ENGINEERING JOURNAL

Article

Input Modeling Prioritization Using Statistically User Profile for Pairwise Test Case Generation with Constraints Handling

Sompong Nakornburi[a] **and Taratip Suwannasart**[b,*]

Software Engineering Laboratory, Center of Excellence in Software Engineering. Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok 10330, Thailand
E-mail: [a]nakornburi.s@gmail.com, [b]Taratip.S@chula.ac.th (Corresponding author)

Abstract. Pairwise testing is a widely used technique for software testing with the reduced size of the test suite and able to detect interactions that trigger the system's faults. In addition, pairwise testing test suites must be able to take constraints between input parameters and parameter values into account. In current practice, identifying and selecting input parameters and parameter values usually depends on tester skills that might not be sufficient. Input parameters and parameter values modeling and tools for easily guiding and prioritizing the selection of optimal input parameters and parameter values for the SUT is also required. In this work, we present an approach for prioritizing input parameters and parameter values modeling using statistical user profile. Our approach is implemented in a tool called UPPTCT which provides the ability to handle constraints on input parameters and parameter values for pairwise testing in order to generate test cases. We conduct experiments to evaluate test case effectiveness and compare our tool with other renowned pairwise test generation and constraints handling tools. The experimental results show that the effectiveness of our approach is significantly more efficient and effective than random testing as a large portion of reported defects with regard to statically user profile were caught by our approach. Furthermore, our tool performs better in some cases and performs comparable results for generating test cases upon input parameters and parameter values for both with constraints handling and without constraints handling.

Keywords: Input parameters modeling, pairwise testing, test case prioritization, test case selection, constraints handling.

ENGINEERING JOURNAL Volume 21 Issue 7
Received 27 April 2017
Accepted 31 August 2017
Published 29 December 2017
Online at http://www.engj.org/
DOI:10.4186/ej.2017.21.7.389

DOI:10.4186/ej.2017.21.7.389

1. Introduction

Software testing is one of the important activities in software development process, it requires more than roughly half of the budget in the total estimated. Modern software systems are designed and intended to run on various platforms and environments. Consequently, a software system can be installed in a different type of platform configurations under different hardware setup and run on the different operating system. Testing on a high configuration software systems is challenge because software systems tend to be more complicated in term of an enormous possible number of test configurations that need to be tested and verified. Nevertheless, verification and validation on software system are needed to ensure the correctness and high quality of product value before delivering. Due to budget and time constraints, exhaustive testing in such a modern software system is practically impossible as it requires extensive efforts and excessive resources [1, 2].

Recent researchers suggest that a particular one of the widely practical used combinatorial testing technique is pairwise testing, it grew into extensive use and has been proved a useful technique for software testing and fault detection [3, 4]. In addition, using a combinatorial testing technique not only detect more defects but also reduces time for designing test cases for a specific system under test (SUT) and significant cost and resources savings [5]. Pairwise testing produces a small test suite that can reduce the cost of testing and helps to detect interaction defects which are constructed by the combination of relevant input parameters and parameter values.

Practical software systems usually have constraints between the combination of parameters and parameter values, resulting in more or less combination between parameters and parameter values are often not valid and cannot be tested. Constraints between parameters and parameter values must be set by a tester before test case generation and must be taken into account during test case generation or after test case generation.

The most important step in combinatorial testing is input parameters and parameter values modeling. It is very ambitious to identify and select an optimal input parameters and parameter values for a SUT. An empirical research shows that identifying and selecting input parameters and parameter values is a creative process which cannot be completely automated [6]. Different testers would think of various models depending on their creativity and experiences [7]. To our knowledge, there is currently no adequate empirical results, scientific recommendations, or any strong theoretical technique to formulate an optimal set of input parameters and parameter values. Therefore, a user profile was found and used to indicate the frequency of a software system execution. By applying statistical user profile, a goal-oriented testing that has been found the most ideal approach that would help us to prioritize testing effort related to the actual usage of the software system and how often of the software system is being used [4, 8, 9]. As a result, we can dispose of each individual tester skills.

In this paper, we propose an approach and a tool to prioritize input parameters and parameter values modeling based on statistically user profile and also provide the ability for constraints handling on input parameters and parameter values using pairwise testing to generate test cases. Our rationale for conducting this research is to bridge the gap between dependency of each tester skills and techniques to formulate an optimal set of input parameters and parameter values.

The remainder of this paper is structured as follows: Section 2 briefly reviews the background and related works. Section 3 describes our proposed modeling approach. Section 4 gives descriptions and the functionalities of our tool implementation. Section 5 reports the experimental results. Finally, section 6 provides conclusion and plan for future work.

2. Background and Related Work

2.1. User Profile

A user profile is an accumulation of settings and information related to a user to run the software which is being tracked. It can be characterized as a user identification with respect to operating environments such as platform, operating system, software applications, hardware, or additional description [10, 11].

A user profile is used to demonstrate relative execution frequencies of a software to be used. With a user profile, a software can be tested more efficiently because testing can focus on the most frequently used operations by establishing relative weights that allows testing efforts to be distributed according to the usage.

It is the most ideal approach that would guarantee that a software is delivered with maximized reliability because the most used of user profiles will be tested the most [12].

The construction of user profile is one of the most important processes in SUT and it requires huge amount of efforts from testers that have the system specifications, application workflows, and application domains expert knowledge of SUT. Testers can analyze the user profiles that represent the dynamic usage characteristics and provide statistical information of SUT, and it is automatically generated from the production environment which is always up to date.

Tomohiko Takagi and Zengo Furukawa [13] presented a novel construction technique of a huge operational profiles concerning to effectively apply statistical testing to the large software project is implemented on a tight schedule by constructing elemental operational profiles, small operational profiles that represent usage characteristics of components of SUT. The key idea is that an operational profile of the whole of SUT is too extensive to be built physically, but operational profile of the components of SUT are small so that it is easy to be constructed.

John D. Musa [14] stated that the user profile can be developed in two forms which are explicit and implicit. For an explicit profile, each of operation is completely described with the values of all of its attributes, occurrence probabilities are explicitly assigned to each operation. For an implicit profile, occurrence probabilities are implicitly assigned to each of operation separately by determining occurrence probabilities for the values for each attribute. Note that the operational attribute values and their related occurrence probabilities yield a sub-profile for each operational attribute.

If an explicit operational profile has been designed, the test cases can be selected efficiently due to the fact that the most frequency used of operational profile will be tested the most. If an explicit operational profile has been designed, key input variables and their related values have been selected according to their occurrence probabilities that must be tested [12]. Additionally, he classified operations in two categories which are frequent and infrequent and critical operation [15].

K. Saravana Kumar and Ravindra Babu Misra presented four categories of operations which are (1) frequent and critical operation, (2) frequent and non-critical, (3) infrequent and critical, and (4) infrequent and non-critical.

According to John D. Musa [14], there are several benefits for applying user profiles. (1) Increase user satisfaction by catching their needs more accurately, (2) Satisfy important user needs quicker with operational development, (3) Reduce costs with decreased operational software, (4) Speed up development and enhance productivity by allocating resources into the criticality one, (5) Guide distribution of review efforts, (6) Reduce system risk with more reasonable testing, (7) Make testing quicker, (8) Help turn the system architecture to user and criticality, (9) Make performance evaluation and management more accurate, and (10) Guide development of better manuals and training.

In this paper, we describe a user profile as a set of settings and information related to a user and the SUT. It can be characterized as the explicit representation of the identity of the user concerning the platform, operating system, software applications, and also hardware information. User profile must be collected and prepared into the system for the purpose of analyzing and prioritizing test cases. The user profile can be in both two forms which are explicit and implicit, also can be classified operations in several categories which are frequent, infrequent, critical, non-critical, and the combination of operation.

An example of a user profile which is retrieved from the database. The user profile comprised of a large number of configuration parameters for a system that is designed to execute on a different platform as shown in Fig 1.

DOI:10.4186/ej.2017.21.7.389

	A	B	C	D	E	F	G	H
1	**UserID**	**Region**	**Country**	**Browser Version**	**Color Depth**	**Microsoft Office**	**Operating System**	**User Rights**
2	UU-11604	CEMA	Denmark	Internet Explorer 11	32 bits	Office 365 (2016) 64-bit	Windows 10 Pro 64-bit	Administrator account (restricted)
3	UU-11612	UKI	United Kingdom	Internet Explorer 10	32 bits	Office 365 (2016) 64-bit	Windows 10 Pro 64-bit	Administrator account (restricted)
4	UU-11818	UKI	United Kingdom	Internet Explorer 11	16 bits	Office 2010 32-bit	Windows 10 Pro 64-bit	Administrator account (elevated privileges)
5	UU-11981	Americas	Canada	Internet Explorer 9	16 bits	Office 365 (2016) 64-bit	Windows 10 Pro 64-bit	Administrator account (restricted)
6	UU-11983	Americas	Canada	Internet Explorer 10	32 bits	Office 365 (2016) 64-bit	Windows 10 Pro 64-bit	Standard Account
7	UU-11983	Americas	United States	Internet Explorer 8	32 bits	Office 2013 32-bit	Windows 7 Professional 64-bit	Administrator account (restricted)
8	UU-12124	Americas	United States	Internet Explorer 11	32 bits	Office 2013 32-bit	Windows 10 Pro 64-bit	Administrator account (restricted)
9	UU-12153	Americas	United States	Internet Explorer 10	32 bits	Office 2013 32-bit	Windows 10 Pro 64-bit	Administrator account (elevated privileges)
10	UU-12170	Americas	United States	Internet Explorer 9	32 bits	Office 2013 32-bit	Windows 10 Pro 64-bit	Standard Account
11	UU-12260	Americas	Canada	Internet Explorer 9	64 bits	Office 365 (2016) 64-bit	Windows 10 Pro 64-bit	Administrator account (restricted)
12	UU-12260	Americas	Canada	Internet Explorer 11	64 bits	Office 2010 64-bit	Windows 10 Pro 64-bit	Administrator account (restricted)
13	UU-12260	Americas	United States	Internet Explorer 11	32 bits	Office 365 (2016) 64-bit	Windows 7 Professional 64-bit	Administrator account (elevated privileges)
14	UU-12391	Asia	Singapore	Internet Explorer 11	32 bits	Office 365 (2016) 64-bit	Windows 8.1 Pro 64-bit	Administrator account (restricted)
15	UU-12391	Asia	Singapore	Internet Explorer 11	64 bits	Office 2010 64-bit	Windows 7 Professional 64-bit	Administrator account (elevated privileges)
16	UU-12524	UKI	United Kingdom	Internet Explorer 8	32 bits	Office 2010 64-bit	Windows 7 Enterprise 64-bit	Standard Account
17	UU-12534	Americas	Puerto Rico	Internet Explorer 11	16 bits	Office 2010 32-bit	Windows 10 Pro 64-bit	Administrator account (restricted)
18	UU-12544	Americas	United States	Internet Explorer 9	32 bits	Office 2010 32-bit	Windows 10 Pro 64-bit	Administrator account (elevated privileges)
19	UU-12545	Americas	United States	Internet Explorer 11	32 bits	Office 2010 32-bit	Windows 7 Professional 64-bit	Administrator account (elevated privileges)
20	UU-12545	Americas	United States	Internet Explorer 8	32 bits	Office 2007 SP3	Windows 10 Pro 64-bit	Administrator account (restricted)
21	UU-1292	UKI	United Kingdom	Internet Explorer 10	16 bits	Office 2007 SP3	Windows 10 Pro 64-bit	Administrator account (restricted)
22	UU-1293	UKI	United Kingdom	Internet Explorer 11	32 bits	Office 365 (2016) 64-bit	Windows 10 Pro 64-bit	Administrator account (elevated privileges)
23	UU-13282	Asia	Singapore	Internet Explorer 11	32 bits	Office 2007 SP3	Windows 10 Pro 64-bit	Administrator account (restricted)

Fig. 1. Example of user profile.

2.2. Pairwise Testing

Pairwise testing is a combinatorial testing technique for generating a test suite that is relatively small, while the input values are enormous [16]. Pairwise testing generates test cases which test all combinations by choosing an individual pair of all possible input parameter values. A pairwise test case requires that every pair of input parameters has to be exercised and covered by at least one test case.

Table 1. Example of platform configuration.

Parameter	Parameter Value
Operating System	Windows Mac OS
Web Browser	Mozilla Firefox Internet Explorer Opera Google Chrome
Database	SQL Server Oracle
Microsoft Office	Office 365 32-bit Office 2013 64-bit Office 2010 32-bit

The concept of pairwise testing is best illustrated by an example in Table 1. The example considers a system that intends to operate on a multiple of platforms. The system must function correctly with four parameters on two Operating System (Windows and Mac OS), four Web Browser (Mozilla Firefox, Internet Explorer, Opera, and Google Chrome), two Database (SQL Server and Oracle), and three Microsoft Office (Office 365 32-bit, Office 2013 64-bit, and Office 2010 32-bit). As a result, there are $2 \times 4 \times 2 \times 3 = 48$ possible test platforms in order to test all combinations. However, there are only 12 test platforms covered all the pairwise interactions between different parameters of the system.

Empirical results shown that software defects are caused by a single input parameter or a combination of a few input parameter values [17, 18].

2.3. Constraint Handling

Practical software regularly has constraints between input parameters and parameter values that can be combined into a test case. In software testing, some combinations between input parameters and parameter

values are commonly infeasible and unachievable in practice for executing in the SUT which will yield expensively false positives. Moreover, the pairwise testing technique does not provide a solution for handling the constraints between input parameters and parameter values, the testers have to review the test cases from pairwise testing and delete invalid test cases manually themselves [19]. For example from Table 1, consider that Internet Explorer cannot be executed on Mac OS so the operating system must not be Mac OS. This gives a constraint that Internet Explorer and Mac OS must not appear in any test case into account.

According to Changhai Nie and Hareton Leuang [20], constraints normally happen in most of the software that are caused by some invalid combinations of parameter values. The existence of constraints increases the difficulty in applying combinatorial testing since (1) Most existing test generation methods cannot deal with these constraints, and eventually ignore them all. Due to the fact that ignoring constraints might lead to the generation of the test suite is not valid, impractical test planning, and wasted test effort. (2) It is not easy to design a general algorithm for test generation considering the constraints. (3) Even a few number of constraints may impact an enormous number of invalid configurations. When the generated test suite contains many invalid test cases, this will cause a decreased in combination coverage. (4) Complicated constraints may exist in the SUT, also multiple constraints can interact to generate additional implicit constraints. It is both time consuming and highly error prone to manually deal with constraints in test suite generation. Thus, proper handling of constraints is a key issue we must address in test suite generation. However, it is better than ignoring constraints even the number of test cases might increase or decrease from pairwise testing after constraints have been applied.

Sunint Kaur Khalsa and Yvan Labiche [21] presented a survey of available algorithms and tools for combinatorial testing. They showed that there are four mechanisms for handling constraints (1) Handle constraint before executing a test generation. (2) Replace invalid test cases with valid ones after a test suite has been generated using a specific technique [12]. (3) Integrate constraint handling into the Covering Arrays generation algorithm with an ad-hoc procedure. (4) Integrate the selection of valid tuples by integrating a Satisfiability Solver.

Mats Grindal, Jeff Offutt, and Jonas Mellin [22] presented constraint handling strategies for input parameters models by using combinatorial testing strategies to select test cases. Four new constraint handling methods have been presented. Start with the abstract parameter and sub-models methods result in generate conflict-free input parameter models. The avoid method makes sure that only conflict-free test cases are selected during the test case identification and selection process. Finally, the replace method removes conflicts from selected test cases.

3. Proposed Modeling Approach

This section describes the proposed approach of prioritizing input parameters and parameter values as well as constraints modeling. There are two components for modeling the pairwise testing which are input parameters and parameter values, constraints among input parameters and parameter values. Each component is described separately in the following subsections.

3.1. Input Parameter and Parameter Value Modeling

Identifying input parameters and parameter values is an important step in pairwise testing. It is very difficult to identify optimal input parameters and parameter values for the SUT. It has been established that identifying input parameters and parameter values is a creative process that cannot be completely automated [6]. Different testers would think of various models depends on their creativity and experiences [7]. To our knowledge, there is currently no adequate empirical results, scientific recommendations, or any strong theoretical technique to formulate an optimal set of input parameters and parameter values. Therefore, we propose an approach to identify optimal input parameters and parameter values by using statistical user profile. Using statically user profile has been found as the best ideal approach that would assist testers to set an appropriate priority of testing and ensure that most of the actual usage the SUT has been tested [8, 9].

3.2. Constraints Modeling

Constraints are a truly essential aspect of the SUT because testers can define the explicit exclusion or inclusion on some combinations between input parameters and parameter values for the purpose of customizing and

reducing the size of the generated test suite. For example, if they are invalid, they must be excluded from the generated test suite.

Constraints between input parameters and parameter values must be identified and set by a tester before considering test case generation and after the process of input parameters and parameter values formulation. Constraints can be set as a form of logical expressions and rules. The logical expression and rules define a condition that must be achieved by every test case and must be performed correctness checking during test case generation whether a combination of identified parameters and parameter values or a test case violates the constraints or not [3, 4, 23]. The form of logical expression and rule can be converted from one to another as shown in Table 2.

Table 2. Example of constraint types.

Logical Expression	Term and Rule
AND	AND - connects two parameter and parameter value conditions and returns true only if both conditions are true
OR	OR - connects two parameter and parameter value conditions and returns true if either condition is true or if both conditions are true
=	EQUAL TO – parameter can never be combined with parameter values for all the other parameters
!=	NOT EQUAL TO – parameter can be ever combined with values for the other parameters

4. Tool Implementation

We have focused on the propositional reasoning ability of using statistical user profile prioritization based on input parameters and parameter values for pairwise testing strategy. We have therefore developed a tool that performs constraint handling by using pairwise testing to generate test cases using statistical user profile based prioritization from input parameters and parameter values of our proposed approach, named "User Profile based Pairwise Test Case Generation Tool (UPPTCT)".

Figure 2 illustrates a snapshot of our proposed approach and how our tool works. Our approach and tool involve three steps which are described separately in the following subsections. The number of each subsection corresponds to the number of each step.

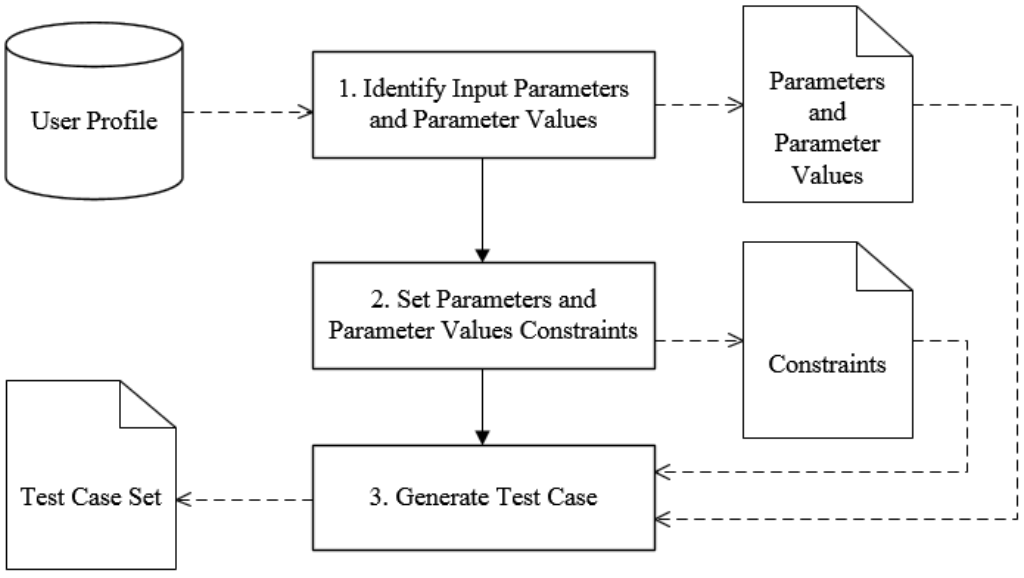

Fig. 2. Overview of our tool.

4.1. Identify Input Parameters and Parameter Values

Input parameters and parameter values identification start with an analysis of the behavior of the SUT. Input parameters and parameter values for generating test case can be analyzed and identified from the user profile which is retrieved from the database and always keeps up to date. In this paper, the user profile requires a form of comma-separated values (CSV) format file type that allows uploading into the tool. The CSV file can be exported from any modern database where each row represents an input parameter value and each column represents an input parameter. The user profile file is an input to the tool for identifying input parameters and parameter values in order to generate test cases.

The imported user profile contains enormous amounts of information. Consequently, all information is not related and relevant to the SUT. In order to properly identify the most important input parameters and parameter values which impact the test focus. A tester is responsible for considering on input parameters and parameter values that are relevant, related, and interested for the SUT.

NO.	SELECT	PARAMETER
1	☐	Region
2	☑	Operating System
3	☑	Browser Version
4	☑	Microsoft Office
5	☑	Color Depth
6	☐	Product Name
7	☑	User Rights

Fig. 3. Example of identified input parameters for SUT.

For example, Fig 3 will be used as running example in the remainder of this paper. It shows input parameters for the SUT to be executed on a multiple of platforms. Five parameters have been identified from the input user profile that are relevant, related, and interested by a tester as it covers the most critical functionalities for the SUT which are Operating System, Browser Version, Microsoft Office, Color Depth, and User Rights.

The identification of an optimal number of parameter values from identified parameters on the most highly usage parameters requires a statistical strategy for analyzing and interpreting. The parameter values of identified input parameters are automatically calculated by the total number and the percentage of users and ordered by the summation of the most highly usage from the input user profile by the tool. This tool could help to guide the identification of the optimal input parameter values. By means of this, the testers can identify the most important values with a high frequency usage characteristics from testing point of view. It also allows testers to know the most appropriate test cases which will be generated and executed. In addition, it indicates the usage characteristics of the SUT statistically. From the identified input parameters for the SUT in Fig 3, the most frequently used is 10 percent and most of the users have been identified and selected as input parameter values by the tester as shown in Fig 4. The result of input parameters and parameter values that have been identified and formulated for the SUT are shown in Table 3.

DOI:10.4186/ej.2017.21.7.389

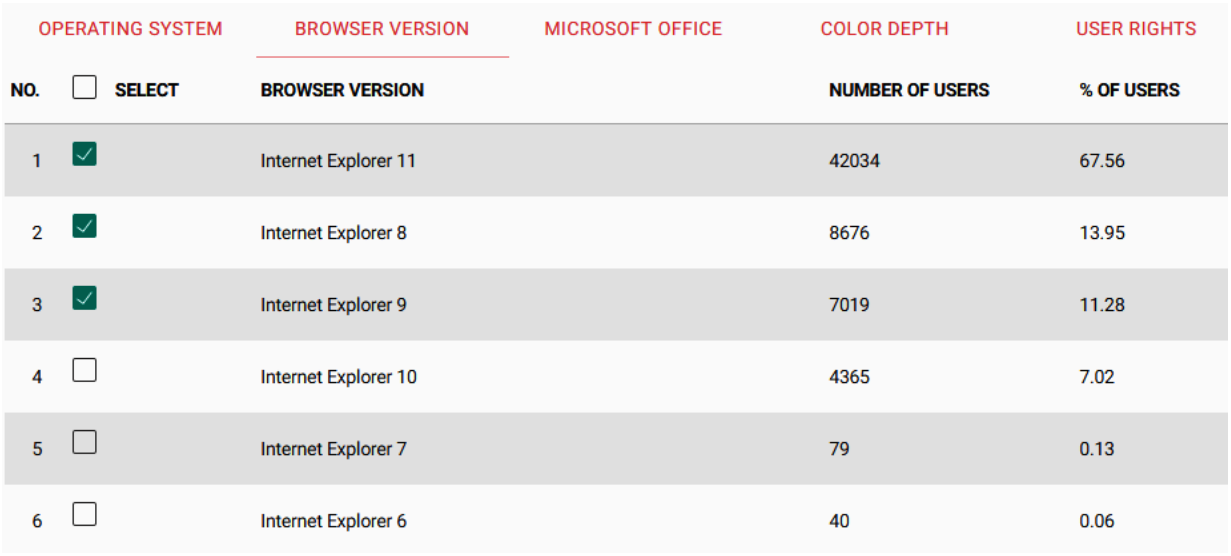

Fig. 4. Example of identified input parameter values for SUT.

Table 3. Identified input parameters and parameter values for SUT.

Parameters	Parameter Values	Number of Users	%
Operating System	Windows 10 Pro 64-bit	25,595	45.96%
	Windows 8.1 Pro 64-bit	11,399	18.32%
	Windows 7 Enterprise 64-bit	6,948	11.17%
	Windows 7 Professional 32-bit	6,363	10.23%
Browser Version	Internet Explorer 11	42,034	67.56%
	Internet Explorer 8	8,676	13.95%
	Internet Explorer 9	7,019	11.28%
Microsoft Office	Office 2010 32-bit	31,639	50.85%
	Office 2013 64-bit	11,255	18.09%
	Office 2013 32-bit	8,011	12.88%
Color Depth	32 bits	60,361	97.02%
User Rights	Standard Account	39,018	62.72%
	Administrator account (restricted)	14,200	22.82%
	Administrator account (elevated privileges)	8,404	13.51%

4.2. Set Parameters and Parameter Values Constraints

Sometimes combination between input parameters and parameter values might not be valid hence it must be excluded from the resulting test suite. Constraints between identified and selected input parameters and parameter values need to be set by the tester before taken the constraints into account and after test case generation for the purpose of handling invalid test cases.

Our tool provides the ability to set constraints between identified and selected input parameters and parameter values that each generated test case must achieve a given set of constraint from the tester. At present, we support two effectively handle different types of constraint specifications which are logical expressions and rules as described in Section 3 and each type can be converted from one to another. There are two features available for constraints handling of the tool. The features are briefly explained as following.

4.2.1. System constraint

Ability to eliminate invalid combination that never happens in practice, misleads results, and any invalid combination must be excluded from the generated test suite. System Constraint can be elicited from business requirements, business rules, or system requirements specifications. Consider the example in Table 3, the combination between Windows 7 Professional 32-bit and Office 2013 64-bit is impossible to occur and must

not be generated as a test case due to the fact that Microsoft Office 64-bit version cannot actually be installed on 32-bit Operating System but it can be installed only on 64-bit Operating System. System Constraint will be identified in form of invalid pair value and it can be separated by entering a new line as shown in Fig. 5.

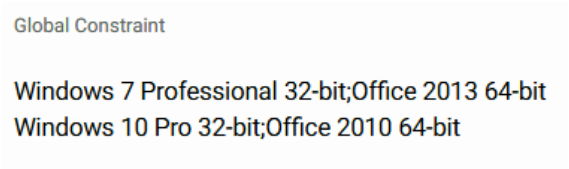

Fig. 5. Example to set system constraint for SUT.

4.2.2. Application constraint

Ability to handle with a specific and complex constraint for a specific SUT between identified input parameters and parameter values. By determining the necessary conditions to generate test cases whether it needed or not in order to decrease the size or customize the resulting test suite. Our tool allows testers to identify the identifying constraint ongoing process before starting test case generation as they may require inclusion or exclusion set of test cases for a specific testing. Considering the example in Fig 4, the most frequently used on operating system and browser is Windows 10 Pro 64-bit and Internet Explorer 11 therefore this test platform is generated as a critical scenario, while Windows 10 Pro 64-bit and Internet Explorer 9 is eliminated test case. Figure 6 illustrates the sample of application constraints for operating system and browser as previously mentioned.

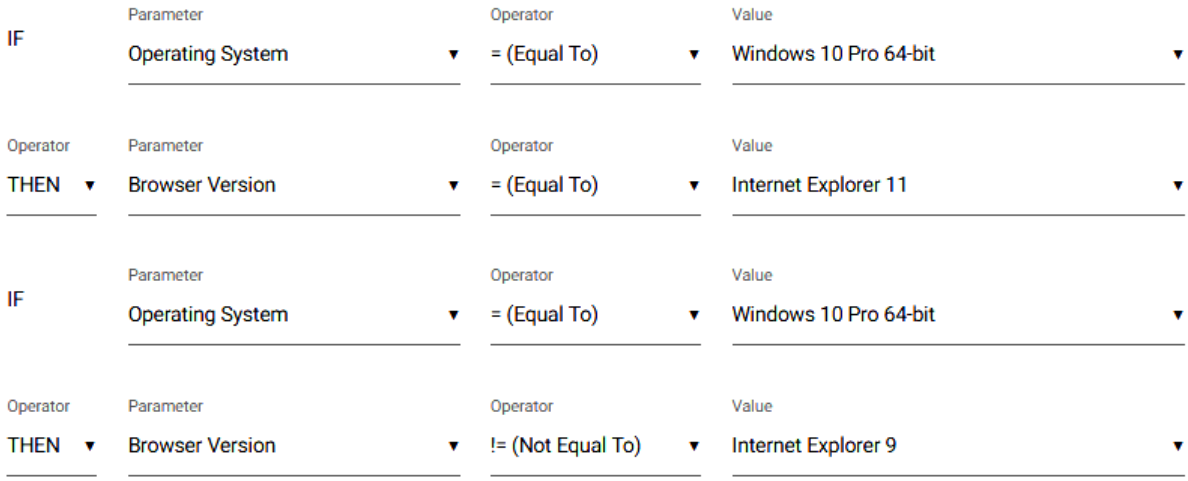

Fig. 6. Example to set application constraints for SUT.

The application constraint will take into account once the test case has been generated using replace technique [12] which causes the resulting test suite satisfies the application constraints and also satisfies pairwise coverage.

4.3. Generate Test Suite

For the step to generate test case from information provided by the tester, our tool gathers and formulates all information including identified input parameters and parameter values and defined constraints for test case generation.

Test case generation begins with the initial combination of two parameters and parameter values, repeats matching the first two parameters and parameter values until all the pair parameter values are covered then starts the next parameter. Then, continuing to perform the next iteration of combination step until all the input parameters and parameter values from Fig. 4 are covered. The resulting test suite consists of infeasible test cases correspond to invalid test case as shown in Fig. 7.

#TC	OPERATING SYSTEM	BROWSER VERSION	MICROSOFT OFFICE	COLOR DEPTH	USER RIGHTS
1	Windows 7 Enterprise 64-bit	Internet Explorer 11	Office 2010 32-bit	32 bits	Standard Account
2	Windows 8.1 Pro 64-bit	Internet Explorer 8	Office 2013 64-bit	32 bits	Administrator account (restricted)
3	Windows 10 Pro 64-bit	Internet Explorer 9	Office 2013 32-bit	32 bits	Administrator account (elevated privileges)
4	Windows 7 Professional 32-bit	Internet Explorer 8	Office 2010 32-bit	32 bits	Administrator account (restricted)
5	Windows 8.1 Pro 64-bit	Internet Explorer 9	Office 2013 64-bit	32 bits	Administrator account (elevated privileges)
6	Windows 10 Pro 64-bit	Internet Explorer 11	Office 2013 32-bit	32 bits	Standard Account
7	Windows 7 Professional 32-bit	Internet Explorer 9	Office 2010 32-bit	32 bits	Administrator account (elevated privileges)
8	Windows 7 Enterprise 64-bit	Internet Explorer 11	Office 2013 64-bit	32 bits	Standard Account
9	Windows 10 Pro 64-bit	Internet Explorer 8	Office 2013 32-bit	32 bits	Administrator account (restricted)

Fig. 7. A test suite obtained from pairwise without constraint handling.

Considering Fig. 7, there are two invalid test cases in the resulting test suite which are #TC3 and #TC9. Those test cases are not possible to execute on the given SUT and need to be manipulated by applying identified constraints into account.

Our tool is developed based on the replace mechanism. It builds on the concept of invalid test cases to be replaced after the test suite has been generated completely to maintain the test coverage of the test suite. To verify generated test suite, we perform correctness checking to ensure if all the identified constraints are satisfied. Searching for the invalid test case in the test suite, the invalid test case is cloned if found. In each clone, one or more of the identified parameter values which violate one or more constraints will be changed to another identified parameter value that is removed as the invalid test case. Then, continuing to perform the next iteration on replacing step until all of the generated test cases are covered. Moreover, if the invalid identified parameter values work on the same test case, the invalid test case will be removed right away to ensure termination. The resulting of the final test suite that is applied constraints handling as shown in Fig. 8.

#TC	OPERATING SYSTEM	BROWSER VERSION	MICROSOFT OFFICE	COLOR DEPTH	USER RIGHTS
1	Windows 7 Enterprise 64-bit	Internet Explorer 11	Office 2010 32-bit	32 bits	Standard Account
2	Windows 8.1 Pro 64-bit	Internet Explorer 8	Office 2013 64-bit	32 bits	Administrator account (restricted)
3	Windows 7 Professional 32-bit	Internet Explorer 8	Office 2010 32-bit	32 bits	Administrator account (restricted)
4	Windows 8.1 Pro 64-bit	Internet Explorer 9	Office 2013 64-bit	32 bits	Administrator account (elevated privileges)
5	Windows 10 Pro 64-bit	Internet Explorer 11	Office 2013 32-bit	32 bits	Standard Account
6	Windows 7 Professional 32-bit	Internet Explorer 9	Office 2010 32-bit	32 bits	Administrator account (elevated privileges)
7	Windows 7 Enterprise 64-bit	Internet Explorer 11	Office 2013 64-bit	32 bits	Standard Account
8	Windows 7 Enterprise 64-bit	Internet Explorer 11	Office 2013 32-bit	32 bits	Administrator account (elevated privileges)
9	Windows 7 Enterprise 64-bit	Internet Explorer 11	Office 2013 32-bit	32 bits	Administrator account (restricted)

Fig. 8. A final test suite generated from pairwise with constraint handling.

5. Experimental Results

Our experiments consist of two parts. The first part is to compare our tool with other known pairwise test generation and constraints handling tools. The second part is designed to evaluate the test case effectiveness.

5.1. Comparison with Other Tools

In this section, we compare our tool with other three existing well-known representative test case generation tools. We first compare our tool with constraints and without constraints. The comparison helps us to see that our tool works as expected concerning other tools and strategies. Regarding our tool is implemented based on constraints handling, it is important to note that some of the existing tools are not available to perform the experiment and existing work does not compare the number of generated test cases with constraints handling. It is very difficult to compare our tool with existing available tools as it relies on the number of input parameters and parameter values. We briefly introduce the existing test case generation tools with constraint handling supported we use.

ACTS [24] or Advanced Combinatorial Testing System is a combinatorial test generation research tool developed by National Institute of Standards and Technology (NIST) and University of Texas at Arlington. No license is required and there are no restrictions on distribution or using the tool. ACTS generates test sets in t-way coverage of input parameter values and supports constraints and variable-strength tests.

PICT [25] or Pairwise Independent Combinatorial Testing is a pairwise test generation reach tool developed by Microsoft Corporation. A free pairwise testing tool runs as a command line and support constraints handling.

CTWeb [26] is a pairwise test generation research tool developed based on PROW algorithm (Pairwise with constRaints, Order, and Weight).

Our comparison uses 7 examples of different input parameters and parameter values and constraints taken from the available published literatures. The number of generated test cases with constraints and without constraints for each example is shown in Fig. 9.

#	Example System	Input Parameters and Values	Constraints	Number of TC Without Constraints				Number of TC With Constraints			
				ACTS	PICT	PROW	UPPTCT	ACTS	PICT	PROW	UPPTCT
1	Basic Billing System [27]	3^4	2	9	9	9	9	12	11	11	12
2	Cruise Control System [27]	$4^1\ 3^1\ 2^4$	2	12	12	14	12	14	12	12	12
3	Simplified-Pizza-Ordering System [28]	$2^3\ 3^2$	3	9	9	12	9	9	7	12	13
4	Hotel Guide System [22]	$3^3\ 4^1$	3	13	12	15	12	13	13	15	16
5	Configuration Problems [29]	$3^1\ 4^2\ 2^1$	5	16	16	19	16	16	15	15	20
6	Online PC Retailing System [30]	$4^4\ 2^5\ 3^2$	6	20	27	30	16	22	27	31	16
7	Messaging System [27]	$3^4\ 2^2\ 3^1$	6	19	21	27	16	20	19	30	10

Fig. 9. Comparison of other tools.

From the result in Fig. 9, we can observe that our tool, UPPTCT, performs superior in some cases and perform equivalent results for generating test cases from input parameters and parameter values with constraints handling and without constraints handling. The outperformer on both with and without constraints cases are Online PC Retailing System and Messaging System which is approximately 28.57%.

5.2. Evaluation of the Test Case Effectiveness

In this section, we evaluate the defect detection ability of the generated test suite from our tool. To compare test case effectiveness from our approach and random testing, we used the same input parameters and parameter values of the SUT obtained from 62,214 records of the real user profile.

We use the real production defects that had been logged over the past 6 months after released a new version of the product as a representative, there were 958 defects reported from the customers. We analyzed, classified, and enumerated defects that have interactions cause based on the defect description. We found that 211 defects reported in total are interaction defects between one or more parameters and can be categorized into 4 groups of interaction as shown in Table 4.

DOI:10.4186/ej.2017.21.7.389

Table 4. Defects distribution and classification.

#	Interaction Group	Interaction Defects Reported
1	OS and Internet Explorer	22
2	OS and Microsoft Office	33
3	OS and Antivirus	18
4	Application and Web Browser	138
	Total	**211**

We use the same number of test suite obtained from our tool comparing with 5 random testing test suites. The numbers of test cases in random testing are the same numbers of test cases generated by our tool. We selected the most frequently used at more than 10%, 20%, 30%, 40%, and all users respectively as a set of extrapolated from the user profile. It is necessary to mention that the real user profile we use contain maximum value generated test cases which are 40% of users as shown in Fig. 10.

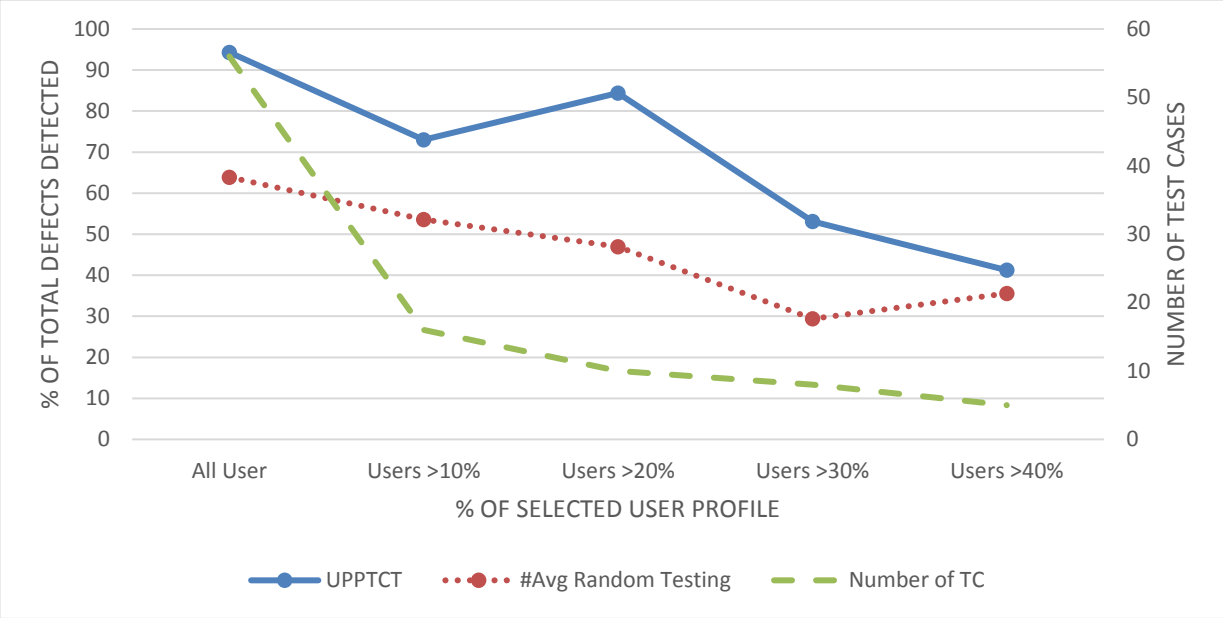

Fig. 10. Comparison of defects detection.

Preliminary findings from our results in Fig. 10 shows that the effectiveness of our approach is significantly higher than random testing. A large portion of reported defects being caught with regard to statically user profile by our approach comparing with test case generated from random testing. We see evidence of correlations between the user profile and defects as the majority of reported defects from the customers which are predicted to be the most frequency used obtained from the user profile statistically. With a high frequency usage from the user profile turns out to be really important as a guideline for test case generation, the test cases can be generated straightforward corresponding to the statistical user profile that could be detected the majority of defects.

Our further analysis demonstrates the ability to detect defects depends on the range of selected user profile for generating test cases. It can be seen that the number of generated test case decreases markedly, at the lower range of selected user profile. As a result, there is a significantly strong relationship between the number of selected user profile, the number of generated test cases, and the number of defects detection.

Similar results of additional research about applying statistical user profile have been conducted. Applying statistical user profile approach is exceptionally valuable in improving the software quality and reliability in reduce testing efforts and also reduce the software development cost [8, 12]. Moreover, implementing the automation to collect the user profile would help reduce in human errors as the user profiles are the most important part of the approach and therefore require special attention from testers [9, 13].

6. Conclusion and Future Work

In this paper, we have presented the approach and tool to prioritize input parameters and parameter values modeling based on statistically user profile and also provided the ability for constraints handling on input parameters and parameter values when using pairwise testing to generate a test suite. The process of identifying input parameters and parameter values and constraints defined from a user profile still requires testers to get involved in and provide decision making for determining the percentage of the combinations between identified input parameters and parameter values.

At the most frequently used of the user profile turns out to be the most important in guiding testing the SUT that could help detect the majority of defects. Therefore, if our approach is used, the number of defects escape to the production environment and found by customers will decline. Our analysis shows that our approach can help testers focus on what important for the SUT is, and allows testers to provide decision makers with statistically defensible options based on an understanding of the statically user profile for testing the specific technical interest and user perspective. Using a high percentage of the most frequency usage from the user profile helps the tester on identifying and selecting the optimal input parameters and parameter values to generate test cases by gathering high frequency used from user profile statically. The most frequently used has been tested the most and is more likely to capture defects with a comparatively small number of test cases than other testing strategies.

There are several directions for the future work. We would expand an experiment with a large scale of a user profile in the real world system. Ultimately, we would like to apply other data on user profiles for prioritizing input parameters and parameter values instead of using only frequency usage as a statistical approach.

References

[1] M. Utting and B. Legeard, *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann, 2010.

[2] A. Javeed and C. Yilmaz, "Combinatorial interaction testing of tangled configuration options," in *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference* on, 2015, pp. 1-4.

[3] S. Gao, B. Du, Y. Jiang, J. Lv, and S. Ma, "An efficient algorithm for pairwise test case generation in presence of constraints," in *Systems and Informatics (ICSAI), 2014 2nd International Conference* on, 2014, pp. 406-410.

[4] S. Nakornburi and T. Suwannasart, "A tool for constrained pairwise test case generation using statistical user profile based prioritization," in *Computer Science and Software Engineering (JCSSE), 2016 13th International Joint Conference on*, IEEE, 2016, pp. 1-6.

[5] Li, X., Gao, R., Wong, W. E., Yang, C., & Li, D, "Applying combinatorial testing in industrial settings," in *Software Quality, Reliability and Security (QRS), 2016 IEEE International Conference on*, IEEE, August 2016, pp. 53-60.

[6] M. Grochtmann and K. Grimm, "Classification trees for partition testing," *Software Testing, Verification and Reliability*, vol. 3, pp. 63-82, 1993.

[7] M. N. Borazjany, L. Yu, Y. Lei, R. Kacker, and R. Kuhn, "Combinatorial testing of ACTS: A case study," in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, 2012, pp. 591-600.

[8] D. K. Yadav, "Operational profile based software test case allocation," in *Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference* on, 2015, pp. 1775-1779.

[9] S. Herbold, J. Grabowski, and S. Waack, "A model for usage-based testing of event-driven software," in *Secure Software Integration & Reliability Improvement Companion (SSIRI-C), 2011 5th International Conference* on, 2011, pp. 172-178.

[10] (2016, Aug. 21). *User Profiles* [Online]. Available: http://www.mantisbt.org/manual/admin.user.profiles.html

[11] (2016, Aug. 21). *What is a User Profile? - Definition from Techopedia* [Online]. Available: https://www.techopedia.com/definition/16137/user-profile

[12] H. Koziolek, "Operational profiles for software reliability," in *Seminar on Dependability Engineering*, Germany, Citeseer, 2005.

[13] T. Takagi and Z. Furukawa, "Construction technique of large operational profiles for statistical software testing," in *Computer and Information Science*. Springer, 2013, pp. 187-199.

[14] J. D. Musa, "The operational profile," in *Reliability and Maintenance of Complex Systems*. Springer, 1996, pp. 333-344.
[15] J. D. Musa, "Operational profiles in software-reliability engineering." *IEEE Software*, vol. 10 no. 2, pp. 14-32, 1993.
[16] W. A. Ballance, S. Vilkomir, and W. Jenkins, "Effectiveness of pair-wise testing for software with Boolean inputs," in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, 2012, pp. 580-586.
[17] D. R. Kuhn, R. N. Kacker, and Y. Lei, "Sp 800-142. Practical combinatorial testing," National Institute of Standards and Technology, Technology Administration, US Department of Commerce, 2010.
[18] C. B. Monteiro, L. A. V. Dias, and A. M. da Cunha, "A case study on pairwise testing application," in *Information Technology: New Generations (ITNG), 2014 11th International Conference* on, 2014, pp. 639-640.
[19] S. Nakornburi and T. Suwannasart, "Constrained pairwise test case generation approach based on statistical user profile," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, 2016, pp.445-448.
[20] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Computing Surveys (CSUR)*, vol. 43, p. 11, 2011.
[21] S. K. Khalsa and Y. Labiche, "An orchestrated survey of available algorithms and tools for combinatorial testing," in *2014 IEEE 25th International Symposium on Software Reliability Engineering*, 2014, pp. 323-334.
[22] M. Grindal, J. Offutt, and J. Mellin, "Handling constraints in the input space when using combination strategies for software testing," School of Humanities and Informatics, University of Skövde, Technical Report HS-IKI-TR-06-001, 2006.
[23] L. Yu, Y. Lei, M. Nourozborazjany, R. N. Kacker, and D. R. Kuhn, "An efficient algorithm for constraint handling in combinatorial test generation," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, 2013, pp. 242-251.
[24] Csrc.nist.gov. (2016, August 21). *NIST Computer Security Division - Automated Combinatorial Testing for Software (ACTS)* [Online]. Available: http://csrc.nist.gov/groups/SNS/acts/
[25] J. Czerwonka, "Pairwise testing in the real world: Practical extensions to test-case scenarios," in *Proceedings of 24th Pacific Northwest Software Quality Conference*, Citeseer, 2006, pp. 419-430.
[26] B. Pérez Lamancha and M. Polo. (2016, April 17). *Automated test Case Generation with Combinatorial Techniques* [Online]. Available: http://alarcosj.esi.uclm.es/CombTestWeb/combinatorial.js
[27] C. Lott, A. Jain, and S. Dalal, "Modeling requirements for combinatorial software testing," in *ACM SIGSOFT Software Engineering Notes*, 2005, pp. 1-7.
[28] A. Alsewari, K. Zamli, and B. Al-Kazemi, "Generating t-way test suite in the presence of constraints," *Journal of Engineering and Technology (JET)*, vol. 6, pp. 52-66, 2015.
[29] B. P. Lamancha, M. Polo, and M. Piattini, "PROW: A Pairwise algorithm with constRaints, Order and Weight," *Journal of Systems and Software*, vol. 99, pp. 1-19, 2015.
[30] Y. Zhao, Z. Zhang, J. Yan, and J. Zhang, "Cascade: a test generation tool for combinatorial testing," in *Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference* on, 2013, pp. 267-270.