VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

NGUYEN THU TRANG

Automated Localization and Repair for Variability Faults in Software Product Lines

DOCTOR OF PHILOSOPHY DISSERTATION

Major: Software Engineering

Hanoi - 2024

Abstract

Software Product Line (SPL) systems are becoming popular and widely employed to develop large industrial projects. However, their inherent variability characteristics pose extreme challenges for assuring the quality of these systems. Although automated debugging in single-system engineering has been studied in-depth, debugging SPL systems remains mostly unexplored. In practice, debugging activities in SPL systems are often performed manually in an ad-hoc manner. This dissertation sheds light on the automated debugging SPL systems by focusing on three fundamental tasks, including *false-passing* product detection, variability fault localization, and variability fault repair.

First, *this dissertation aims to improve the reliability of the test results by detecting false-passing products in SPL systems failed by variability bugs.* Given a set of tested products of an SPL system, our approach, CLAP, collects failure indications in failing products based on their implementation and test quality. For a passing product, CLAP evaluates these indications, and the stronger the indications, the more likely the product is *false-passing*. Specifically, the possibility of the product being *false-passing* is evaluated based on if it has a large number of statements that are highly suspicious in the failing products and if its test suite is lower quality compared to the failing products' test suites.

Second, *this dissertation presents* VARCOP*, a novel and effective variability fault localization approach.* For an SPL system failed by variability bugs, VARCOP isolates suspicious code statements by analyzing the overall test results of the sampled products and their source code. The isolated suspicious statements are the statements related to the interaction among the features that are necessary for the visibility of the bugs in the system. In VARCOP, the suspiciousness of each isolated statement is assessed based on both the overall test results of the products containing the statement as well as the detailed results of the test cases executed by the statement in these products.

Third, *this dissertation proposes two approaches, product-based and system-based, to repair the variability bugs in an SPL system to fix the failures of the failing products and not to break the correct behaviors of the passing products.* For the product-based approach, each failing product is fixed individually, and the obtained patches are then propagated and validated on the other products of the system. For the system-based approach, all the products are repaired simultaneously. The patches are generated and validated by all the sampled products of the system in each repair iteration. Moreover, to improve the repair performance of both approaches, this Dissertation also introduces several heuristic rules for effectively and efficiently deciding where to fix (*navigating modification points*) and how to fix (*selecting suitable modifications*). These heuristic rules use intermediate validation results of the repaired programs as feedback to refine the fault localization results and

evaluate the suitability of the modifications before actually applying and validating them by test execution.

To evaluate our approaches, this dissertation conducted several experiments on a large public dataset of buggy SPL systems. The experimental results show that CLAP can effectively detect *false-passing* and *true-passing* products with an average accuracy of more than 90%. Especially, the precision of *false-passing* product detection by CLAP is up to 96%. This means among ten products predicted as *false-passing* products, more than nine products are precisely detected.

For variability fault localization, VARCOP significantly improves two state-of-the-art techniques by 33% and 50% in ranking the incorrect statements in the systems containing a single bug each. In about two-thirds of the cases, VARCOP correctly ranks the buggy statements at the top-3 positions in the ranked lists. For the cases containing multiple bugs, VARCOP outperforms the state-of-the-art approaches two times and ten times in the proportion of bugs localized at the top-1 positions.

Furthermore, for repairing variability faults, our results show that the product-based approach is around 20 times better than the system-based approach in the number of correct fixes. Notably, the heuristic rules could improve the performance of both approaches by increasing of 30-150% the number of correct fixes and decreasing of 30-50% the number of attempted modification operations.

Keywords: *Software product line*, *variability fault*, *coincidential correctness*, *fault localization*, *automated program repair*

Chapter 1

Introduction

Nowadays, Software Product Line (SPL) systems (or Configurable Systems, in general) are becoming popular and widely employed to develop large industrial projects. An SPL system is a product family containing a set of products sharing a common code base. Each product is identified by the selected features. In other words, a project adopting the SPL methodology can tailor its functional and nonfunctional properties to the requirements of users. This has been done using a very large number of *options* which are used to control different *features* additional to the *core software*. A set of *selections* of all the features (*configurations*) defines a program *variant* (*product*). For example, Linux Kernel supports thousands of features controlled by +12K compile-time options that can be configured to generate specific kernel *variants* for billions of scenarios.

The variability of SPL system creates many benefits in software developments. However, this characteristic also challenges Quality Assurance (QA). In comparison with the traditional single-system engineering (aka. non-configurable system), fault detection, localization, and repair through testing in SPL systems are more problematic, as a bug can be *variable* (so-called *variability bug*), which can only be exposed under *certain* combinations of the system features. In particular, there exists a set of features that must be selected to be *on* and *off together* to necessarily reveal the bug. Due to the presence/absence of the *interaction* among the features in such set, the buggy statements behave differently in the products where these features are on and off together or not. Hence, *the incorrect statements can only expose their bugginess in certain products, yet cannot in others.* Specially in an SPL system, variability bugs only cause failures in certain products, while the others still pass all their tests.

Although automated debugging in single-system engineering has been studied in-depth, debugging SPL systems still remains mostly unexplored. This dissertation focuses on automated debugging SPL systems in three main tasks: detecting *false-passing* products, localizing variability faults, and repairing such faults in SPL systems. Our proposed process for automated debugging SPL system is shown in the bottom half of Figure 1.1. Due to the dynamic nature of SPL systems, with numerous combinations and interactions

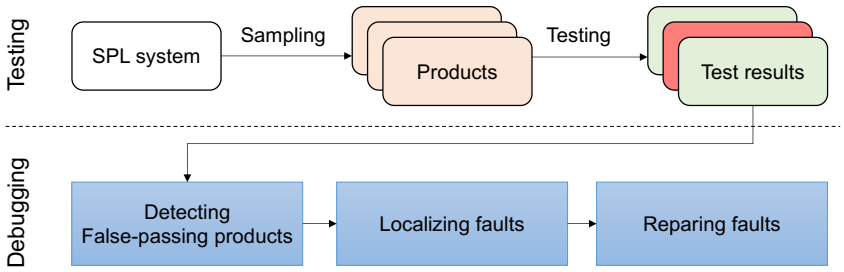

Figure 1.1: The proposed debugging process of SPL systems

among features, it amplifies the difficulties of debugging the buggy SPL systems.

This dissertation aims to propose approaches for automatically debugging SPL systems failed by variability bugs. To improve the reliability of the test results, this dissertation propose CLAP, an approach for detecting *false-passing* products. Next, this dissertation presents VARCOP, a novel FL approach specialized for variability faults of SPL systems. Finally, this dissertation introduces two *product-based* and *system-based* approaches to automatically repairing variability faults.

First, *this dissertation introduces* CLAP*, an approach for* ***detecting false-passing products*** *of buggy SPL systems.* The intuition of our approach is that for a buggy SPL system, the sampled products can share some common functionalities. If the unexpected behaviors of the functionalities are revealed by the tests in some (failing) products, the other products having similar functionalities are likely to be caused failures by those unexpected behaviors. In CLAP, *false-passing* products can be detected based on the *failure indications* which are collected by reviewing the *implementation* and *test quality* of the failing products. To evaluate the possibility that a passing product is a *false-passing* one, this dissertation proposes several *measurable attributes* to assess the strength of these failure indications in the product. The stronger indications, the more likely the product is *false-passing*.

Second, *this dissertation proposes* VARCOP*, a novel approach for* ***localizing variability bugs***. Our key ideas in VARCOP is that variability bugs are localized based on (i) the interaction among the features which are necessary to reveal the bugs, and (ii) the bugginess exposure which is reflected via both overall test results at the product-level and the detailed test results at the test case-level.

Third, *this dissertation proposes two approaches, product-based and system-based, for* ***automatically repairing variability bugs*** *of the SPL systems.* Furthermore, *this dis-*

sertation also introduces several heuristic rules for improving the performance of the two approaches in repairing buggy SPL systems. this dissertation starts from the observation that, in order to effectively and efficiently fix a bug, an APR tool must correctly decide (i) where to fix (*navigating modification points*) and (ii) how to fix (*selecting suitable modifications*). Our heuristic rules focus on enhancing the accuracy of these tasks by leveraging intermediate validation results of the repair process.

In summary, this dissertation makes the following main contributions:

- The formulation of the *false-passing* product detection problem in SPL systems and a large benchmark for evaluating *false-passing* product detection techniques.
- CLAP: an effective approach to detect *false-passing* products in SPL systems and mitigate their negative impact on variability fault localization performance. CLAP's implementation can be found at: *https://ttrangnguyen.github.io/CLAP/*.
- A formulation of *Buggy Partial Configuration (Buggy PC)* where the interaction among the features in the *Buggy PC* is the root cause of the failures caused by variability bugs in SPL systems.
- VARCOP: A novel effective approach/tool to localize variability bugs in SPL systems. VARCOP's implementation can be found at: *https://ttrangnguyen.github.io/VARCOP/*.
- Heuristic rules for navigating modification points and selecting suitable modifications to improve the performance of APR tools.
- The product-based and system-based approaches for repairing variability bugs in the source code of SPL systems. The implementation the proposed approaches can be found at: *https://github.com/ttrangnguyen/SPLRepair*.
- Extensive experimental evaluations showing the performance of the approaches.

The remainder of this dissertation is organized as follows. Chapter 2 introduces the background and reviews the related studies. Our proposed approach for detecting *false-passing* products is introduced in Chapter 3. The proposed approach for localizing variability faults is described in Chapter 4. Chapter 5 shows our two product-based and system-based approaches for repairing variability faults in SPL systems. Finally, Chapter 6 summarizes and concludes this dissertation.

Chapter 2

Background and Literature Review

This chapter introduces background and the concepts which are used in the following sections of the dissertation. First, this chapter introduces the key concepts of the SPL systems, the main testing methodologies, FL and APR techniques. Next, this chapter reviews the related works. Finally, this chapter introduces the popular benchmarks for evaluating testing and debugging approaches of the SPL systems.

For SPL engineering, instead of analyzing and implementing a single product each, developers target a *variety of products* that are *similar* but *not identical*. For this purpose, the development process of SPL systems considers two important factors: *variability* and *reuse*. In the overview process of developing an SPL system, there are two main processes: Domain engineering and Application engineering. Overall, an SPL is a product family that consists of a set of products sharing a common code base. These products distinguish from the others in terms of their *features*.

Unlike non-configurable code, bugs in SPL systems can be *variable* and only cause the failures in certain products.

Definition 2.1 ***(Variability Bug).*** *Given a buggy SPL system $\mathfrak{S}$ and a set of products of the system, P, which is sampled for testing, a variability bug is an incorrect code statement of $\mathfrak{S}$ that causes the unexpected behaviors (failures) in a set of products which is a non-empty strict subset of P.*

In order to guarantee the quality of software, testing is the most popular method. Although testing could help discover faults due to the observed erroneous behaviors, finding and fixing them is an entirely different matter. Fault localization, identifying the locations of program faults, is critical in program debugging, yet widely recognized as a tedious, time-consuming, and prohibitively expensive activity. For effective and efficient fault finding, multiple FL approaches for partially or fully automated figuring out the positions of the faults have been proposed. These FL approaches are often categorized into eight groups according to their techniques, including slice-based, spectrum-based, statistics-based, program state-based, machine learning-based, data mining-based, model-based,

and miscellaneous techniques.

Amongst these techniques, Spectrum-Based Fault Localization (SBFL) is considered the most prominent due to its lightweight, efficiency, and effectiveness. Specifically, SBFL is a dynamic program analysis technique that leverages the testing information (i.e., test results and code coverage) for measuring the suspiciousness scores of the code components such as statements, basic blocks, methods, etc. The intuition is that, in a program, the more failed tests and the fewer passed tests executed by a code component, the more suspicious the code component is. The component with the higher suspiciousness score is more likely to be buggy.

To reduce the cost of software maintenance, multiple APR techniques have been proposed in the past. The most popular APR approach is *test-suite-based program repair*, such as GenProg, Nopol, and Cardumen, which use test suites as the specification of the program's expected behaviors. For repairing a program failed by at least one test, these APR approaches attempt to generate candidate patches. Then, the available test cases are used to check whether the generated patches can fix the program.

Chapter 3

False-passing Product Detection

Thorough testing is generally required to guarantee the quality of programs. However, it is often hard, tedious, and time-consuming to conduct thorough testing in practice. Various bugs could be neglected by the test suites since it is extremely difficult to cover all the programs' behaviors. Moreover, there are kinds of bugs which are challenging to be detected due to their difficulties in infecting the program states and propagating their incorrectness to the outputs. Consequently, even when the defects is reached, there are test cases that still obtain correct outputs, i.e., *coincidentally correct/passed* tests. Indeed, coincidental correctness is a prevalent problem in software testing, and this phenomenon causes a severely negative impact on FL performance.

Similar to testing in non-configurable code, the coincidental correctness phenomenon also happens in SPL systems and causes difficulties in finding faults in these systems. Specifically, for an SPL system, a set of products is often sampled for testing. Each sampled product is composed of a set of features of the system and tested individually by its test suite as a singleton program. For a buggy SPL system, the bugs could be in one or more products. Ideally, if a product contains bugs (*buggy products*), the bugs should be revealed by its test suite. In other words, there should be at least a failed test after testing. However, if the test suite of a buggy product is ineffective in detecting the bugs, the product's overall test result will be passing. For instance, the suite does not cover the product's buggy statements or those test cases could reach the buggy statements but could not propagate the incorrectness to the outputs, the product still passes all the tests. Such a passing product is indeed a buggy product, yet incorrectly considered as passing. That passing product is namely a *false-passing* product. Due to the unreliability of test results, these *false-passing* products might negatively impact the FL performance. In particular, the performance of two main SBFL strategies in SPL systems, *product-based* and *test case-based*, is affected.

First, the *product-based* FL techniques evaluate the suspiciousness of a statement in a buggy SPL system based on the appearance of the statement in failing and/or passing products. Specially, the key idea to find bugs in an SPL system is that a statement which

is included in more failing products and fewer passing products is more likely to be buggy than the other statements of the system. Misleadingly counting a buggy product as a passing product incorrectly decreases the number of failing products and increases the number of passing products containing the buggy statement. Consequently, the buggy statement is considered less suspicious than it should be.

Second, the *test case-based* FL techniques measure the suspicious scores of the statements based on the numbers of failed and passed tests executed by them. Indeed, *false-passing* products could lead to under-counting the number of failed tests and over-counting the number of passed tests executed by the buggy statements. The reason is that *false-passing* products contain bugs, but there is no failed test. In these *false-passing* products, the buggy statements are not executed by any test, or they are reached by several tests, yet those tests *coincidentally* passed. Both low coverage test suite and coincidentally passed tests can cause inaccurate evaluation for the buggy statements.

This chapter introduces CLAP, a novel *false-passing* product detection approach for SPL systems that failed by variability bugs. The intuition of our approach is that for a buggy SPL system, the sampled products can share some common functionalities. If the unexpected behaviors of the functionalities are revealed by the tests in some (failing) products, the other products having similar functionalities are likely to be caused failures by those unexpected behaviors. In CLAP, *false-passing* products can be detected based on the *failure indications* which are collected by reviewing the *implementation* and *test quality* of the failing products. To evaluate the possibility that a passing product is a *false-passing* one, this chapter proposes several *measurable attributes* to assess the strength of these failure indications in the product. The stronger indications, the more likely the product is *false-passing*.

The proposed attributes are belonged to two aspects: *product implementation* (products' source code) and *test quality* (the adequacy and the effectiveness of test suites). The attributes regarding *product implementation* reflect the possibility that the passing product contains bugs. Intuitively, if the product has more (suspicious) statements executing the tests failed in the failing products of the system, the product is more likely to contain bugs. For the *test quality* of the product, the *test adequacy* reflects how its suite covers the product's code elements such as statements, branches, or paths. A low-coverage test suite could be unable to cover the incorrect elements in the buggy product. Hence, the product with a lower-coverage test suite is more likely to be *false-passing*. Meanwhile, the

test effectiveness reflects how intensively the test suite verifies the product's behaviors and its ability to explore the product's (in)correctness. The intuition is that if the product is checked by a test suite which is less effective, its overall test result is less reliable. Then, the product is more likely to be a *false-passing* one.

Furthermore, this dissertation discusses several strategies to mitigate the negative impact of *false-passing* products on the performance of the FL approaches. Since the negative impact is mainly caused by the unreliability of the test results, our goal is to improve the *reliability* of the test results by enhancing the test quality based on the failure indications. Moreover, the reliability of test results could also be improved by disregarding the unreliable test results at either product-level or test case-level.

This dissertation conducted several experiments on a large dataset of variability bugs which contains 823 buggy versions of six widely-used SPL systems. Totally, there are 14,191 *false-passing* products and 22,555 *true-passing* products. Our results show that CLAP achieves more than 90% *Accuracy* in detecting *false-passing* and *true-passing* products. This dissertation also evaluates the capability of CLAP in mitigating the negative impact of *false-passing* products on the FL performance. The experimental result shows that CLAP can greatly mitigate the negative impact of *false-passing* products on localizing variability bugs and help developers find bugs much faster.

Chapter 4

Variability Fault Localization

The variability that is inherent to SPL systems challenges QA activities. In comparison with the single-system engineering, fault detection and localization through testing in SPL systems are more problematic, as a bug can be *variable*, which can only be exposed under *some* combinations of the system features. Specially, there exists a set of the features that must be selected to be on and off together to necessarily reveal the bug. Due to the presence/absence of the *interaction* among the features in such set, the buggy statements behave differently in the products where these features are on and off together or not. Hence, *the incorrect statements can only expose their bugginess in some particular products, yet cannot in the others.* Specially, in an SPL system, variability bugs only cause failures in certain products, and the others still pass all their tests. This variability property causes considerable difficulties for localizing this kind of bugs in SPL systems.

Despite the importance of *variability fault localization*, the existing FL approaches are not designed for this kind of bugs. These techniques are specialized for finding bugs in a particular product. For instance, to isolate the bugs causing failures in multiple products of a single SPL system, the slice-based methods could be used to identify all the failure-related slices for each product independently of others. Consequently, there are multiple sets of (large numbers of) isolated statements that need to be examined to find the bugs. This makes the slice-based methods become impractical in SPL systems.

In addition, the state-of-the-art technique, SBFL can be used to calculate the suspiciousness scores of code statements based on the test information (i.e., program spectra) of each product of the system separately. For each product, it produces a ranked list of suspicious statements. As a result, there might be multiple ranked lists produced for a single SPL system which is failed by variability bugs. From these multiple lists, developers cannot determine a starting point to diagnose the root causes of the failures. Hence, it is inefficient to find variability bugs by using SBFL to rank suspicious statements in multiple variants separately.

Another method to apply SBFL for localizing variability bugs in an SPL system is that one can treat the whole system as a single program. This means that the mechanism control-

ling the presence/absence of the features in the system (e.g., the preprocessor directives `#ifdef`) would be considered as the corresponding conditional `if-then` statements during the localization process. By this adaptation, a single ranked list of the statements for variability bugs can be produced according to the suspiciousness of each statement. Note that, this dissertation considers the product-based testing. Specially, each product is considered to be tested individually with its own test set. Additionally, a test, which is designed to test a feature in domain engineering, is concretized to multiple test cases according to products' requirements in application engineering. Using this adaptation, the suspiciousness of the statement is measured based on the total numbers of the passed and failed tests executed by it in all the tested products. Meanwhile, the characteristics including the interactions between system features and the variability of failures among products are also useful to isolate and localize variability bugs in SPL systems. However, these kinds of important information are not utilized in the existing approaches.

This chapter proposes VarCop, a novel fault localization approach for variability bugs. Our key ideas in VarCop is that variability bugs are localized based on (i) the interaction among the features which are necessary to reveal the bugs, and (ii) the bugginess exposure which is reflected via both the overall test results of products and the detailed result of each test case in the products.

For a buggy SPL system, VarCop detects sets of the features which need to be selected on/off together to make the system fail by analyzing the overall test results (i.e., the state of passing all tests or failing at least one test) of the products. This dissertation calls each of these sets of the feature selections a *Buggy **P**artial **C**onfiguration (Buggy PC)*. Then, VarCop analyzes the interaction among the features in these *Buggy PCs* to isolate the statements which are suspicious. In VarCop, the suspiciousness of each isolated statement is assessed based on two criteria. The first criterion is based on the overall test results of the products containing the statement. By this criterion, the more failing products and the fewer passing products where the statement appears, the more suspicious the statement is. Meanwhile, the second one is assessed based on the suspiciousness of the statement in the failing products which contain it. Specially, in each failing product, the statement's suspiciousness is measured based on the detailed results of the products' test cases. The idea is that if the statement is more suspicious in the failing products based on their detailed test results, the statement is also more likely to be buggy in the whole system.

This chapter conducted experiments to evaluate VarCop in both single-bug and multiple-bug settings on a dataset of 1,570 versions (cases) containing variability bug(s). The performance of VarCop is compared with the state-of-the-art approaches including (SBFL), the combination of the slicing method and SBFL (S-SBFL), and Arrieta et al. using 30 most popular SBFL ranking metrics. The experimental results show that VarCop significantly outperformed the baselines in all the studied metrics.

Chapter 5

Automated Variability Fault Repair

In practice, bugs are an inevitable problem in software programs. Developers often need to spend about 50% of their time on addressing software bugs. Detecting and fixing bugs in SPL systems could be very complicated due to their variability characteristics. Echeverría et al. conducted an empirical study to evaluate engineers' behaviors in fixing errors and propagating the fixes to other products in an industrial SPL system. They showed that fixing buggy SPL systems is challenging, especially for large systems. Indeed, in an SPL system, each product is composed of a different set of features. Due to the interaction of different features, a bug in an SPL system could manifest itself in some products of the system but not in others, so called *variability bugs*. In order to fix variability bugs, the program repair approaches need to find patches which not only work for one product but also for all the products of the system, i.e., the program repair approaches need to fix the incorrect behaviors of all *failing products*, and do not break the correct behaviors of the *passing products*.

To reduce the cost of software maintenance and alleviate the heavy burden of manually debugging activities, multiple automatic program repair (APR) approaches have been proposed in recent decades. These approaches employ different techniques to automatically (i.e., without human intervention) synthesize patches that eliminate program faults and obtain promising results. However, these approaches focus on fixing bugs in a single non-configurable system.

In the context of SPL systems, there are several studies attempting to deal with the variability bugs at different levels, such as model or configuration. For example, Arcaini et al. attempt to fix bugs in the variability models. Weiss et al. repair misconfigurations of the SPL systems. However, repairing variability bugs at the source code level still remains unexplored.

This research aims to make the first attempt at automatically repairing variability bugs in the source code of SPL systems. This chapter proposes two approaches, *product-based* and *system-based*, for repairing buggy SPL systems at the source code level. For the *product-based approach* ($ProdBased_{basic}$), each failing product of the system is repaired

individually, and then the obtained patches, which cause the product under repair to pass all its tests, are propagated and validated on the other products of the system. For the *system-based approach* ($SysBased_{basic}$), instead of repairing one individual product at a time, all the products are considered for repairing simultaneously. Specifically, the patches are generated and then validated by all the sampled products of the system in each repair iteration. For both approaches, the valid patches are the patches causing all the available tests of all the sampled products of the system to pass.

Furthermore, this chapter also introduce several *heuristic rules* for improving the performance of the two approaches in repairing buggy SPL systems. The heuristic rules are started from the observation that, in order to effectively and efficiently fix a bug, an APR tool must correctly decide (i) where to fix (*navigating modification points*) and (ii) how to fix (*selecting suitable modifications*). Our heuristic rules focus on enhancing the accuracy of these tasks by leveraging intermediate validation results of the repair process.

For *navigating modification points*, APR tools often utilize the *suspiciousness scores*, which refer to the probability of the code elements to be faulty. These scores are often calculated once for all before the repair process by FL techniques such as spectrum-based or mutation-based FL. However, a lot of additional information can be obtained during the repairing process, such as the modified programs' validation results. Such information can provide valuable feedback for continuously refining the navigation of the modification points. Therefore, in this work, besides suspiciousness scores, the *fixing scores* of the modification points, which refer to the ability to fix the program by modifying the source code of the corresponding points, are used for navigating modification points in each repair iteration. The fixing scores are continuously measured and updated according to the intermediate validation results of the modified programs. The intuition is that *if modifying the source code at a modification point mp causes (some of) the initial failed test(s) to be passed, mp could be the correct position of the fault or have relations with the fault.* Otherwise, modifying its source code cannot change the results of the failed tests. The modification point with a high fixing score and high suspiciousness score should be prioritized to attempt in each subsequent repair iteration.

After a modification point is selected, APR tools generate and *select suitable modifications* for that point and evaluate them by executing tests. This dynamic validation is time-consuming and costs a large amount of resources. In order to mitigate the wasted time of validating incorrect modifications, this dissertation introduces *modification suitability*

measurement for lightweight evaluating and quickly eliminating unsuitable modifications. The suitability of a modification at position *mp* is evaluated by the similarity of that modification with the original source code and with the previous attempted modifications at *mp*. The intuition is that *the correct modification at mp is often similar to its original code and the other successful modifications at this point, while the modifications similar to the failed modifications are often incorrect.* Thus, the more similar a modification is to the original code and to the successful modifications, and the less similar it is to the failed modifications, then the more suitable that modification is for attempting at *mp*.

These heuristic rules are embedded on both product-based and system-based approaches, and the enhanced versions are called $ProdBased_{enhanced}$ and $SysBased_{enhanced}$.

To evaluate the proposed approaches, this dissertation conducts several experiments with $ProdBased_{basic}$, $SysBased_{basic}$, $ProdBased_{enhanced}$, and $SysBased_{enhanced}$ on a dataset of 318 buggy versions of 5 SPL systems (i.e., 318 variability bugs). The experimental results show that the product-based approach is considerably better than the system-based approach by **12 to 30 times** in the number of plausible fixes and about **20 times** in the number of correct fixes. Interestingly, our heuristics could help to boost the performance of both product-based and system-based approaches by up to **200%**. For instance, by adopting the APR tool Cardumen, $ProdBased_{basic}$ and $SysBased_{basic}$ can **correctly fix 13 and 0 systems** respectively, while $ProdBased_{enhanced}$ and $SysBased_{enhanced}$ **correctly fix 40 and 1 systems** respectively. Moreover, the repair performance could be negatively impacted by FL tools since the modification points are selected based on FL results which are often imperfect. To mitigate the impact of the third-party FL tool, this dissertation assesses the effectiveness of the repair approaches if correct FL results are provided. In this experiment, the results show that the product-based approach is better than the system-based approach about **3 times** in effectiveness and **9 times** in efficiency. In addition, the proposed heuristic rules help to increase **30-150%** the number of correct fixes and decrease **30-70%** the number of attempted modification operations of the corresponding basic approaches.

Chapter 6

Conclusion

The contribution of the dissertation: SPL systems have gained momentum in the software industry. By the configurable mechanism and reusable parts, SPL engineering allows developers to quickly and easily create multiple products tailored to individual customers' requirements. This helps to reduce costs and improve the performance of the software development process. However, due to the variability inherent to SPL systems, testing and debugging these systems is very challenging. Although automated debugging in single-system engineering has been studied in-depth, debugging SPL systems remain primarily unexplored.

This dissertation aims to shed light on automated debugging SPL systems by focusing on three tasks: *false-passing* product detection, variability fault localization, and variability fault repair. The contributions of the dissertation can be concluded as follows:

First, *the dissertation proposed* CLAP*, an approach for detecting false-passing products of buggy SPL systems.* Chapter 3 formulated the *false-passing* products detection problem. To solve this problem, CLAP introduced six *measurable attributes* to assess the strength of the failure indications in the products. These indications refer to the *implementation* and *test quality*; the stronger the indications, the more likely the product is *false-passing*.

Our results show that CLAP achieves more than 90% *Accuracy* in detecting *false-passing* and *true-passing* products. Especially, the *Precision* of CLAP in *false-passing* product detection is up to 96%.This means, among 10 products predicted as *false-passing* products by CLAP, there are more than 9 products which are indeed *false-passing* ones. This dissertation also evaluates the capability of CLAP in mitigating the negative impact of *false-passing* products on the FL performance. This dissertation conducted experiments on two state-of-the-art variability fault localization approaches with the five most popular SBFL ranking metrics. Interestingly, CLAP can significantly improve their performance in ranking buggy statements by up to 30%. This shows that CLAP can greatly mitigate the negative impact of *false-passing* products on localizing variability bugs and help developers find bugs much faster. The tool public is made public at *https://ttrangnguyen.github.io/CLAP/*.

Second, *the dissertation proposed* VarCop*, an approach for localizing variability faults.* Chapter 4 presented our observations about the visibility/invisibility of this kind of fault in SPL systems. Chapter 4 formulated the conditions (*Buggy PC*) to make variability fault visible in the products of a buggy SPL system and introduced important properties for detecting *Buggy PC*. For a buggy SPL system, VarCop localizes variability bugs by detecting *Buggy PC* to narrow the search space. Then, VarCop considers both the overall and detailed test results to figure out the positions of the faults.

The experimental results show that VarCop significantly outperformed the baselines in all the studied metrics. For the cases containing a single incorrect statement (single-bug), our results show that VarCop significantly outperformed S-SBFL, SBFL, and Arrieta et al. in **all 30/30** metrics by **33%**, **50%**, and **95%** in *Rank*, respectively. Impressively, VarCop correctly ranked the bugs at the top-3 positions in **+65%** of the cases. In addition, VarCop effectively ranked the buggy statements first in about **30%** of the cases, which **doubles** the corresponding figure of SBFL.

For localizing multiple incorrect statements (multiple-bug), after inspecting the first statement in the ranked list resulted by VarCop, up to **10%** of the bugs in a system can be found, which is **2 times** and **10 times** better than S-SBFL and SBFL, respectively. Especially, our results also show that in **22**% and **65%** of the cases, VarCop effectively localized at least one buggy statement of a system at top-1 and top-5 positions. From that, developers can iterate the process of bugs detecting, bugs fixing, and regression testing to quickly fix all the bugs and assure the quality of SPL systems. The tool public is made public at *https://ttrangnguyen.github.io/VARCOP/*.

Third, *the dissertation proposed product-based and system-based approaches for automatically repairing variability faults.* Chapter 5 introduced the detailed algorithms of these two approaches, and their enhanced versions with embedded heuristic rules. To improve fault repair performance, our heuristic rules leverage the intermediate repair information to guide the process of navigating modification points and selecting suitable modifications.

The experimental results show that the product-based approach is considerably better than the system-based approach by **12 to 30 times** in the number of plausible fixes and about **20 times** in the number of correct fixes. Interestingly, our heuristics could help to boost the performance of both product-based and system-based approaches by up to **200%**. For instance, by adopting the APR tool Cardumen, $ProdBased_{basic}$ and $SysBased_{basic}$ can **correctly fix 13 and 0 systems** respectively, while $ProdBased_{enhanced}$

and $SysBased_{enhanced}$ **correctly fix 40 and 1 systems** respectively. Moreover, the repair performance could be negatively impacted by FL tools since the modification points are selected based on FL results which are often imperfect. To mitigate the impact of the third-party FL tool, this dissertation assesses the effectiveness of the repair approaches if correct FL results are provided. In this experiment, the results show that the product-based approach is better than the system-based approach about **3 times** in effectiveness and **9 times** in efficiency. In addition, the proposed heuristic rules help to increase **30-150%** the number of correct fixes and decrease **30-70%** the number of attempted modification operations of the corresponding basic approaches. The tool is made public at *https://github.com/ttrangnguyen/SPLRepair*.

The limitation of the dissertation: For each proposed approach, the dissertation carefully analyzes the contribution of each component in the approaches to the whole performance, as well as the sensitivity of the approaches with the different inputs to figure out their weaknesses. Some limitations can be mentioned:

- Although the dataset uses the systems widely used in the existing work, this dataset only contains artificial bugs of Java SPL systems, so this dissertation cannot conclude the similar results for real-world faults.
- All systems in the benchmark are developed in Java. Therefore, this dissertation cannot claim that similar results would have been observed in other programming languages or technologies.
- To guarantee the reliability of SPL systems' test results, the flaky test problem is still challenging and has not been addressed.

Future works: From the results achieved in the dissertation, as well as the remaining limitations, there are some research directions for future work:

- *Collecting real-world variability bugs in larger SPL systems to more thoroughly evaluate the techniques.* Abal et al. have collected and presented a dataset of 98 real-world variability bugs in Linux, Apache, BusyBox, and Marlin systems. These bugs are essential for evaluating the QA tools of SPL systems. However, most of these bugs are compilation bugs, and they are not provided with test suites. Thus, this dataset does not fit well with the approaches leveraging testing information like SBFL, Clap, or VarCop, etc. In practice, collecting real-world bugs is very challenging. Thus,

it requires in-depth analysis and design to collect the bug systematically and automatically. In future work, I plan to investigate the bug-fixing commits which are often logged and reported. From these commits, the bug-introducing commits can be traced back and then the buggy versions of the systems can be obtained.

- *Extending the experiments with more APR tools.* This dissertation evaluated the variability bug repair performance with jGenProg and Cardumen. These tools can repair the program at different levels, i.e., statement and expression levels. However, there are much more APR tools, especially with the development of large language models and generative AI, multiple new APR tools have been introduced. In the next study, I plan to conduct more experiments with diverse APR tools to thoroughly evaluate the contribution of the heuristic rules and extend our conclusions.

- *Handling the flaky test problem to improve the quality of the test suites.* For the debugging approaches leveraging test results, the quality of the test suites is an essential factor. The low-quality test suites could result in both coincidental correctness and flaky test problems. The coincidental correctness leads to under-counting the failed tests and over-counting the passed tests, negatively impacting the performance of FL approaches. In this dissertation, CLAP has been introduced to address this phenomenon at the product level. Meanwhile, the flaky tests yield both passing and failing results despite zero changes to the code or test. This unreliability of the test results provides incorrect indications for FL and APR techniques. Thus diminishing their performance. In the future, I plan to analyze the symptoms of the flaky tests and design a specialized approach to detect these tests in SPL systems.

List of Publications

NTT1 . Nguyen, Thu-Trang, Kien-Tuan Ngo, Son Nguyen, and Hieu Dinh Vo. "A variability fault localization approach for software product lines." IEEE Transactions on Software Engineering 48, no. 10 (2021): ISSN 0098-5589, DOI: `https://doi.org/10.1109/TSE.2021.3113859`, ISI/Q1.

NTT2 . Nguyen, Thu-Trang, and Hieu Dinh Vo. "Detecting Coincidental Correctness and Mitigating Its Impacts on Localizing Variability Faults." In 2022 14th International Conference on Knowledge and Systems Engineering (KSE), pp. 1-6. IEEE, 2022.

NTT3 . Nguyen, Thu-Trang, Kien-Tuan Ngo, Son Nguyen, and Hieu Dinh Vo. "Detecting false-passing products and mitigating their impact on variability fault localization in software product lines." Information and Software Technology 153 (2023): ISSN 0950-5849, volume 153, DOI: `https://doi.org/10.1016/j.infsof.2022.107080`, ISI/Q1.

NTT4 . Nguyen, Thu-Trang, Xiao-Yi Zhang, Paolo Arcaini, Fuyuki Ishikawa, and Hieu Dinh Vo. "Automated Program Repair for Variability Bugs in Software Product Line Systems." Journal of Systems and Software. ISI/Q1 (accepted).

This list contains four publications.