*Impact Factor (SJIF): 3.632*

**International Journal of Advance Research in Engineering, Science & Technology**

*e-ISSN: 2393-9877, p-ISSN: 2394-2444*
***(Special Issue for ITECE 2016)***

# A Brief Comparision Between Software Product Line Testing-A Systematic Mapping And Sound Empirical Evidence in Software Testing

Meet Delvadiya[1], Kirti Santoki[2], Devshree Mehta[3]

[1]*ComputerDepartment, B.H.Gardi College Of Engineering & Technology*
[2]*ComputerDepartment, B.H.Gardi College Of Engineering & Technology*
[3]*ComputerDepartment, B.H.Gardi College Of Engineering & Technology*

***Abstract ---- This survey paper shows comparision of two software techniques.It highlights two software testing techniques:1)Empirical Evidence of Software Testing and 2)Software Product Line Testing.In this paper first we understand each technique and than comparining with following parameters or view points. how it works?what is its prons and cons? How is it implement?And more important issue in software testing is howmuch is it expensive?which is suitable for the developer?***

***Keywords – Case generation , unit testing ,search based software engineering , security exception, software product line***

## I.INTRODUCTION

Efficient testing strategies are important for any organization with a large share of their costs in software development. In an organization using software product lines (SPL) it is even more crucial since the share of testing costs increases as the development costs for each product decreases. Testing of a software product line is a complex and costly task since the variety of products derived from the product platform is huge.

Early literature on product lines did not spend much attention to testing but the issue is brought up after that, and much research effort is spent on a variety of topics related to product line testing. In order to get a picture of existing research we launched a systematic mapping study of product line testing. The aim is to get an overview of existing research in order to find useful results for practical use and to identify needs for future research.

Several promising techniques have been proposed to automate different tasks in software testing, such as test data generation for object-oriented software. However, reported studies in the literature only show the feasibility of the proposed techniques, because the choice of the employed artifacts in the case studies (e.g., software applications) is usually done in a non-systematic way. The chosen case study might be biased, and so it might not be a valid representative of the addressed type of software (e.g., internet applications and embedded systems). The common trend seems to be to accept this fact and get over it by simply discussing it in a threats to validity section. In this paper, we evaluate search-based software testing (in particular the EVOSUITE tool) when applied to test data generation for open source projects.

## II.SOFTWARE PRODUCT LINE TESTING-A SYSMETIC MAPPING STUDY

### 2.1Research Method:

#### 2.1.1.Research Questions:

The goal of this study is to get an overview of existing research on product line testing. The overall goal is defined in four research questions:

*RQ1 Which challenges for testing software product lines have been identified?*
*RQ2 In which fora is research on software product line testing published?*
*RQ3 Which topics for testing product lines have been investigated and to what extent?*
*RQ4 What types of research are represented and to what extent?*

**2.1.2. Sysmetic Mapping***:*

In order to get an overview of the research on SPL testing, a systematic mapping study is carried through. The mapping process consists of three activities;

i) Search for relevant publications,
ii) Definition of a classification scheme
iii) Mapping of publications.

Mapping system consists five steps and there are following here:
i)Exploratory Search.
ii)Extention by related work.
iii)Screening main conference proceeding.
iv)Validation against database.
v)Validation against systmetic review.
A summary of the inclusion and exclusion criteria is:

• **Inclusion:** Peer reviewed publications with a clear focus on some aspect of software product line testing.

• **Exclusion:** Publications where either testing focus or software product line focus is lacking. Non-peer reviewed publications.

**2.1.3. Threats to validity:**

This defines first three terms or keywords: construct validity, internal validity, external validity.
Construct validity: *It* reflects to what extent the phenomenon under study really represents what the researchers have in mind and what is investigated according to the research questions.

Internal validity: *It* is concerned with the analysis of the data.

External validity: *It* is about generalization from this study.

**2.2 Challenges in software line testing:**

**Large number of tests**

A major challenge with testing a software product line regards the large number of required tests. In order to fully test a product line, all possible uses of each generic component, and preferably even all possible product variants, need to be tested.

The main issue here is how to reduce redundant testing and to minimize the testing effort through reuse of test artefacts.

**Reusable components and concrete products**

The second most considerable challenge, which of course is closely related to the previous, is how to balance effort spent on reusable components and product variants.

**Variability**

Variability is an important concept in software product line engineering, and it introduces a number of new challenges to testing. Variability is expressed as variation points on different levels with different types of interdependencies.

**2.3 Classification Schemes**

Publications are classified into categories in three different dimensions. There are following here:i) *research focus ii) type of contribution and iii) research type*.

Six categories of research focus:

i) test organization and process,
ii) test management,
iii) testability,
iv) system and acceptance testing (ST and AT),
v) integration testing (IT),

vi) unit testing (UT), and
vii) automation.

Contribution type is classified into five categories: *Tool*, *Method*, *Model*, *Metric*, and *Open Items.*

**2.4 Discussion:**

The surveyed research indicates software product line testing being a rather immature area. Software product line testing seems to be a "discussion" topic. There is a well established understanding about challenges, as summarized in upper section. However, when looking for solutions to these challenges, we mostly find proposals. This is future research topic and it most discussion topic in research area.

## III . SOUND EMPIRICAL EVIDENCE IN SOFTWARE TESTING

**3.1 Basic Knowledege of Emipirical Evidence**

Software testing is expensive activity in software development therefore much research effort has been put into the question of how to automate it as much as possible. In this topic, we focus on test data generation for code coverage, in particular branch coverage in the context of object-oriented software. The simplest automated testing technique in this context is perhaps random testing but during the years different sophisticated techniques have also been proposed. At a high level, the current state of the art can roughly be divided into three main groups: variants of i)random testing ii) dynamicsymbolic execution and iii) search-basedsoftware testing.

For "simple" techniques such as random testing, it is possible to provide rigorous answers based on theoretical analyses. For more complex techniques where mathematical proofs become infeasible or too hard, researchers have to rely on empirical analyses. There are several challenges when carrying out empirical studies, among which there is the choice of the case study. If a technique works well in the lab on a specific case study, will it also work well in the real-world when it is applied by practitioners on their software? It might be that a novel technique works well in the lab just because the case study is too simple or small, but then it might fail on real-world instances. Even if real-world instances are used in a case study, the proposed technique might be too specific/biased toward those instances, and still fail when applied on new instances by practitioners.

**3.2 Objectives:**

The performance of test generation tools is commonly evaluated in terms of the achieved code coverage. High code coverage by itself is not sufficient in order to find defects as there are further major obstacles, most prominently the oracle problem: Except for special kinds of defects, such as program crashes or undeclared exceptions, the tester has to provide an oracle that decides whether a given test run detected an error or not. This oracle could be anything from a formal specification, test assertions, up to manual assessment. The oracle problem entails further problems; for example, in order to be able to come up with a test assertion a generated test case needs to be easily understandable and preferably short. However, in all cases a prerequisite to the oracle problem is to find an input that takes the program to a desired state. Therefore, in our experiment we compare the results in terms of the achieved branch coverage.

**3.3 SOFTWARE ENGINEERING EXPERIMENTATION**

To get a better picture of the current practice in evaluations in software engineering research, we surveyed the literature on test generation for object-oriented software. This is not meant to be an exhaustive and systematic survey, but rather a representative sample of the literature to motivate the work presented in this topic.

**3.4 Threats to Validity**

Threats to internal validity come from how experiments were carried out. We used the EVOSUITE tool for our experiment, which is an advanced research prototype for Java test data generation. Although EVOSUITE has been carefully tested, it might have internal faults that compromised the validity of the results. Furthermore, because EVOSUITE is based on randomized algorithms, we repeated each experiment on each class 10 times to take this randomness into account. However, because our study was focused on obtaining insights on the challenges of applying test generation tools in realistic settings, our research questions did not deal with comparisons of algorithms, and so statistical tests werenot required.

A possible threat to the construct validity is how we evaluated whether there are unsafe operations when testing a class. We considered the security exceptions thrown by all method calls in a test case, even when those methods do not belong to the class under test. Potentially, EVOSUITE might have tried to satisfy parameters of the class under test using classes that lead to actions blocked by the security manager, even if these parameters could also have been satisfied with other classes that do not result in any security exceptions.

The main goal of this topic was to deal with the threats to external validity that afflict current research in software testing. The SF100 corpus is a statistically sound representative of open source projects, and our results are also statistically valid for the other Java projects stored in Source Forge. For example, even if we encountered high kurtosis in the number of classes per project and branches per class, median values are not particularly affected by extreme outliers. To reduce such threat to validity, we used bootstrapping to create confidence intervals for some of the statistics.

## IV.CONCLUSION

We discuss both techniquies in detail in section II & III respectively. So it is conclude that sound of empirical evidence in software testing is best suitable for small product testing.it has less cost than software product line.
Software product line is a thinking idea. It is not practically possible in laboratory. Software product line have many challenges and now a days there are under resolving process where as empirical evidence is much better technique for test case generation and software testing. The software product line technique is discuss tipic in now a days.

**References**

[1] W. Afzal, R.Torkar, R. Feldt, A Systematic Mapping Study on Non-Functional Search-Based Software testing, in 20th *International Conference on Software Engineering and Knowledge Engineering (SEKE),* (2008).
[2] A. Bertolino, S. Gnesi, Use Case-based Testing of Product Lines, Proc. *ESEC/FSE*, pp. 355-358, ACM Press.(2003).
[3] T. Kahsai, M. Roggenbach, B.-H. Schlinglof: Specification-based testing for software product lines. *In Sixth IEEE International Conference on Software Engineering and Formal Methods, SEFM 2008, Cape Town, South Africa, 10-14 November 2008* (2008)
[4] R. Kolb and D. Muthig. Techniques and Strategies for Testing Component-Based Software and Product Lines. Chapter 7 in *Development of Component-Based Information Systems. Advances in Management Information Systems Volume 2 / 2006* , pages 123 - 139 (2006)
[5] B.P Lamancha,. M.P Usaola and M.P Velthius. Software Product Line Testing - A Systematic Review. *4th International Conference on Software and Data Technologies* (*ICSOFT)* p. 23-30. (2009)
[6] J. J. Williams. Test Case Management of Controls Product Line Points of Variability. *International Workshop on Software Product Line Testing, SPLiT*, (2004)
[7] H Zeng, W Zhang, D Rine. Analysis of Testing Effort by Using Core Assets in Software Product Line Testing. *International Workshop on Software Product Line Testing, SPLiT*, (2004)
[8] J. H. Andrews, T. Menzies, and F. C. Li. Genetic algorithms for randomized unit testing. IEEE Transactions on Software Engineering (TSE), 37(1):80–94, 2011.
[9] G. Fraser and A. Arcuri. Evosuite: Automatic test suite generation for object-oriented software. In ACM Symposium on the Foundations of Software Engineering (FSE), 2011.