Cost-Based Test Case Prioritization Technique for Software Product Line

Satendra Kumar[1], Rajkumar[2]

Department of Computer Science,

Gurukula Kangri Vishwavidyalaya Haridwar, Uttarakhand, India

Abstract - Nowadays, almost all software industries are transforming from single software product development to multiple similar types of software development. Software industries are adopting the concept of software product line (SPL) which gives the new dimension to the development of multiple similar types of software using a feature model (FM). In FM, the feature is a behavior and a capability of an SPL which captures the information regarding all possible software products and represents in terms of features and relationships among them. If the number of features increases linearly, on the effect of that newly generated products increase exponentially. Moreover, testing of an SPL is not an easy task as compared to single software testing. To overcome this problem, we generate the test cases using FM on the basis of feature groups. After generating the test cases, we prioritize them to make the effective and efficient testing of an SPL. In this paper, we proposed cost-based test case prioritization (TCP) technique for SPL. Cost-based prioritization is performed at the cost of each configuration.

Keywords: Configuration, Cost, Feature model, Software product line, Test case, Test case prioritization.

I. INTRODUCTION

Software industries are gradually changing the concept of software development towards the software product line (SPL). SPL reduces the time and cost of software product development and enhances the quality of products. If user's requirements increase, it makes the complex and expensive development. Therefore, in the product line technique, there is involving most common one feature from the thousands of features which are including special features concerned with the customer needs. Initial stages of an SPL take more time to develop the product as compared to the development of a similar type of product using the single system because it requires some efforts for realization architecture and planning of the infrastructure. New Products can be developed easily using the infrastructure. Therefore, it will take less amount of time to develop the new products. Products are derived from SPL. A product consists of several components selected from existing component libraries. These components connected to each other with a common platform to perform particular functionalities. To increase the productivity and quality of software products, expert software engineers use the practices and technologies from a variety of fields [1].

SPL Engineering is related to developing a set of similar types of products. Products are developed by reusing a similar group of features instead of developing each product from the initial stage. Features have the main role to differentiate all products in an SPL. Behavior and capabilities of a software system are also defined by a feature. Often, SPLs are represented by feature models (FMs). The information of all the feasible software products is captured by FM that is represented by the relationships and features among them. A smart-phone SPL is represented by figure 1.1. We perform the automated analysis of FMs using the information which is extracted from FMs with the help of a computer. These analyses concentrate on studying consistency, complexity, and variability degree, etc. properties of the SPL. In the last 2 decades, many tools, techniques, and operations have been presented for the analysis purpose of FMs [2, 3].

In SPL testing, first of all, a set of products is generated, after that, testing is performed on each product. The products are also called SPL configurations. In SPL, a test case can be defined as a configuration or product. A test case is a set of features of an SPL. There are various techniques used to generate test cases and to detect the faults. Some techniques are effective to detect faults in a minimum amount of time and some effective in terms of cost of testing. If, feature combinations number is high then it may generate thousands of different products [4]. During the configuration process, we select each valid feature from an FM and generate the configurations. If an FM has n number of features then it will potentially generate 2^n configurations due to their tree-like and hierarchical structure. Given their hierarchical and tree-like structure, an FM with n features can potentially produce 2^n configurations. If the number of features increases linearly, on the effect of those new generating configurations (products) increase exponentially [5]. So thoroughly testing of an SPL is impossible because, testing of each product is too costly. In this context, the order is usually assumed to be inappropriate in which products are tested. As a result, this condition may be occurs that most important test cases, those detect large number of faults, are run at the last place. Before gazing the correction of faults, it forces to the tester to wait for a

long time. In a worst case, the faults could be undetected, if the testing resources are exhausted before running all the test cases. It depends on the tester in which order, he wants to test the products. That's why, we need test case prioritization (TCP) techniques to order the test cases (products) for the testing purpose.

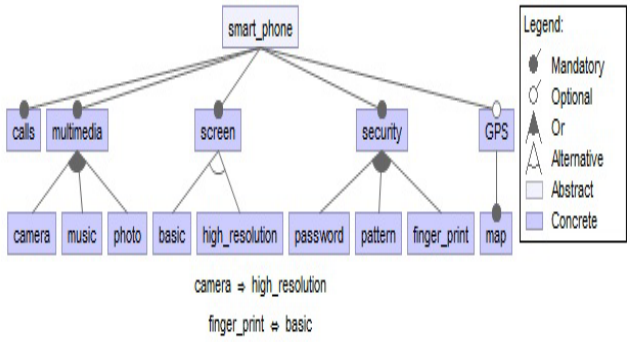

Fig. 1.1 An example of a feature model of SPL smart-phone

TCP techniques schedule the test cases for execution in an order that attempts to increase their effectiveness at meeting some performance goal. There are many goals such as code coverage at the fastest rate, increase the rate of fault detection or exercise components in the expected frequency of use of test cases. Various ordering criteria may be proposed for a given goal [2]. After generation of products, the tester can schedule the test cases, according to the cost of configurations in order to grow the rate of fault detection. In this paper, we propose a technique to prioritize the test cases, according to the cost of products after generating them. We generate the cost of each product in the following way.

Feature model is generated using the feature IDE tool. Features are selected from the feature model. We assign the cost to each feature. After this process, we generate the configurations of all the possible products. We add the cost of all the features included in a configuration and generated the cost of each configuration. If a product has maximum cost, then we select that product because maximum mandatory features are included in that product configuration. After that, we choose the next test case which has less cost of the previous test case (configuration). This process is continued until all the test cases are not chosen. TCP techniques are supplementary to the test case selection (TCS) techniques in demanding testing environment.

The remaining paper is organized in the following manners: In section 2, we present the related work of prioritization approach for SPL. Section 3 describes the feature modeling and how to generate configurations from feature model. We present Cost-based TCP technique and algorithm for SPL in section 4. We implement the proposed algorithm and generate the cost-based prioritized test cases in section 5. Finally, we present the conclusion and future work in section 6.

II. RELATED WORK

Several TCP techniques have been proposed for SPLs. Some of them are described here which are related to our work. For SPLs, Sánchez et al. [2] presented TCP technique. To increase the frequency of fault detection which provides the earlier feedback and decreases the debugging effort, they presented five prioritization criteria in their research paper to arrange the test cases to execute on an order. For analysis of FMs, these prioritization criteria are fully automated because they depend on standard approach and matrix. Experimental results show that dissimilar prioritization criteria offer the distinct rate of fault detections. In this paper, results also show that the fault detection of both random and pairwise based SPL test suites accelerated by the some proposed criteria. This work is a complement for TCS technique. This work is not only concentrated on which product of SPL should be tested, but also on how they should be tested. This work concentrated on the order of test cases of SPL in which they are run does matter.

For SPL testing, Al-Hajjaji et al. [6] presented similarity based prioritization approach. They prioritized the configurations to know that which products have the defects earlier. They also assumed that maximum errors are occurred due to the interaction of some features. They applied and estimated their methodology on some sampling algorithms Chvatal, CASA, and ICPL. They performed their experiments on Mobile Phone and Smart Home SPLs of different size. They found that early interaction coverage of similarity based prioritization is better than CASA, random and Chvatal order.

For evaluating FM of an SPL, Ensan et al. [5] proposed an approach to reduce the test space and prioritize the test cases. By testing some test cases, they wanted to get higher error coverage. To achieve this goal, they performed two-step process: (1) feature selection (2) test case prioritization. They reduced the test space by selecting the features which are correlated to the important goals and prioritized the generated test cases. They had proposed feature selection approach based on goal model. Goal model was able to model the preferences of the domain stakeholders. They validated their approach using an e-shop SPL.

Devroey et al. [8] proposed an approach based on Statistical Prioritization for SPLs Testing. In this paper, they proposed sampling technique to reduce the number of products to test. They started from FM and applied a coverage criterion to create fault finding, tractable, configuration lists to be tested. They explored how ideas of a usage model (Markov chain) based statistical testing used to take out interested configurations according to the opportunity of their execution. Executions of

configurations were stored in featured transition systems (FTS). It is a compact demonstration of SPL behavior.

Wang et al. [9] presented search based multi-objective test prioritization technique to validate video conferencing products of a future test scheduling system of Cisco, which consider multiple cost and effectiveness measures. It was an industrial case study. Multi-objective technique decreases the execution time and increases the number of prioritized test cases, faults detection capability and feature pairwise coverage. A fitness function is also used for test case prioritization, problem-based on cost effectiveness measures. The fitness function is evaluated empirically with the three most used search algorithms which are (1+1)EA (Evolutionary algorithm), Random Search and 500 artificial designed problems. The experiment results showed that (1+1) EA perform better than others to resolve the problem of test case prioritization.

Machado et al.[10] presented a tool named SPLConfig. Using search based algorithms, it was used to generate product configuration in the SPL. They described the working of the tool and concise the method under the tool and its functionaries. In general, their results were satisfactory. Still, their goals were an improvement of SPLConfig tools for future work in order to include (i) another non-functional requirement (ii) the problem occurs during the development and maintenance of product for industries. In such a way, minimize the development efforts which are used to integrate the feature to develop the products.

III. FEATURE MODELING

Feature is a characteristic of a software product related to some stakeholder. It depends on the need of the stakeholders that a feature can be a requirement, functional or nonfunctional characteristic or a technical function. Feature model (FM) is a hierarchy or tree-like structure with features, creating nodes of the tree. The line and features groupings, both represent to the feature variability. Four types of feature groups are following: *"mandatory"*, *"alternative" "optional"*, and *"or"*. When generating the configurations, we need to decide which feature is to be included in a configuration. There are following rules applied to generate configurations: If a parent feature is involved in a configuration,

- All its ***mandatory*** sub features must be involved ("n from n"),
- Exactly one feature must be involved from ***alternative*** features group ("1 from n"),
- Selected any number of ***optional*** features from optional group ("m from n, 0 < = m<=n"),
- At least one feature must be selected from ***or*** features group ("m from n, m>1").

FM holds the information regarding all possible software products and represents in terms of relationships and features among them [7]. In an SPL, some combinations of features are considered valid that are derived from the above rules. Such valid combinations are called configurations [6]. We can generate a product from each configuration. Valid combinations are also generated from an FM.

The above rules are applied to the feature model as illustrated in figure 1.1. A feature is selected when its parent feature is selected. All the mandatory features such as calls, multimedia, screen, and security are selected. But GPS belongs to the optional group so it may or may not be involved in the configuration. If we select the GPS feature, then we have to select the map feature because the map is a mandatory feature. In the OR-group, at least one feature must be selected. Features such as camera, music, and photo belong to OR-group so we can select one or more feature(s) of them. Password, pattern and finger_print features also belong to OR-group so we have to select at least one feature of them. Features such as basic and high_resolution belong to an alternative group so we can select exactly one feature of them.

There are two types of features: (1) abstract (2) concrete. If implementation artifacts are mapped with a feature, then it is called concrete feature; otherwise, it is called abstract feature. Moreover, features may have additional dependencies such as cross-tree constraints which cannot be defined by only the hierarchical structure. Propositional formulas are called Cross-tree constraints usually shown in the above feature model of smart_phone. As shown in Figure 1.1, Cross-tree constraints such as requires and excludes are used by the feature model of SPL smart_phone. In figure 1.1, requires constraint is denoted by => symbol. For example, if feature camera is selected, then feature high_resolution must be selected to be included in a smart_phone. As shown in figure 1.1, excludes constraint is denoted by <=> symbol. For example, features finger_print and basic cannot involve in a smart_phone at the same time [6].

IV. PROPOSED COST-BASED TEST CASE PRIORITIZATION TECHNIQUE

Cost-based test case prioritization technique is needed to minimize the cost and time of testing of all the products which are generated by the feature model. Cost-based TCP approach is applied from the initial stage when we assign the cost to each feature. But the main work of the TCP is started when we generate the configurations. There is following procedure to perform the cost-based TCP technique.

First of all, a list of features is generated and the cost assigned to each feature. Features are involved in the list according to the customers' requirements or market demands. We generate the feature model of an SPL using the feature IDE tool. Valid configurations are generated using the feature IDE tool. Assign the cost to each feature of all the configurations according to the list that is previously generated. Add the cost of all the features included in a configuration and generate the cost of each configuration. In such a way, we get the cost of each product before testing them. If a configuration or product has the highest cost, then we select that product for testing because large numbers of features are included in that product, necessary for market demand. Mandatory features are involved in each configuration and have important roles to generate the products.

Algorithm Cost-based test case prioritization algorithm

1: $F = \{f_1, f_2, f_3, ..., f_n\}$ (set of features)

2: $Cost = \{r_1, r_2, r_3, ..., r_n\}$ (set of cost, cost assigns to each feature)

3: Draw the feature model of an SPL.

4: Generate a set of valid configurations.

5: Cost assigns to the features of each configuration according to the previous cost.

6: **Input:** $C = \{c_1, c_2, c_3, ..., c_m\}$ (set of valid configurations)

7: Find output: P= (list of prioritized configurations)

8: P ← [ ]

9: i=1

while (i<=|C|) do

$Cost(c_i) = 0$

k=1

while (k<=|c_i|) do

$Cost(c_i) = Cost(c_i) + c_i[k]$

k=k+1

end while

return(Cost (c_i))

i=i+1

end while

return(Cost(C))

10: **while** (C ≠ empty)

Select $c_i \in C$ where c_i has maximum cost

P.add(c_i)

C.remove(c_i)

end while

11: return P

12: **Output:** P= {cost-based prioritized configuration}

V. IMPLEMENTATION OF PROPOSED TECHNIQUE

First of all, we generate the list of features will be included in configurations to generate the products.

{smart_phone, calls, multimedia, camera, music, photo, screen, basic, high_resolution, security, password, pattern, finger_print, GPS, map}

After that, the cost is assigned to each feature to generate the cost-based test cases on the basis of the actual cost of each individual component/feature of the smart-phone.

{smart_phone=2000, calls=700, multimedia=1000, camera=3000, music=1000, photo=300, screen=5000, basic=2000, high_resolution=3000, security=3000, password=500, pattern=600, finger_print=2000, GPS=1000, map=500}

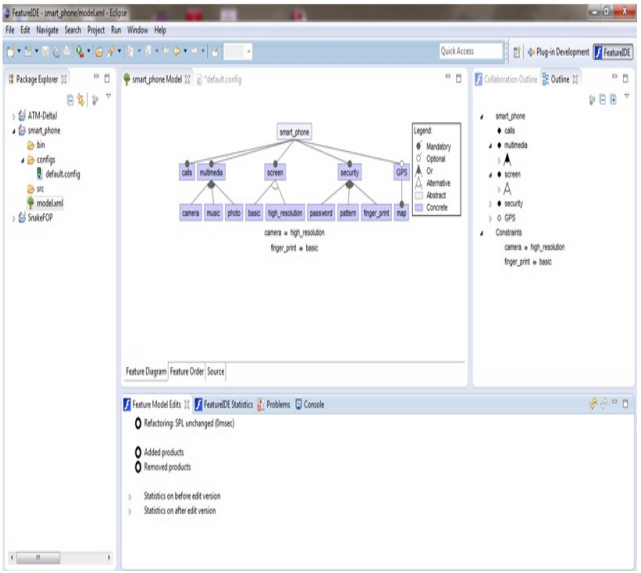

Fig. 5.1 FeatureIDE editor to generate the feature model

We implement the proposed approach using the featureIDE tool. FeatureIDE tool is an open source framework which is based on Eclipse and that supports all the phases of development of SPLs projects from domain analysis to products generation. FeatureIDE concentrates on the whole development process and integrates the tool to implement the SPLs into an integrated development environment [6] such as eclipse. FeatureIDE is used to create a feature model by adding, removing and editing features using four feature groups which are mentioned in above section III. Here, we are using SPL editor i.e.

featureIDE to draw a feature model of smart_phone as shown in figure 1.1. There are four parts shown in the figure 5.1 of the featureIDE tool. First one is package explorer. It shows the details of all the created projects. The second one is a featureIDE model editor. The third one is an outline view and the last one is the feature model edits view, which shows the details of features.

Here, we are using featureIDE tool to generate the valid configurations. We can also generate the configurations manually, but it will be complicated for the large feature model and it is not an efficient solution to generate the valid configurations which should be generated automatically. A user can generate and edit the valid configurations using the featureIDE tool as shown in figure 5.2. The second part shows the 8 valid possible configurations. Fourth part shows featureIDE statistics, which hold the information about the generated SPL as Smart_Phone.

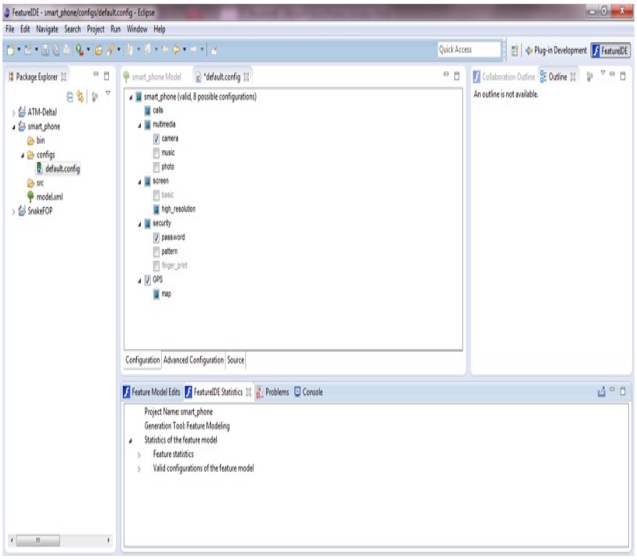

Fig. 5.2 Generation of configurations using featureIDE tool

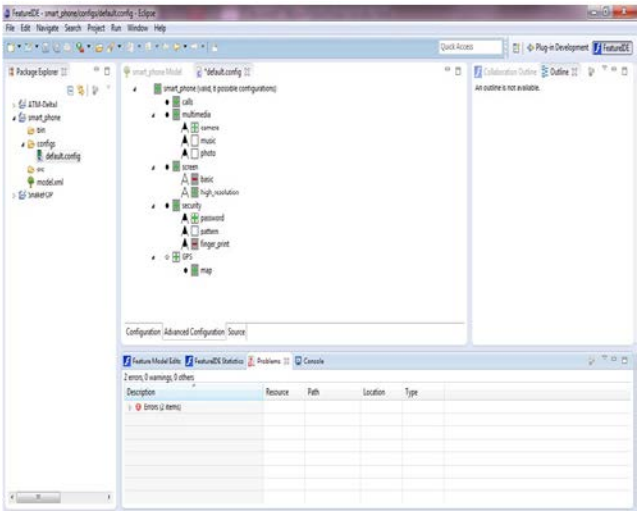

Fig. 5.3 Generation of advanced configuration

Advance configurations are also generated using the featureIDE tool as shown in figure 5.3. In the second part, boxes with plus sign show the selected feature and boxes with the minus sign show the excluded features. Fourth part shows the errors occur after running the projects.

This is a different form of representation of valid configurations. There is no other operation performed to generate the advanced configurations. Lot of configurations can be generated from the above feature model. But we are writing 8 valid configurations in the following manners according to the selected features in figure 5.3.

c1= {smart_phone, calls, multimedia, camera, screen, high_resolution, security, password, GPS, map}

c2= {smart_phone, calls, multimedia, camera, music, screen, high_resolution, security, password, GPS, map}

c3= {smart_phone, calls, multimedia, camera, music, photo, screen, high_resolution, security, password, GPS, map}

c4= {smart_phone, calls, multimedia, camera, music, photo, screen, high_resolution, security, password, pattern, GPS, map}

c5= {smart_phone, calls, multimedia, camera, music, screen, high_resolution, security, password, pattern, GPS, map}

c6= {smart_phone, calls, multimedia, camera, photo, screen, high_resolution, security, password, pattern, GPS, map}

c7= {smart_phone, calls, multimedia, camera, photo, screen, high_resolution, security, password, pattern, finger_print, GPS, map}

c8= {smart_phone, calls, multimedia, camera, music, photo, screen, high_resolution, security, password, pattern, finger_print, GPS, map}

According to the algorithm, we assign the cost to each feature of each configuration and add them. We generate the cost of each configuration. In such a way, we get the cost of each product before testing them.

c1={smart_phone, calls, multimedia, camera, screen, high_resolution, security, password, GPS, map}

c1={2000+700+1000+3000+500+3000+3000+500+1000+500} =15200

c2={smart_phone, calls, multimedia, camera, music, screen, high_resolution, security, password, GPS, map}

c2={2000+700+1000+3000+1000+5000+3000+3000+500+1000+500}=20700

c3={smart_phone, calls, multimedia, camera, music, photo, screen, high_resolution, security, password, GPS, map}

c3={2000+700+1000+3000+1000+300+5000+3000+3000+500+1000+500} =21000

c4={smart_phone, calls, multimedia, camera, music, photo, screen, high_resolution, security, password, pattern, GPS, map}

c4={2000+700+1000+3000+1000+300+5000+3000+3000+500+600+1000+500} =21600

c5={smart_phone, calls, multimedia, camera, music, screen, high_resolution, security, password, pattern, GPS, map}

c5={2000+700+1000+3000+1000+5000+3000+3000+500+600+1000+500} =21300

c6={smart_phone, calls, multimedia, camera, photo, screen, high_resolution, security, password, pattern, GPS, map}

c6={2000+700+1000+3000+300+5000+3000+3000+500+600+1000+500} =20600

c7={smart_phone, calls, multimedia, camera, photo, screen, high_resolution, security, password, pattern, finger_print, GPS, map}

c7={2000+700+1000+3000+300+5000+3000+3000+500+600+2000+1000+500} =22600

c8={smart_phone, calls, multimedia, camera, music, photo, screen, high_resolution, security, password, pattern, finger_print, GPS, map}

c8={2000+700+1000+3000+1000+300+5000+3000+3000+500+600+2000+1000+500} =23600

Here, C is a set of configurations. Cost(C) is a set of cost all the configurations. P is a set of prioritized test cases based on cost.

C= {c1,c2,c3,c4,c5,c6,c7,c8}

Cost(C)={15200,20700,21000,21600,21300,20600,22600, 23600}

P= {} here, p is an empty set.

After generating the cost of all the configurations, we select the configuration which has the highest cost and adds it to P set and removes it from set C. Now we look maximum cost in the set Cost(C). 23600 is a maximum cost which is related to the configuration c8. Now, we add c8 in P-set and remove it from set C.

P= {c8}

C= {c1,c2,c3,c4,c5,c6,c7}

Cost(C)={15200,20700,21000,21600,21300,20600,22600 }

After this, we see that c7 has the maximum cost. C7 is 8

P= {c8, c7}

C= {c1,c2,c3,c4,c5,c6}

Cost(C)= {15200,20700,21000,21600,21300,20600}

This process runs continue until all the configurations is not added in set P. In such a way, we get the prioritized set P.

P={c8,c7,c4,c5,c3,c2,c6,c1}

VI. CONCLUSION

SPL testing is a complicated task as a comparison to the single software system testing. To overcome this problem, TCP technique is used. All the proposed TCP techniques concentrated on the different approach instead of cost-based approach. According to those techniques, if a product has higher cost still it is tested later. No one has concentrated on the cost-based prioritization. If a configuration has a higher cost, it is prioritized first and after that second one is prioritized. This process is continued until all the configurations are not prioritized.

VII. FUTURE SCOPES

Our approach is not automated and test space is also increased due to a large number of features using in the feature model. Test cases also increase as the amount of features increase in the feature model. So in the future, we'll automate this approach and reduce the size of the test space. We'll eliminate the test cases which are not mandatory to reduce the size of the test space.

REFERENCES

[1] Kaur, M. and Kumar, P. (2014) 'Systematic Review on Software Product Line Engineering (SPLE)', *International Journal of Advanced Research in Computer Science and Software Engineering*, Vol. 4 No. 1, pp. 1096-1101.

[2] Sánchez, A. B., Sergio, S. and Antonio R. C. (2014) 'A comparison of test case prioritization criteria for software product lines' in *ICST 2014: Proceedings of Software Testing, Verification and Validation*, IEEE Seventh International Conference, Cleveland, OH, USA, pp. 41-50.

[3] Kumar, S. and Rajkumar. (2016) 'Test Case Prioritization Techniques for Software Product Line: A Survey' *in ICCCA 2016: International Conference on Computing, Communication and Automation*, IEEE International Conference, Noida, India, pp. 884-889.

[4] Mendonca, M., Branco, M. and Cowan, D. (2009) 'S.p.l.o.t. - software product lines online tools' In *OOPSLA 2009*: *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, Orlando, Florida, USA, pp. 761-762.

[5] Ensan, A., Bagheri, E., Asadi, M., Gasevic, D. and Biletskiy Y. (2011) 'Goal-Oriented Test Case Selection and Prioritization for Product Line Feature Models' In *ITNG 2011: Proceeding of Information Technology: New*

Generations, IEEE Eight International Conference, Las Vegas, NV, USA, pp. 291-298.

[6] Al-Hajjaji, M., Thüm, T., Meinicke, J. and Saake, G. (2014) 'Similarity-Based Prioritization in Software Product-Line Testing' In *SPLC 2014, Proceedings of the 18th International Software Product Line Conference*, Florence, Italy, pp. 197-206.

[7] *Pure-System project.* [online] http://www.pure-systems.com/ (Accessed 21 December 2016)

[8] Devroey, X., Perrouin, G., Cordy, M., Schobbens, P.Y., Legay, A. and Heymans, P. (2014) 'Towards Statistical Prioritization for Software Product Lines Testing' In *VaMoS 2014, Proceedings of the Eighth International workshop on Variability Modeling of Software-Intensive Systems*, Sophia Antipolis, France, Article No. 10.

[9] Wang, S., Buchmann, D., Ali, S., Gotlieb, A., Pradhan, D. and Liaaen, M. (2014) 'Multi-Objective Test Prioritization in Software Product Line Testing: An Industrial Case Study' In *SPLC 2014, Proceedings of the 18th International Software Product Line Conference*, Florence, Italy, pp. 32-41.

[10] Machado, L., Pereira, J., Garcia, L. and Figueiredo, E. (2014) 'SPLConfig: Product Configuration in Software Product Line' In*: CBSoft 2014, Brazilian Congress on Software, Tools Session*, Brazil, pp. 1–8.