Spectrum-Based Feature Localization: A Case Study using ArgoUML

Gabriela K. Michelon
Johannes Kepler University Linz
Linz, Austria
gabriela.michelon@jku.at

Bruno Sotto-Mayor
Ben Gurion University of the Negev
Be'er Sheva, Israel
mail@brunosottomayor.com

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

Aitor Arrieta
University of Mondragon
Mondragon, Spain
aarrieta@mondragon.edu

Rui Abreu
FEUP and INESC-ID
Porto, Portugal
rui@computer.org

Wesley K. G. Assunção
DI - Pontifical Catholic University
Rio de Janeiro, Brazil
wassuncao@inf.puc-rio.br

ABSTRACT

Feature localization (FL) is a basic activity in re-engineering legacy systems into software product lines. In this work, we explore the use of the Spectrum-based localization technique for this task. This technique is traditionally used for fault localization but with practical applications in other tasks like the dynamic FL approach that we propose. The ArgoUML SPL benchmark is used as a case study and we compare it with a previous hybrid (static and dynamic) approach from which we reuse the manual and testing execution traces of the features. We conclude that it is feasible and sound to use the Spectrum-based approach providing promising results in the benchmark metrics.

CCS CONCEPTS

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

KEYWORDS

dynamic feature localization, spectrum-based localization, ArgoUML SPL Benchmark

ACM Reference Format:
Gabriela K. Michelon, Bruno Sotto-Mayor, Jabier Martinez, Aitor Arrieta, Rui Abreu, and Wesley K. G. Assunção. 2021. Spectrum-Based Feature Localization: A Case Study using ArgoUML. In *25th ACM International Systems and Software Product Line Conference - Volume A (SPLC '21), September 6–11, 2021, Leicester, United Kingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3461001.3473065

1 INTRODUCTION

The increasing demand for tailored software-intensive products makes companies start considering the transition from monolithic single-purpose systems, or families of systems created with an opportunistic copy-paste-modify approach, to higher levels of systematic reuse for the creation of variants [11]. One of the usually desired transition destinations is Software Product Line (SPL) Engineering and the creation of feature-oriented SPLs [2]. In this context of extractive SPL adoption [10], feature localization (FL) is an important task in the "Detection" phase of the re-engineering process, being the basis for the following phases of "Analysis" and "Transformation" [3]. FL consists of taking some feature information as input and recovering the traces from where this feature is implemented (e.g., source code classes, methods, etc., depending on the desired results granularity). One main separation of categories of FL techniques is based on whether the system under study needs to be executed or not, i.e., dynamic, static, or hybrid (combination of two or more) FL techniques [21].

The ArgoUML SPL benchmark [12] was proposed for comparable results in FL for families of systems. The use of benchmarks in the SPL research field has been acknowledged as an important direction to advance the state of the art [22] and, for this specific benchmark, static FL techniques have been proposed to locate features in sets of variants [6, 14, 16, 17] as well as a hybrid approach [15]. The benchmark [12] provides a feature location ground-truth for eight features within the ArgoUML Java source code based on the source code annotations of a manual extraction [5] using the original ArgoUML. To show that certain automation can be desired, according to an analysis [13], the manual localization took around 4 months per feature and 0.7 months per feature-specific KLoC.

In this work, we focus on dynamic FL and we explore the usage of Spectrum-based localization [24]. Similar to the mentioned hybrid approach [15], we consider only the original scenario, i.e., the single system of the original ArgoUML, aiming to contribute to the state of the art of FL for re-engineering single systems into SPLs. Therefore, our contributions are: (i) a dynamic FL approach using Spectrum-based FL, and (ii) a comparison and discussion of our approach with the previous ones: hybrid [15] and static [14], for the ArgoUML SPL Benchmark [12]. We do not claim novelty but soundness as a new solution to the challenge. Source code, instructions, and results' data are publicly available[1] to reproduce or reuse our solution, or to reason with the results.

SPLC '21, September 6–11, 2021, Leicester, United Kingdom

ACM ISBN 978-1-4503-8469-8/21/09…$15.00
https://doi.org/10.1145/3461001.3473065

[1] https://github.com/jabiercoding/DynamicFL commit 023e56b on 7th June, 2021

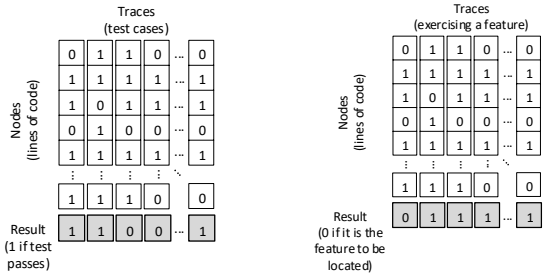
2 SPECTRUM-BASED LOCALIZATION

Spectrum-based localization (SBL) techniques have been mainly used for fault localization [24]. Figure 1a shows an illustration of a spectrum where each trace (column) represents a test case, and the nodes (rows) are the lines of code of the program under study with a 1 when the line has been executed at least once for this given test. The last row is the result of the test with 1 when the test passes and 0 when it fails. Using a spectrum, numerous *ranking metrics* [24] have been proposed to create a ranking of suspiciousness, i.e., trying to point developers to the lines that are potentially creating the issue or at least narrowing the part of the source code that is worthy to examine or debug. Ranking metrics are based on formulas mainly considering the number of executed lines failing or passing the test (e_f and e_p, respectively), and those that are not executed represented as n_f and n_p. Well established ranking metrics are Ochiai [1] or Tarantula [9]:

$$Ochiai = \frac{e_f}{\sqrt{(e_f+n_f))\times(e_f+e_p)}}, Tarantula = \frac{\frac{e_f}{e_f+n_f}}{\frac{e_f}{e_f+n_f}+\frac{e_p}{e_p+n_p}}$$

Other simpler ranking metrics are Wong1 (e_f), Wong2 ($e_f - e_p$), or Hamming ($e_f + n_p$). Each node is assigned by the ranking metric with a continuous value normalized to a range from 0 (apparently completely unrelated) to 1 (apparently faulty). Values in between will have to be analyzed under a certain user-defined *threshold*. Wong et al. [23] present an overview of SBL ranking metrics.

SBL has been used for other purposes beyond fault localization (e.g., FL [20], or program comprehension [4, 19]). In this work, we use it for FL using a Spectrum like the one shown in Figure 1b. Compared to fault localization, our traces are executions of the program purposely exercising a given feature (manual executions or tests that we know are related to a given feature), and the result will be 0 for the traces that belong to the feature that we want to locate, i.e., as if it was the "faulty" code to locate. The obtained ranking when applying one of the ranking metrics will be based on the suspiciousness of each line of code belonging to a feature under analysis.



(a) Fault localization **(b) FL**

Figure 1: Illustration of a spectrum for different purposes.

3 STUDY DESIGN AND EXECUTION

Figure 2 illustrates the design of our study, which we explain next.

Exercising the features. We reused execution traces from Michelon et al. [15]. These traces were made publicly available[2] in files with data about the lines of code that were executed per feature. Two

[2]Dataset with feature execution traces: http://doi.org/10.5281/zenodo.5035177

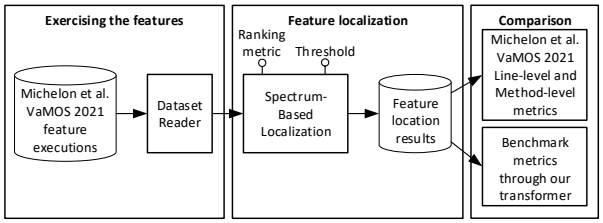


Figure 2: Approach.

scenarios were analyzed: (i) manual exercises and (ii) exercises from existing tests that are related to the features. Videos of the manual feature executions can be watched as part of the supplementary material of that work [15], and the tests (1198 in total) stem from Fischer et al. work [8]. By reusing the feature execution files, we are able to compare the current results with the ones reported by the hybrid approach [15]. Creating our own manual executions or using different tests might introduce a significant bias as, given the nature of dynamic approaches, the obtained results are highly sensitive to this input.

Table 1 presents a characterization of these feature exercises where: ELoC (Exercise LoC) is the total number of unique LoC for the exercise of each feature. FSLoC (Fully-specific LoC) is the number of LoC from ELoC which are part of the feature and never appear in the exercise of other features. FLoC (Feature LoC) is the total number of LoC of the feature as per the benchmark data [12]. Notice that each FSLoC is not necessarily a FLoC as it might be just a "coincidence" that, in the execution traces, no other feature executed that line. EFLoC (Exercise Feature LoC) is the percentage of the total number of LoC of the feature exercised from the execution traces reused from Michelon et al. [15]. EFLoC is the ratio of FLoC to the execution traces of the corresponding feature scenario, and also in

Table 1: Characteristics of the feature exercises.

	Manual			
Feature	ELoC	FSLoC	FLoC [12]	EFLoC [15]
ActivityDiagram	20185	1139	2282	29% (34%)
CollaborationDiagram	19000	1028	1579	34% (37%)
DeploymentDiagram	19980	1590	3147	41% (41%)
SequenceDiagram	18537	1562	5379	30% (30%)
StateDiagram	19534	1288	3917	36% (36%)
UsecaseDiagram	19320	1312	2712	36% (36%)
Average	*19426*	*1320*	*2169*	*34% (36%)*
	Tests			
Feature	ELoC	FSLoC	FLoC [12]	EFLoC [15]
ActivityDiagram	3091	78	2282	1% (2%)
CollaborationDiagram	3095	42	1579	2% (2%)
DeploymentDiagram	2965	22	3147	≈ 0% (0%)
SequenceDiagram	2984	5	5379	≈ 0% (0%)
StateDiagram	3542	482	3917	6% (6%)
UsecaseDiagram	3061	64	2712	1% (1%)
Logging	3009	940	2159	3% (8%)
Cognitive	9119	6071	16319	14% (14%)
Average	*3858*	*963*	*4687*	*3% (4%)*

parentheses, the ratio considering traces of all feature scenarios. On average, only around 36% of the FLoC were executed for the manual executions and 4% for the tests.

Feature localization. We use SBL, and as mentioned before, two aspects are key for SBL techniques, (i) the metric used to rank, and (ii) the threshold to decide when a line of code is suspicious enough to be considered part of a feature. The threshold is needed as the benchmark does not consider probability as part of the metrics computation. In this work, we report the results using several combinations of ranking metrics and threshold values. Concretely, 33 ranking metrics with 10 threshold values for each one (0.1, 0.2, ..., 0.9, 1.0) are used. The goal is not to overfit the technique to this dataset by using those 330 combinations but to provide an overview of what can be the results of SBL. We implemented the SBL technique using an existing Java library[3].

Comparison. In Michelon et al. [15], metrics based on line- and method-level were used due to existing feature location techniques being limited to method-level or do not yield satisfactory results when applied to single systems. Similarly to the feature traces, we reused the code to calculate the line- and method-level metrics from [15] to compare the two FL techniques in equal conditions. In addition, Michelon et al. [15] computed the benchmark metrics (similar to statement-level). However, the execution traces granularity is at the line-level and then the results are also based on lines of code, so it is not straightforward how to transform them to the convention established by the benchmark [12]. This convention mixes class-level, method-level, and refinements of classes and methods (e.g., a "Refinement" tag in a method indicates that some lines in the method correspond to a feature but not the whole method). We implemented our own transformer with a simple implementation consisting of always adding a class-level localization when there is at least one line in the class. By using this naive approach, which does not consider methods or Refinement tags at class- and method-level, we assume a certain loss in recall. Other more sophisticated approaches have been explored, leading to worse results, and thus a better transformer for this specific benchmark will have to be part of further work. Nonetheless, the results are positive.

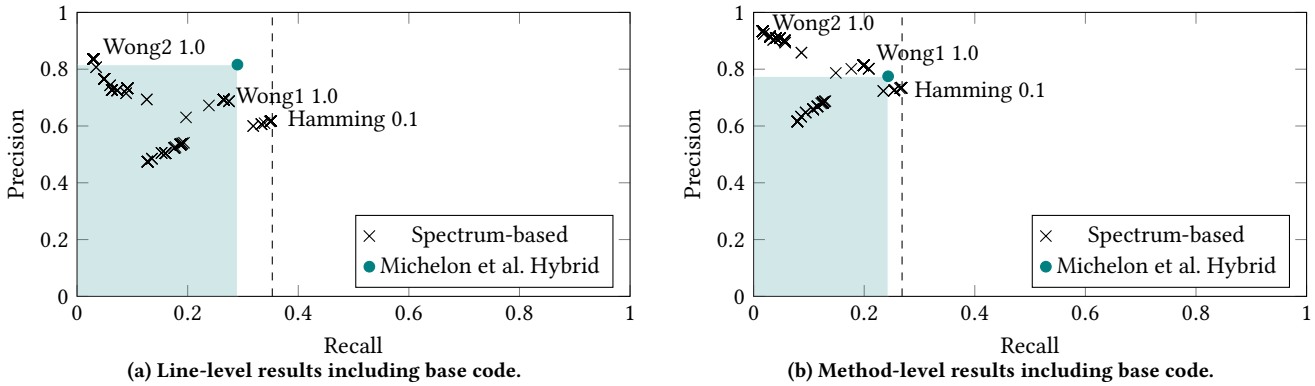
[3]https://github.com/FaKeller/stardust commit 0071fe3 on 27th Jan., 2016

4 RESULTS

Comparison with the Hybrid approach. The static analysis implemented in ECCO tool [7] was used in [15] for refining overlapping traces from the execution traces. In [15], the FL is intended for re-engineering single systems into SPLs for creating variants, which should contain the base plus a/set of feature-specific source code. Thus, line- and method-level metrics reported in [15] consider true positives, not only the feature-specific line and methods but also the source code that is common for all features, i.e., the base of ArgoUML. For the SBL techniques, including the source code of the base in the precision and recall metrics is not potentially beneficial, given that the standard ranking metrics are not designed for identifying the base plus the target feature, but only the target feature. However, the results are also competitive in this comparison.

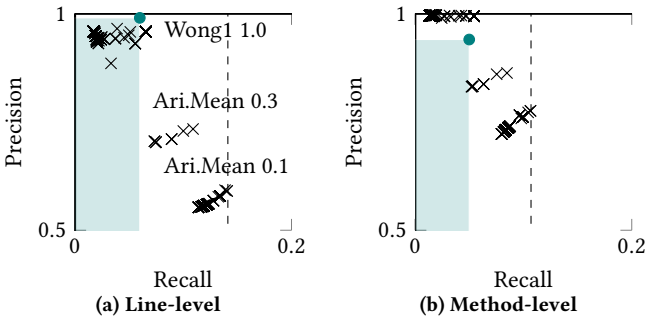
Figures 3 and 4 present the results of the different ranking metrics for the manual and test execution traces using the line- and method-level metrics as defined in [15]. The vertical dashed lines show the boundaries regarding recall given that even if we include all the source code executed through the input execution traces, these execution traces did not include all the feature source code and base code.

In Figure 3, the average of the six diagram features from manual execution from Michelon et al. [15] approach mostly dominates at line-level (the dominated area has a light background). However, ranking metrics in the extremes of precision (e.g., Wong2 1.0) and recall (e.g., Hamming 0.1) are also part of the pareto front (i.e., non-dominated solutions). At the method-level, SBL techniques get closer to [15] even if they were not specifically designed to locate the base source code. Figure 4 shows the results for the tests execution traces for the eight features. At line-level, [15] dominates in terms of precision, but several SBL techniques are close to it, such as Wong1 1.0, which also has a higher recall. The highest precision for SBL is obtained with Wong3 1.0. At the method-level, we have solutions that slightly outperform [15] both in precision and recall, such as Wong1 1.0, Ochiai 0.1 to 0.3, or Tarantula 0.1 to 0.5.



(a) Line-level results including base code. **(b) Method-level results including base code.**

Figure 3: Manual: Each point of "Spectrum-based" is a different ranking metric and threshold. Comparison with Michelon et al. Hybrid 2021 [15] using their precision and recall metrics that considered as true positives the retrieved source code from the feature-specific plus the base source code.



(a) Line-level **(b) Method-level**

Figure 4: Similar to Figure 3 but using tests feature exercises.

Spectrum-based localization results. Figure 5 shows the results with our own transformer to benchmark convention for the manual and test exercises. For comparison, we include the results reported by the hybrid approach [15] using the benchmark convention, and we additionally include Michelon et al. Static [14] results, which is a static approach reasoning on variants overlap. The latter is successful in scenarios with an increasing number of variants and it also considers feature interactions. From this work, we include only the results in the benchmark scenario with only one variant and using the average of the considered features.

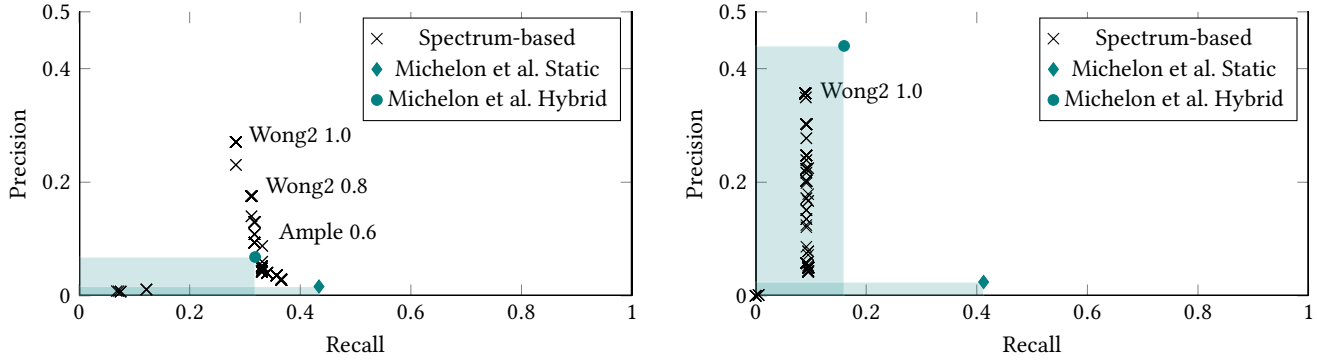
In Figure 5a, the results with the highest precision, in which the manual exercises are obtained with the same values through different ranking metrics (Wong2 0.9 to 1, Ochiai 0.8 to 1, Ochiai2 0.7 to 1, Tarantula 0.9 to 1, Hamming 0.9 to 1, Euclid 1, ArithmeticMean 0.9 to 1, HarmonicMean 0.9 to 1, Anderberg 0.4 to 1, among others). Those are also the ones that obtained the highest F1 (harmonic mean of the precision and recall). For recall, not surprisingly, the highest values are obtained with low thresholds with different techniques (e.g., Wong2 0.1, Hamming 0.1, ArithmeticMean 0.1 to 0.4, among others). We can observe how Michelon et al. Static is in the Pareto front because of its recall value. Michelon et al. Hybrid got lower values in both precision and recall compared to SBL ranking metrics and thresholds (Ample 0.5 to 0.6). Figure 5b shows the results for the testing exercises. In this case, Michelon et al. Hybrid dominates the SBL technique in both dimensions. The ones closer to Michelon et al. Hybrid are Wong2, Tarantula, Hamming, ArithmeticMean 0.9 to 1, Anderberg 0.4 to 1, Ochiai 0.8 to 1, and Ochiai2 0.7 to 1.

Regarding the performance, we computed the average of 30 runs using a laptop model Latitude 5480, Intel(R) Core(TM) i5-7300U processor (2.60GHz), running the Windows 10 operating system. The Dataset Reader takes around 18 seconds as it needs to parse all the execution traces' files. After that, for the creation of the spectrum, the computation of the ranks using the ranking metric for the six features, getting the results under the defined threshold (which we forced to be 0.1 for all the runs in the performed experiment as the worst case given that most of the nodes will be retrieved this way), took less than a second or around a second. Thus, scalability regarding performance does not seem to be a problem for SBL techniques. The runtime performance of the static approach used in [15], after having the execution traces, took on average ≈ 19 seconds per feature. We cannot compare the runtime of our approach and with [15] because the laptops' model is different, but both performed in a reasonable time. As a drawback, we should remember the time needed to exercise the features, a required step in any dynamic FL technique, which can add several minutes for preparation and obtaining the execution traces.

5 DISCUSSION AND CONCLUSIONS

As already acknowledged by the fault localization community, there is no optimal ranking metric in practice [18, 23]. We provide the results of the ones that obtained the highest precision and recall with their corresponding thresholds as candidate suggestions, e.g., Wong2 1.0 seems appropriate if precision is to be optimized. The results also depend on how one "exercises" the features. As we can see in Table 1, the coverage of the features LoC from traces was obtained by exercising the features in Michelon et al. [15] is relatively low. According to these numbers, we can understand how much recall is affected by the execution traces and how much dynamic FL could be improved to reach higher recall.

We explored the use of the Spectrum-based Localization technique for FL and applied it in the ArgoUML SPL Benchmark. Standard spectrum-based ranking metrics obtain competitive results using existing execution traces compared to a previous hybrid approach. As further work, we consider calculating feature interaction locations and use other benchmark scenarios with several variants to refine the results through hybrid approaches. Also, techniques combining ranking metrics through a machine learning phase with already located features can be explored (e.g., [25]).



(a) Manual: Benchmark results for the six features. **(b) Tests: Benchmark results for the eight features.**

Figure 5: Feature exercises: using manual (six diagram features) and tests (all eight features) traces.

REFERENCES

[1] Rui Abreu, Peter Zoeteweij, and Arjan J. C. van Gemund. 2007. On the Accuracy of Spectrum-Based Fault Localization. In *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION (TAICPART-MUTATION '07)*. IEEE, USA, 89–98.

[2] Sven Apel, Don S. Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-Oriented Software Product Lines - Concepts and Implementation.* Springer, New York, USA.

[3] 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.

[4] Bruno Castro, Alexandre Perez, and Rui Abreu. 2019. Pangolin: an SFL-based toolset for feature localization. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, IEEE, New York, USA, 1130–1133.

[5] 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, New York, USA, 191–200.

[6] Daniel Cruz, Eduardo Figueiredo, and Jabier Martinez. 2019. 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, VAMOS 2019, Leuven, Belgium, February 06-08, 2019*. ACM, New York, USA, 16:1–16:10.

[7] Stefan Fischer, Lukas Linsbauer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. 2014. Enhancing Clone-and-Own with Systematic Reuse for Developing Software Variants. In *30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014*. IEEE, New York, USA, 391–400.

[8] Stefan Fischer, Gabriela Karoline Michelon, Rudolf Ramler, Lukas Linsbauer, and Alexander Egyed. 2020. Automated test reuse for highly configurable software. *Empir. Softw. Eng.* 25, 6 (2020), 5295–5332.

[9] James A. Jones, Mary Jean Harrold, and John Stasko. 2002. Visualization of Test Information to Assist Fault Localization. In *Proceedings of the 24th International Conference on Software Engineering (ICSE '02)*. Association for Computing Machinery, New York, NY, USA, 467–477.

[10] 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, New York, USA, 282–293.

[11] Jacob Krüger and Thorsten Berger. 2020. An empirical analysis of the costs of clone- and platform-oriented software reuse. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, New York, USA, 432–444.

[12] Jabier Martinez, Nicolas Ordoñez, Xhevahire Tërnava, Tewfik Ziadi, Jairo Aponte, Eduardo Figueiredo, and Marco Tulio Valente. 2018. Feature Location Benchmark with argoUML SPL. In *22nd International Systems and Software Product Line Conference - Volume 1 (SPLC'18)*. ACM, New York, USA, 257–263.

[13] 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 *SPLC '20: 24th ACM International Systems and Software Product Line Conference, Montreal, Quebec, Canada, October 19-23, 2020, Volume A*. ACM, New York, USA, 6:1–6:6.

[14] Gabriela Karoline Michelon, Lukas Linsbauer, Wesley K. G. Assunção, and Alexander Egyed. 2019. Comparison-based feature location in ArgoUML variants. In *Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume A, Paris, France, September 9-13, 2019*. ACM, New York, USA, 17:1–17:5.

[15] Gabriela K. Michelon, Lukas Linsbauer, Wesley K.G. Assunção, Stefan Fischer, and Alexander Egyed. 2021. A Hybrid Feature Location Technique for Re-EngineeringSingle Systems into Software Product Lines. In *15th International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS'21)*. Association for Computing Machinery, New York, NY, USA, Article 11, 9 pages.

[16] Johann Mortara, Xhevahire Tërnava, and Philippe Collet. 2020. Mapping features to automatically identified object-oriented variability implementations: the case of ArgoUML-SPL. In *VaMoS '20: 14th International Working Conference on Variability Modelling of Software-Intensive Systems, Magdeburg Germany, February 5-7, 2020*. ACM, New York, USA, 20:1–20:9.

[17] Richard Müller and Ulrich W. Eisenecker. 2019. A graph-based feature location approach using set theory. In *Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume A, Paris, France, September 9-13, 2019*. ACM, New York, USA, 16:1–16:5.

[18] Spencer Pearson, José Campos, René Just, Gordon Fraser, Rui Abreu, Michael D Ernst, Deric Pang, and Benjamin Keller. 2017. Evaluating and improving fault localization. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, IEEE, New York, USA, 609–620.

[19] Alexandre Perez and Rui Abreu. 2014. A diagnosis-based approach to software comprehension. In *Proceedings of the 22nd International Conference on Program Comprehension*. ACM, New York, USA, 37–47.

[20] Alexandre Perez and Rui Abreu. 2016. Framing program comprehension as fault localization. *Journal of Software: Evolution and Process* 28, 10 (2016), 840–862.

[21] Julia Rubin and Marsha Chechik. 2013. A Survey of Feature Location Techniques. In *Domain Engineering, Product Lines, Languages, and Conceptual Models*, Iris Reinhartz-Berger, Arnon Sturm, Tony Clark, Sholom Cohen, and Jorn Bettin (Eds.). Springer, New York, USA, 29–58.

[22] 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, New York, USA, 26:1–26:12.

[23] W. Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. 2016. A Survey on Software Fault Localization. *IEEE* 42 (08 2016), 1–1. https://doi.org/10.1109/TSE.2016.2521368

[24] W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. 2016. A Survey on Software Fault Localization. *IEEE Transactions on Software Engineering* 42, 8 (2016), 707–740.

[25] Jifeng Xuan and Martin Monperrus. 2014. Learning to Combine Multiple Ranking Metrics for Fault Localization. In *30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014*. IEEE, New York, USA, 191–200.