	25	50	75	100	125	150	175	200	225	250

For applying the refactorings, we use two automatic refactoring tools: *Jextract*[2] [21] and *JDeodorant*[3] [17]. The *Jextract* tool is based on the similarity structure of the system to identify refactoring opportunities whereas *JDeodorant* identifies code smells in software and outputs the appropriate refactorings to resolve them. We use the two tools for extracting methods and classes. However, the rename is performed by reviewing source code and giving meaningful names to fields, methods and classes using the Eclipse rename functionality.

[2] https://github.com/aserg-ufmg/jextract
[3] https://github.com/tsantalis/JDeodorant

Data Preparation After Alignment It is the third and final step of our study. It also contains 20 experiments (10 for each location technique). We keep track of all refactoring operations for each variant using Eclipse refactoring history, which provides the needed information to create refactoring scripts. So, we use these scripts to automatically propagate the applied refactorings (in one or many variants) to all variants. For each experiment, the input artifacts still the five variants but with additional refactorings.

In the experiment #12 (see Table 1), we have a priori five refactored variants with 250 refactorings in total. Five scripts are then created where each script capitalizes the refactorings applied on one variant. Then, scripts are used to propagate the refactorings to all variants. Table 2 presents the number of refactorings applied successfully. The objective of alignment is to respond to the RQ2 and RQ3 research questions.

Table 2. Number of refactorings after Alignment

Experiment (DVs)	#23	#24	#25	#26	#27	#28	#29	#30	#31	#32
Experiment (LSI)	#33	#34	#35	#36	#37	#38	#39	#40	#41	#42
NREFACT	103	198	284	332	386	427	503	608	646	687

Unfortunately, as we see in Table 2, not all refactorings can be applied because of conflicts. A conflict arises when two or more separate refactoring operations should be applied into the same code element (field, class, method, package), or when the element has been deleted.

3.3 Feature Location Workflows

As previously mentioned, we use two automatic IR-based feature location techniques DVs and LSI. Both extract from input artifacts (the five variants) methods and classes, then generate for each of them a document. Each variant is treated separately and have a set of characteristics such as number of features, number of LOCs (NLOCs) and number of generated documents (see Table 3). Furthermore, they give as results the code source of features in the same ground-truth's format. In that way, the ground-truth can be used to calculate the performance metrics: precision, recall and F-score.

Table 3. Variant characteristics

Variant	**Features**	**NLOCs**	**Documents**
1	05	305,970	15,563
2	05	308,821	15,475
3	05	295,927	14,881
4	05	327,311	16,168
5	05	337,940	16,730

3.4 Degradation-based Evaluation

We define a new evaluation metric *Degradation Degree*, denoted `DD` to measure the performance degradation of a feature location technique. This metric is a distance between two `F-score` values (obtained from ArgoUML SPL benchmark). Thus, it is based on `F-Score` metric and defined by the following formula:

$$\texttt{DD (Experiment \#n)} = FScore_0 - \texttt{FScore (Experiment \#n)} \quad (1)$$

Where `DD` refers to the Degradation Degree qualifying each experiment after alignment. $FSore_0$ denotes the F-score obtained before refactoring depending on what technique we use, and `#n` denotes the experiment number.

4 Results for Answering the Research Questions

- ***RQ1:*** Does refactoring affect feature location results?

To answer RQ1, we perform for each technique 10 experiments by varying the number of refactoring operations. The ArgoUML SPL benchmark provides the values of precision (P), recall (R) and F-score (F). However, DD values are calculated using the precedent formula (1). The results are shown in Table 4.

Table 4. Evolution of `DD` depending on the number of refactorings using DVs and LSI

	DVs				LSI			
NREFACT	P	R	F	DD	P	R	F	DD
0	0.04470	0.04292	0.04379	0.00000	0.16083	0.19439	0.17602	0.00000
25	0.04309	0.03954	0.04124	0.00256	0.16020	0.18254	0.17143	0.00459
50	0.03788	0.03206	0.03473	0.00907	0.15384	0.16955	0.16131	0.01471
75	0.03109	0.02446	0.02738	0.01642	0.15084	0.15955	0.15507	0.02095
100	0.02847	0.02107	0.02422	0.01957	0.14189	0.12956	0.13544	0.04058
125	0.02147	0.01611	0.01841	0.02539	0.13124	0.10196	0.11476	0.06126
150	0.01502	0.01410	0.01454	0.02925	0.08608	0.07436	0.07979	0.09623
175	0.00645	0.00735	0.00687	0.03692	0.05908	0.04744	0.05262	0.12340
200	0.00375	0.00430	0.00400	0.03979	0.02096	0.02981	0.02461	0.15141
225	0.00170	0.00295	0.00216	0.04164	0.00811	0.00930	0.00866	0.16736
250	0.00044	0.00064	0.00052	0.04327	0.00073	0.00085	0.00153	0.17450

In both techniques, we observe that `DD` increase when `NREFACT` increase. Furthermore, the two variables are correlated with a coefficient of correlation equal to 9.90E-01 (≈1) for DVs (resp. 9.81E-01 for LSI). Thus, we apply a linear regression, which is a technique of machine learning used to obtain a mathematical model. This latter allows developers to make predictions for (or extrapolate) the value of `DD` depending on the different values of `NREFACT` (see Fig. 4). The regression equations ((2) for DVs and (3) for LSI) obtained are:

$$\texttt{DD} = 0.000187 \times \texttt{NREFACT} + 0.000555 \quad (2)$$

$$\texttt{DD} = 0.000798 \times \texttt{NREFACT} - 0.022011 \quad (3)$$

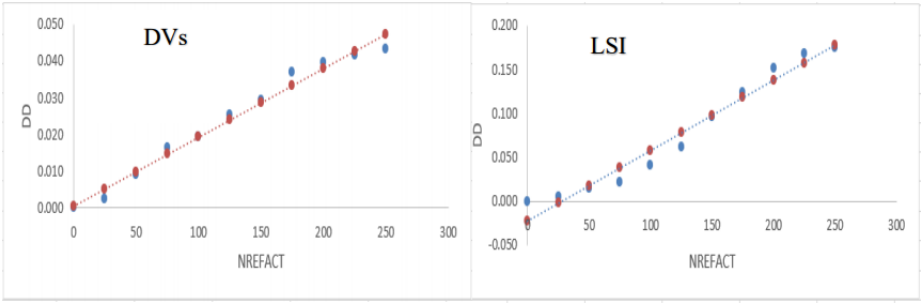

Fig. 4. Regression lines to predict degradation degree based on `NREFACT`.

For example, in our context, if the value of `NREFACT` is equal to 500, the expected value of the degradation degree `DD` will be equal to 0.094289 for DVs (resp. 0.376943 for LSI).

- ***RQ2:*** How to cover the negative impact of refactoring on feature location techniques?

To answer RQ2, we align the variants. As we mentioned before, we keep track of all refactoring operations in script files. Then, we perform the scripts to apply the possible refactorings on all variants. Our objective is to reduce the negative impact of refactoring variants on the feature location techniques. Fig. 5 shows that the alignment allows slightly improving the precision and the recall metrics (and thus F-score). The results indicate that the alignment of variants could restore the performance for both techniques.

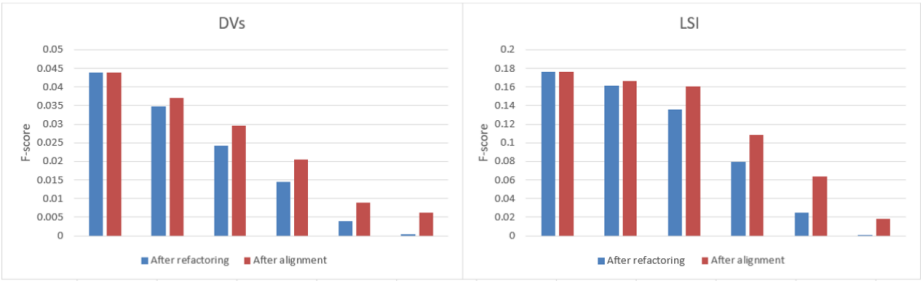

Fig. 5. Slightly improvement of F-score after Alignment.

• ***RQ3:*** What is the new vision to implement for preserving the performance of a feature location technique?

Our study shows that the refactoring of variants has a negative impact on the feature location processes and the alignment of variants could deal with this problem (see Fig. 5). The alignment performed in our study is not suitable because of increasing conflictual refactorings. Therefore, we are currently working to propose a new *Alignment process*, which can include the following three activities:

1. Refactoring identification that aims to automatically identify refactoring operations by analyzing source code related to the variants to be inputted.
2. Refactoring selection that has the role of selecting the suitable set of refactorings to be propagated to all variants in a systematic way avoiding conflicts as much as possible.
3. Alignment that propagates automatically the selected refactorings to all variants.

5 Related Work

Existing feature location techniques can be mainly grouped into two categories: Static and Dynamic [5, 7]. None of these feature location techniques consider the changes of refactorings. There are a number of benchmarks [16, 15, 6] that have been used to evaluate FL techniques performance [5, 24, 18], while we used one of them to estimate the performance of FL techniques with refactored variants. We have previously presented the ArgoUML SPL benchmark, which have been used to evaluate many FL techniques [5, 18, 19]. In addition, Martinez et. al [16] have proposed the use of eclipse variants to evaluate feature location techniques. This benchmark is mainly based on the assumption that eclipse features can be mapped to SPL features and proposes a generation of the ground-truth of the benchmark from a specific eclipse distribution. Many techniques have been evaluated using this benchmark. This benchmark is also useful to evaluate FL but, without considering the noises introduced by refactoring operations. As we presented in our study, our objective is to identify the impact of refactoring variants on the FL process.

Many approaches have also been proposed to identify refactorings in the source code [11, 22, 23]. Unfortunately, these approaches only identify refactorings from the source code of a single application. As we discussed in our Alignment Process, one interesting direction is to extend these existing approaches by first identifying refactorings from many variants and then, applying a systematic alignment process to make the refactoring activities transparent during the FL process.

6 Conclusion

In this paper, we have presented a study on the impact of refactorings on feature location techniques using ArgoUML SPL benchmark. We apply Document

Vectors (DVs) and Latent Semantic indexing (LSI), automatic feature location approaches based on textual information retrieval techniques. We find that refactorings of a variant subset reduce the techniques performance and propagating these refactorings to all variants (alignment) slightly improves the results.

Our study provides a set of experiments to show the negative impact of refactorings on the feature location techniques by calculating the degradation degree (DD). The value of DD increases linearly based on the number of the applied refactorings. Thus, the value of DD can be extrapolated.

However, the experimental results obtained after alignment are not good because the use of an ad-hoc technique. As future work, we will implement the outlined alignment process and experiment it with other feature location techniques using different benchmarks.

References

1. Apel, S., Batory, D.S., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines - Concepts and Implementation. Springer, heidelberg, Berlin (2013)
2. Assunção, W.K., Lopez-Herrejon, R.E., Linsbauer, L., Vergilio, S.R., Egyed, A.: Reengineering legacy applications into software product lines: a systematic mapping. Empirical Software Engineering **22**(6), 2972–3016 (2017)
3. Berger, T., Rublack, R., Nair, D., Atlee, J.M., Becker, M., Czarnecki, K., Wasowski, A.: A survey of variability modeling in industrial practice. In: Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems. pp. 1–7. ACM, Pisa, Italy (2013)
4. Couto, M.V., Valente, M.T., Figueiredo, E.: Extracting software product lines: A case study using conditional compilation. In: 15th European Conference on Software Maintenance and Reengineering. pp. 191–200. IEEE Computer Society, Oldenburg, Germany (2011)
5. Cruz, D., Figueiredo, E., Martinez, J.: A literature review and comparison of three feature location techniques using argouml-spl. In: Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems. pp. 1–10. ACM, Leuven, Belgium (2019)
6. Dit, B., Holtzhauer, A., Poshyvanyk, D., Kagdi, H.H.: A dataset from change history to support evaluation of software maintenance tasks. In: Proceedings of the 10th Working Conference on Mining Software Repositories. pp. 131–134. IEEE Computer Society, San Francisco, USA (2013)
7. Dit, B., Revelle, M., Gethers, M., Poshyvanyk, D.: Feature location in source code: a taxonomy and survey. J. Softw. Evol. Process. **25**, 53–95 (2013)
8. Dumais, S.T., Furnas, G.W., Landauer, T.K., Deerwester, S., Harshman, R.: Using latent semantic analysis to improve access to textual information. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 281–285. Association for Computing Machinery, New York, USA (1988)
9. Fowler, M.: Refactoring Improving the Design of Existing Code. Addison-Wesley, Boston, MA, USA (1999)
10. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.S.: Feature-oriented domain analysis (foda) feasibility study. software engineering institute. Universitas Carnegie Mellon, Pittsburgh, Pennsylvania (1990)

11. Kim, M., Gee, M., Loh, A., Rachatasumrit, N.: Ref-finder: a refactoring reconstruction tool based on logic query templates. In: Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 371–372. ACM, Santa Fe, NM, USA (2010)
12. Krueger, C.W.: Variation management for software production lines. In: International Conference on Software Product Lines. pp. 37–48. Springer, San Diego, CA, USA (2002)
13. Le, Q.V., Mikolov, T.: Distributed representations of sentences and documents. In: Proceedings of the 31th International Conference on Machine Learning. pp. 1188–1196. JMLR.org, Beijing, China (2014)
14. Linden, F.J.v.d., Schmid, K., Rommes, E.: Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. Springer-Verlag, Berlin, Heidelberg (2007)
15. Martinez, J., Ordoñez, N., Tërnava, X., Ziadi, T., Aponte, J., Figueiredo, E., Valente, M.T.: Feature location benchmark with argouml SPL. In: Proceeedings of the 22nd International Systems and Software Product Line Conference. vol. 1, pp. 257–263. ACM, Gothenburg, Sweden (2018)
16. Martinez, J., Ziadi, T., Papadakis, M., Bissyandé, T.F., Klein, J., Traon, Y.L.: Feature location benchmark for extractive software product line adoption research using realistic and synthetic eclipse variants. Information and Software Technology pp. 46–59 (2018)
17. Mazinanian, D., Tsantalis, N., Stein, R., Valenta, Z.: Jdeodorant: clone refactoring. In: Proceedings of the 38th International Conference on Software Engineering. pp. 613–616. ACM, Austin, TX, USA (2016)
18. Michelon, G.K., Linsbauer, L., Assunção, W.K.G., Egyed, A.: Comparison-based feature location in argouml variants. In: Proceedings of the 23rd International Systems and Software Product Line Conference. pp. 1–5. ACM, Paris, France (2019)
19. Müller, R., Eisenecker, U.: A graph-based feature location approach using set theory. In: Proceedings of the 23rd International Systems and Software Product Line Conference. pp. 88–92. Association for Computing Machinery, NY, USA (2019)
20. Rubin, J., Chechik, M.: A survey of feature location techniques. In: Domain Engineering, pp. 29–58. Springer (2013)
21. Silva, D., Terra, R., Valente, M.T.: Jextract: An eclipse plug-in for recommending automated extract method refactorings. CoRR pp. 1–8 (2015)
22. Silva, D., Valente, M.T.: Refdiff: detecting refactorings in version histories. In: Proceedings of the 14th International Conference on Mining Software Repositories. pp. 269–279. IEEE Computer Society, Buenos Aires, Argentina (2017)
23. Tan, L., Bockisch, C.: A survey of refactoring detection tools. In: 6th Collaborative Workshop on Evolution and Maintenance of Long Living Systems. pp. 100–105. CEUR-WS.org, Stuttgart, Germany (2019)
24. Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T.: Featureide: An extensible framework for feature-oriented software development. Science of Computer Programming pp. 70–85 (2014)
25. Wille, R.: Formal concept analysis as mathematical theory of concepts and concept hierarchies. In: Ganter, B., Stumme, G., Wille, R. (eds.) Formal Concept Analysis, Foundations and Applications. vol. 3626, pp. 1–33. Springer (2005)
26. Ziadi, T., Henard, C., Papadakis, M., Ziane, M., Traon, Y.L.: Towards a language-independent approach for reverse-engineering of software product lines. In: Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea - March 24 - 28, 2014. pp. 1064–1071. ACM (2014)