Constrained Pairwise Test Case Generation Approach based on Statistical User Profile

Sompong Nakornburi and Taratip Suwannasart

***Abstract*— Pairwise testing which is the most used of combinatorial strategy has been shown to be a very effective testing technique. An important problem in pairwise testing deals with constraints and selects the optimal input parameters and values. This paper proposes an approach to select an optimal set of input parameters and values using statistical user profile for pairwise test case generation and to apply constraint handling to deal with unrealistic combinations.**

***Index Terms*—pairwise testing, combinatorial testing, software testing, test cases generation, test case selection**

I. Introduction

Software testing is an important activity in software development process to assure product value delivered to customers. Moreover, software testing generally consumes between 30 and 60 percent of overall development effort [1].

In the past few years, an increasing trend on software system shown that it usually operates in a very complex environment. Exhaustive testing in a software system is practically impossible for real-world software that requires extensive efforts and enormous resources [2], [3]. Therefore, testing techniques have been researched and adopted on software testing.

One of the most used combinatorial techniques is pairwise testing, a type of test case selection method which test cases are created by the combinations of interesting values previously identified by a tester [4]. Pairwise testing has been proven are very effective in fault detection with less costs and aims at testing numerous combinations of inputs with reducing set of test cases. Pairwise testing considers only combinations of all possible pairs [5]. However, practical system testing often has constraints on the combination of parameter values. As a result, some combinations of parameter values are frequently invalid.

Each individual tester would come up with different models depends on creativity and experience [6]. Input parameters and values are essential steps in pairwise testing. However, there is currently no adequate scientific recommendations or any solid theoretical method to select an optimal set of input parameters and values [7]. Consequently, the goal of this paper is to make a step toward filling such that gap.

Manuscript received November 09, 2015.

S. Nakornburi and T. Suwannasart are with the Software Engineering Laboratory Center of Excellence in Software Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok, Thailand (e-mail: Sompong.N@student.chula.ac.th, Taratip.S@chula.ac.th).

In this paper we propose an approach to generate test cases for pairwise testing by applying user profile for guiding and prioritizing input parameters and values to be selected and also providing solution for the constraints handling between parameters and values.

The remainder of this paper is structured as follows: Section II briefly reviews the background on pairwise testing, constraint handling, and user profile. Section III describes our proposed approach of constraint handling in pairwise test case generation using statistical user profile to select input parameters and values. Finally, section IV provides conclusion and a plan for future work.

II. Background

A. Pairwise Testing

Pairwise (2-way combination) testing is a combinatorial technique which aims at testing all possible numerous combinations set of input parameters. Each pair of values of any two parameters is covered by at least one test case [5], [8]. Empirical results show that software defects are triggered by a single input parameters or a combination of two input parameters [3], [5], [9].

To illustrate the concept of pairwise testing, consider a system with three parameters A, B and C. Each of the first two parameters consist of two values {A1, A2}, {B1, B2} and the third parameter consist of three values {C1, C2, C3} respectively. This will be end up with 16 different pairs in total. In order to test all combinations would need $2 \times 2 \times 3 = 12$ test cases. Furthermore, a generated pairwise test case set includes of 6 test cases covered in all parameters participated in a particular parameter interactions for this scenario.

B. Constraint Handling

In software testing, some combination of parameters and values are frequently invalid and untestable because they do not exist for the system under test. The pairwise testing technique does not handle the constraints between input parameters and values, the tester must review the results obtained from pairwise testing and manually delete bad pair test cases themselves.

Changhai Nie and Hareton Leuang [2] claim that the existence of constraints increases the difficulty in applying combinatorial testing due to the fact that (1) Most existing test generation methods cannot deal with any constraints, and eventually ignore them. Ignoring constraints might lead

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

to the generation of test configurations that is invalid. Resulting in ineffective test planning and wasted test effort. (2) It is difficult to design a general algorithm for test generation with consideration of constraints. (3) Even a small number of constraints may give rise to an enormous number of invalid configurations. When the generated test suite contains many invalid test cases, this will cause a loss of combination coverage. (4) Complicated constraints may exist in the system under test, and multiple constraints can interact to produce 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.

More or less in the number of test cases may be obtained for pairwise testing after applied constraints handling but it is better than those obtained without taking constraints into account.

Constraints must be specified by a tester before test generation. Constraints can be specified as a set of logical expression. A logical expression describes a condition that satisfies by every single test and is used to perform validity check during test generation [10], [11].

Mats Grindal, Jeff Offutt, and Jonas Mellin [12] present constraint handling methods in input parameters models when using combination strategies to select test cases. One of handling constraint is the replace method, bad pair in test cases can be resolved after the test case set has been generated and preserves the coverage of the test case set. Instead of removing bad pair test cases, the bad pair test cases are cloned, in each clone one or more of the parameter values involved in the conflict is changed to an arbitrary value that can remove the bad pair. With the replace method, almost as simple, reduces the final test cases, completely general the combination strategy, and short execution time.

C. *User Profile*

A user profile describes an environment and a collection of settings that the user uses to run the software, which are being tracked. The profile information includes platform, operating system, software, hardware, and additional description, for example [13], [14].

In this paper, we describe a user profile as a collection of settings and information associated with a user and the software under test. It can be defined as the identity of the user considering to the operating environment, which could be platform, operating system, software applications, or hardware information. User profile must be collected into the system for analyzing and prioritizing test cases.

With a user profile, a quantitative characterization of the way a system can be tested more efficiently because testing can focus on how users will employ the product and the relative importance of different uses. It is a practical approach to ensure that a system delivered with a maximized reliability because the operation most used also has been tested the most [15].

III. PROPOSED APPROACH

This section describes the proposed approach of constrained pairwise test case generation based on statistical user profile. The whole process of constructing consists of four steps:

1) Select Input Parameters and Values
2) Define Parameters and Constraint Values
3) Generate Test Case
4) Execute Test Case Set

Fig. 1 shows the schematic representation of our proposed approach. The four steps of the approach are described separately in the following subsections.

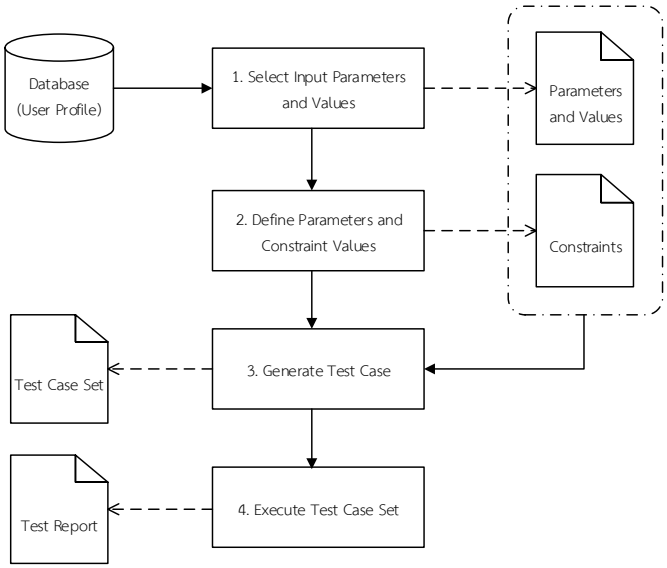

Fig. 1. Constrained pairwise test case generation based on statistical user profile approach

A. *Select Input Parameters and Values*

The input parameters and values for generating test case to be selected based on the user profile which are collected from the database. In this paper, the user profile must be in a comma-separated values (CSV) format file. CSV files can be exported from any modern database. For example, suppose we have a user profile intended to execute on a variety of platforms shown in Fig. 2.

	A	B	C	D	E	F
1	**Region**	**Country**	**Operating System**	**Browser Version**	**Microsoft Office**	**Antivirus**
2	Americas	United States	Windows 7 Enterprise 64-bit	Internet Explorer 11	Office 365 (2013) 32-bit	McAfee VirusScan Enterprise
3	Americas	United States	Windows 7 Enterprise 64-bit	Internet Explorer 11	Office 2013 32-bit	McAfee VirusScan Enterprise
4	UKI	United Kingdom	Windows 7 Professional 64-bit	Internet Explorer 11	Office 365 (2013) 32-bit	McAfee VirusScan Enterprise
5	Americas	Canada	Windows 7 Professional 64-bit	Internet Explorer 11	Office 2010 SP2 32-bit	Symantec Endpoint Protection
6	Americas	United States	Windows 7 Professional 64-bit	Internet Explorer 10	Office 2007 SP3	Symantec Endpoint Protection
7	UKI	United Kingdom	Windows 7 Professional 64-bit	Internet Explorer 11	Office 2013 32-bit	Symantec Endpoint Protection
8	UKI	United Kingdom	Windows 7 Home Premium 64-bit	Internet Explorer 11	Office 2013 32-bit	McAfee VirusScan Enterprise
9	Asia	Singapore	Windows 7 Professional 64-bit	Internet Explorer 11	Office 2013 32-bit	ESET Endpoint Security 5.0
10	Asia	Singapore	Windows 7 Professional 64-bit	Internet Explorer 10	Office 2013 32-bit	ESET Endpoint Security 5.0
11	Asia	Singapore	Windows 7 Home Premium 64-bit	Internet Explorer 10	Office 2013 32-bit	McAfee VirusScan Enterprise
12	Asia	Singapore	Windows 7 Home Premium 64-bit	Internet Explorer 11	Office 2010 SP1 32-bit	McAfee VirusScan Enterprise
13	Asia	Singapore	Windows 7 Professional 64-bit	Internet Explorer 11	Office 2010 SP2 32-bit	AhnLab V3 Internet Security 8.0
14	Asia	Singapore	Windows 7 Professional 64-bit	Internet Explorer 11	Office 2010 SP2 32-bit	AhnLab V3 Internet Security 8.0
15	Asia	Singapore	Windows 7 Ultimate 32-bit	Internet Explorer 9	Office 2007 SP3	McAfee VirusScan Enterprise
16	CEMA	Belgium	Windows 7 Professional 64-bit	Internet Explorer 9	Office 2007 SP3	Symantec Endpoint Protection
17	CEMA	Germany	Windows 7 Professional 64-bit	Internet Explorer 11	Office 2010 SP2 32-bit	Trend Micro OfficeScan Antivirus
18	CEMA	Germany	Windows 7 Professional 64-bit	Internet Explorer 9	Office 2010 SP2 32-bit	Trend Micro OfficeScan Antivirus
19	CEMA	Switzerland	Windows 7 Enterprise 64-bit	Internet Explorer 11	Office 2013 32-bit	Symantec Endpoint Protection
20	UKI	United Kingdom	Windows 7 Enterprise 64-bit	Internet Explorer 11	Office 2013 32-bit	Trend Micro OfficeScan Antivirus
21	CEMA	Germany	Windows 7 Home Premium 64-bit	Internet Explorer 9	Office 365 (2013) 32-bit	AVG Anti-Virus Free Edition 2014
22	Asia	Singapore	Windows 7 Professional 32-bit	Internet Explorer 9	Office 2010 SP2 32-bit	AVG Anti-Virus Free Edition 2014
23	UKI	United Kingdom	Windows 8.1 64-bit	Internet Explorer 8	Office 2013 64-bit	Trend Micro OfficeScan Antivirus
24	CEMA	Germany	Windows 7 Enterprise 64-bit	Internet Explorer 9	Office 2010 SP2 32-bit	Trend Micro OfficeScan Antivirus
25	CEMA	Germany	Windows 7 Professional 64-bit	Internet Explorer 11	Office 2010 SP2 32-bit	AhnLab V3 Internet Security 8.0

Fig. 2. A sample of user profile from a database

The user profile comprises of a lot of information as it is simply impractical to test a system that is intended to run on a variety of platforms. Thus, testers need to consider only on parameters and values that are related and interested in order

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

to cover the most important parameters and values which impact the test focus. For example, we select Operating System, Browser, Microsoft Office, and Antivirus as input parameters.

The selection of optimal number of values on frequently used parameters requires statistical method for analyzing and interpreting. The values of input parameters are automatically ordered using the summation of the most frequently used by a tool as it can be used to guide the selection of input values. With this, the tester can specify the most important values with a high occurrence usage from testing perspective. It also allows testers to know the most valuable test cases to be generated and executed that reflect the usage characteristics statistically.

From the user profile in Fig. 2., supposing that the most frequently used at least 10 percent of users were selected and formulated as input parameters and values by the tester, which are shown in Table I.

TABLE I
A SAMPLE OF SELECTED PARAMETERS AND VALUES

No.	Operating System	Users	%
1	Windows 7 Enterprise 64-bit	58,991	38
2	Windows 7 Professional 32-bit	25,328	16
No.	**Web Browser**	**Users**	**%**
1	Internet Explorer 10	58,949	38
2	Internet Explorer 11	50,057	32
3	Internet Explorer 9	19,258	12
4	Internet Explorer 8	17,207	11
No.	**Microsoft Office**	**Users**	**%**
1	Office 2010 SP2 32-bit	21,191	37
2	Office 2007 SP3	8,811	16
3	Office 2013 64-bit	7,613	13
No.	**Antivirus**	**Users**	**%**
1	McAfee VirusScan Enterprise	2,735	31
2	Symantec Endpoint Protection	2,264	28

B. Define Parameters and Constraint Values

Some combinations are not valid from the selected parameters and values from previous subsection in Table I that will generate bad pair test cases and must be excluded from the result of test case set. Constraints among selected parameters and values need to be specified by testers before taken into consideration during test generation in order to avoid bad pair test cases when generating test case set.

Currently, we support a collection of terms and rules that come with Natural Language of logical terms and Boolean type of constraints in order to have a meaningful constraint. Table I shows the possible input parameters and values of a platform that could be executed for Operating System (OS), Browser (BROWSER), Microsoft Office (OFFICE), and Antivirus (ANTIVIRUS). A test case set generated does not take constraints into account so the result of test case set will cover combinations including bad pair test cases. Bad pair test cases combine 32-bits Operating System with 64-bits Microsoft Office, or Operating System with Internet Explorer. In the Table II, we give a formal syntax of the expressions that can be used to specify a constraint and ensure that the selected parameters and values in Table I are not generated.

To set constraints between input parameters and values can be set as required or exclude from the final test case set.

TABLE II
A SAMPLE OF DEFINED CONSTRAINTS

#	Constraint	Description
1	IF (OS = “32-bits”) THEN (OFFICE != “64-bits”)	If Operating System is 32-bits, then Microsoft Office must not be 64-bits
2	IF (Browser = “IE”) THEN (OS != “Mac”)	If Browser is Internet Explorer, then Operating System must not be Macintosh
3	IF (OS = “Windows8”) THEN (Browser != “IE7”)	If Operating System is Windows 8, then Browser must not be Internet Explorer 7
4	IF (OS = “Windows10”) THEN (Browser = “IE11”)	If Operating System is Windows 10, then Browser must be Internet Explorer 11

C. Generate Test Case

We first generate a pairwise test case for the first two parameters until cover all values then extend it to another parameter. Starting the next iteration on this step until we cover all the parameters and values from Table I the resulting test case set as shown in Fig. 3.

The highlighted row in Fig. 3., represents to bad pair test cases. In the resulting test case set, there are two bad pair test cases: TC3 and TC9 contain Office 2013 64-bit on Operation System 32-bit version. These test cases are infeasible and need to be managed by taking the constraints into account as defined by the tester in previous subsection in order to generate feasible test case set.

#TC	OS	BROWSER	OFFICE	ANTIVIRUS
1	Windows 7 Professional 32-bit	8	Office 2010 SP2 32-bit	Symantec Endpoint Protection
2	Windows 7 Enterprise 64-bit	8	Office 2007 SP3	McAfee VirusScan Enterprise
3	Windows 7 Professional 32-bit	8	Office 2013 64-bit	McAfee VirusScan Enterprise
4	Windows 7 Enterprise 64-bit	9	Office 2010 SP2 32-bit	Symantec Endpoint Protection
5	Windows 7 Professional 32-bit	9	Office 2007 SP3	McAfee VirusScan Enterprise
6	Windows 7 Enterprise 64-bit	9	Office 2013 64-bit	Symantec Endpoint Protection
7	Windows 7 Professional 32-bit	10	Office 2010 SP2 32-bit	McAfee VirusScan Enterprise
8	Windows 7 Enterprise 64-bit	10	Office 2007 SP3	Symantec Endpoint Protection
9	Windows 7 Professional 32-bit	10	Office 2013 64-bit	Symantec Endpoint Protection
10	Windows 7 Professional 32-bit	11	Office 2010 SP2 32-bit	McAfee VirusScan Enterprise
11	Windows 7 Enterprise 64-bit	11	Office 2007 SP3	Symantec Endpoint Protection
12	Windows 7 Enterprise 64-bit	11	Office 2013 64-bit	Symantec Endpoint Protection

Fig. 3. Test cases obtained using pairwise without constraints

Consider the test case set that is satisfied pairwise coverage, we perform validity check by using a replace method to determine whether all the constraints are satisfied by a test case with the specified constraints from Table II. If a bad pair test case is detected, that test case will be cloned. In each clone, one or more of the parameter values involved in the bad pair is changed to another value that removes the bad pair. Then, the next iteration on this step is started until we cover all generated test cases in Fig. 3.

The number of clones is chosen such that the number of

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

test case is preserved. The resulting final test case set is cover combinations that satisfy those constraints as shown in Fig. 4.

#TC	OS	BROWSER	OFFICE	ANTIVIRUS
1	Windows 7 Professional 32-bit	8	Office 2010 SP2 32-bit	Symantec Endpoint Protection
2	Windows 7 Enterprise 64-bit	8	Office 2007 SP3	McAfee VirusScan Enterprise
3	Windows 7 Enterprise 64-bit	8	Office 2013 64-bit	Symantec Endpoint Protection
4	Windows 7 Enterprise 64-bit	9	Office 2010 SP2 32-bit	McAfee VirusScan Enterprise
5	Windows 7 Professional 32-bit	9	Office 2007 SP3	Symantec Endpoint Protection
6	Windows 7 Enterprise 64-bit	9	Office 2013 64-bit	McAfee VirusScan Enterprise
7	Windows 7 Professional 32-bit	10	Office 2010 SP2 32-bit	McAfee VirusScan Enterprise
8	Windows 7 Enterprise 64-bit	10	Office 2007 SP3	Symantec Endpoint Protection
9	Windows 7 Enterprise 64-bit	10	Office 2013 64-bit	Symantec Endpoint Protection
10	Windows 7 Professional 32-bit	11	Office 2010 SP2 32-bit	McAfee VirusScan Enterprise
11	Windows 7 Enterprise 64-bit	11	Office 2007 SP3	Symantec Endpoint Protection
12	Windows 7 Enterprise 64-bit	11	Office 2013 64-bit	Symantec Endpoint Protection

Fig. 4. Final test case set obtained using pairwise with constraints

D. *Execute Test Case Set*

To evaluate the proposed approach, we consider the number of parameters due to the fact that each parameter usually has different number of values and compare the number of test cases generated with and without constraints with existing algorithms for Combinatorial Testing [2].

IV. Conclusion and Future Work

In this paper we have presented the proposed approach to generate test case for pairwise testing based on user profile, a goal oriented testing that guides and prioritizes testing to the most used of the system under test statistically and allows the tester to handle constraints to avoids bad pair test cases.

There are several directions to continue our work. First, we want to implement a tool from the proposed approach. Second, we want to conduct similar studies of the real-world application to evaluate the effeteness for our approach. Finally, we plan to investigate how to integrate our approach with other automate test execution.

References

[1] M. Utting and B. Legeard, *Practical model-based testing: a tools approach*: Morgan Kaufmann, 2010.

[2] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Comput. Surv.*, vol. 43, pp. 1-29, 2011.

[3] D. R. Kuhn, R. N. Kacker, and Y. Lei, "SP 800-142. Practical Combinatorial Testing," National Institute of Standards \& Technology2010.

[4] B. Pérez 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, 1// 2015.

[5] C. B. A. L. Monteiro, L. A. Vieira 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.

[6] M. N. Borazjany, Y. Linbin, L. Yu, R. Kacker, and R. Kuhn, "Combinatorial Testing of ACTS: A Case Study," in *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*, 2012, pp. 591-600.

[7] S. Vilkomir and B. Amstutz, "Using Combinatorial Approaches for Testing Mobile Applications," in *Software Testing, Verification and Validation Workshops (ICSTW), 2014 IEEE Seventh International Conference on*, 2014, pp. 78-83.

[8] J. Bach and P. J. Schroeder, "Pairwise testing: A best practice that isn't," in *Proceedings of 22nd Pacific Northwest Software Quality Conference*, 2004, pp. 180-196.

[9] P. Flores and C. Yoonsik, "PWiseGen: Generating test cases for pairwise testing using genetic algorithms," in *Computer Science and Automation Engineering (CSAE), 2011 IEEE International Conference on*, 2011, pp. 747-752.

[10] Y. Linbin, L. Yu, M. Nourozborazjany, R. N. Kacker, and D. R. Kuhn, "An Efficient Algorithm for Constraint Handling in Combinatorial Test Generation," in *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*, 2013, pp. 242-251.

[11] G. Shiwei, D. Binglei, J. Yaruo, L. Jianghua, and M. Shilong, "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.

[12] M. Grindal, J. Offutt, and J. Mellin, "Handling constraints in the input space when using combination strategies for software testing," 2006.

[13] (2015, 25 Oct.). *User Profiles*. Available: http://www.mantisbt.org/manual/admin.user.profiles.html

[14] (2015, 25 Oct.). *What is a User Profile? - Definition from Techopedia*. Available: https://www.techopedia.com/definition/16137/user-profile

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

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)