Application of Fused Testing in Software Testing Field

Najla Althuniyan and Tanzila Saba
College of Computer and Information Sciences
Prince Sultan University Riyadh KSA

Abstract:

Testers face difficulties doing sufficient investigations within the available time constrains. In the challenging commercial marketplace, it is not possible to fully test all the different combinations of system test cases. Combination testing is a test case selection technique that combines important values and boundaries of the input parameters of a function or procedure parameters to generate different and useful test cases .This research presents the output from an explanation of six combination testing strategies. The six combination strategies are the Each Choice strategy (EC), the Base Choice strategy (BC), Pair-wise testing strategy (PW), Orthogonal Arrays (OA) , the algorithm from the Automatic Efficient Test Generator (AETG) and the All Combination strategy (AC). Also there are some techniques for combination testing: there are a number of automatic test case generation tools available but these can experience combinatorial explosions. The paper will provide an explanation of creating and using combination testing strategies.

Introduction

Software implementation errors are one of the important significant factors to information system. Development budgets are spent around 50% to 80% of on the testing stage. Exhaustive testing, testing all possible combinations of inputs and execution paths, is impossible for real-world software (Rehman and Saba 2012). Combination testing refers to tests that combine more than one variable at the same time. Sometimes these variables have different range of values. The more variables you combine, the higher the more possible test cases. The tester plans, documents or describes the combinations using combination charts, and should know how to build and fill the chart to perform the combinational testing. There are different strategies and techniques for combination: the Each Choice strategy (EC), the Base Choice strategy (BC), Pair-wise testing strategy (PW), Orthogonal Arrays (OA) , the algorithm from the Automatic Efficient Test Generator (AETG) and the All Combination strategy (AC).

Other approaches depend on meaningful relationships between variables, such as scenario tests, sequential tests, or other risk-focused tests. Usually there will be many combinations identified as possible causes, so substantial additional testing may be needed to determine more test cases. Combination testing is helpful to exam different paths during the run time to check every possible variable. In this research paper, there will be more information about combination testing techniques.

(DOI: dx.doi.org/14.9831/1444-8939.2015/3-1/MAGNT.14)

Background

Nowadays, most of things in the world are done systematically. One part of any system is software. The important feature in any software product is reliability. Reliability can be measured by how many bugs are there in any software product. Not even professional testers can proof if the program has no errors or an infinity number of errors. However, the strategy that is done on a software product can tell how the software is reliable. Testing has always been an important stage of the software development process. Combination testing refers to tests that have more than one variable. Combination testing can saves time and money by eliminating test cases. For example, if you can reduce 800,000 possible test cases to 500 or 400 test cases that is an excellent job. Combination testing reduces test cases into possible and important test cases by partitioning all variables and inputs into equivalences classes, which reduces the complexity for each test case (Saba and Rehman, 2012).

In this paper, techniques and strategies are used to design test cases based on combination testing. Some questions that my come up includes: while these techniques reduce test cases, does it help find bugs? Does it save time? Yes, it does.

The six combination testing strategies researched in this research paper are the Each Choice strategy (EC), the Base Choice strategy (BC), Pair-wise testing strategy (PW), Orthogonal Arrays (OA) , the algorithm from the Automatic Efficient Test Generator (AETG) and the All Combination strategy (AC). These six combination strategies are coverage levels for variables and inputs that range depending on each test case.

A serious problem for multi-variable testing is known as combinatorial explosion. The more variables you should combine, the higher the number of possible test cases you will have. Planning these test cases using these strategies is done with "combinations charts".

Discussion:

A job for the combination testing is the founding of an input parameters set of the tested function or procedure. Each parameter represents an interesting value for the function input parameters. A tester may use different techniques to select some values such as Equivalence Partitioning, Boundary Value Analysis or both. However, there are limitations in these techniques. This paper will provide details about the combination testing strategies. So each tester can choose the appropriate strategies without limitations.

The six combination testing strategies examined in this paper are Each-Choice (EC), Base Choice (BC), Orthogonal Arrays (OA), the strategy used by the tool Automatic Efficient Test Generator (AETG), and All-Combinations (AC). The main goal of this paper is to have more test coverages for different levels of test cases.

Each Choice (EC):

The main point of the Each Choice (EC) testing strategy asks each parameter's value to be used in one test case at least. This definition is used for 1-wise coverage also. For example, if there are three sets and each set contains three parameters, the test cases will be three only. Given this example of three partitions

(DOI: dx.doi.org/14.9831/1444-8939.2015/3-1/MAGNT.14)

with blocks [female, male], [10, 20, 30], and [blue, white], EC can be satisfied in many ways, including the three test cases (female, 10, blue), (male, 20, white), and (female, 30, blue).To generate test case using this strategy, you have to use the unused value of each next parameter. This builds a successful selection of parameters. Moreover, it gives a tester more flexibility in terms how to combine variables and test cases. However, the EC strategy is considered the weakest because broad cases of parameters together with specific values will provide different results which may not be covered with this strategy. Generally, the EC is considered ineffective in complex systems.

Base Choice (BC):

The idea behind BC is "the default testing". It identifies one test base case including the first, smallest, simplest or most likely values to be tested from the end user prospective. These values may be determined by a tester or an end user in most cases. From the first test case, other important test cases can be generated by looking to interested values (from the actual user prospective) for different parameters based on the base test case. If errors arises from a test case that means one or more parameters is used outside the scope. In this strategy each value of each parameter must be covered in at least once during generating the test cases. Given this example of three partitions with blocks [female, male], [10, 20, 30], and [blue, white], BC can create the possible test cases [(female, 10, blue), (male, 20, white), and (female, 30, blue), (female, 20, white)…etc]. Some studies suggest combining basic parameters during creating test case to get effective results. Combining one or more invalid values in a test case are not useful and gives a negative result that has unknown causes. Sometime it is very hard to design one base test case, and then a tester can rely on multiple base test cases. For this, it is very important to have all test cases be documented.

Pair-wise (PW):

Pair-wise testing is based on a fact: that most incorrect software results are caused by interactions of at most two important parameters and most of the time these parameters are the factor for a process. It is also known as "Combinational testing shortcuts". Pair-wise testing strategy is a combinatorial software testing method that takes a pair for all input parameters and creates all possible distinct combinations of values for the given parameters. The number of created test cases is typically equal O(nm), where n and m are the number of capabilities for each of the two parameters with the typical choices. Given the same example of three set with partitions [[female, male], [10, 20, 30], and [blue, white]], the Pair-wise will create sixteen test cases to include the all following combinations: [(female, 10), (B, 10) ,(10, blue), (female, 20), (male, 20) (10, white), (female, 30), (male, 30), (20, blue), (female, blue), (male, blue) (20, white), (female, white), (male, white), (30, blue), (30, white)]. Because there is no testing techniques that can discover all bugs; the PW technique is used a lot with some quality assurance techniques such as symbolic execution, code review, unit testing, and fuzz testing together to find more bugs. Testers must be choosing test factors very carefully because it will save time and efforts than running an exhaustive combination for each test case.

(DOI: dx.doi.org/14.9831/1444-8939.2015/3-1/MAGNT.14)

Orthogonal Arrays (OA):

The Orthogonal Arrays (OA) combination strategy is also known in the mathematic area with the same concept name. Basically, orthogonal arrays are two dimensional arrays of parameters value which with any two columns in the array you will have multiple values of all the pair-wise combinations of values in the array. "An orthogonal array OAλ(N; k, v, t) is an N × k array on v symbols such that every × t sub-array contains all rows of size t from v symbols exactly λ times. where (N – Number of test cases, k – Number of parameters, v – Number of values of each parameter, t – Degree of interaction, λ - 1 for software testing and is often omitted) " [Bierbrauer 001]. However, some duplicates will be happened for some parameters combination. If these duplicates have been extracted, the test cases number still close to the values for the two main parameters.

Orthogonal arrays have been used widely to design scientific experiments. Orthogonal array testing strategy provides the maximum test coverage with a reasonable number of test cases. Therefore, the orthogonal array testing strategy is very useful and more suitable to catch more errors. The OA is extremely valuable for testing huge applications and e-commercial products. However, the more independent parameters should be combined, the complex selection creating of orthogonal array. The OA is a hard solution that is used for a complex problem.

Automatic Efficient Test Generator (AETG):

The Automatic Efficient Test case Generator (AETG) uses the idea of generating test cases from combining parameters or values with the consideration of the relationship between these parameters. The two main components for the Automatic Efficient Test case Generator are relations and fields. A field is usually considered as a parameter with a certain value, and a relation is usually determined by how these fields will interact with each other. The AETG does not create real test cases, but a table of the input values needed to execute all needed test cases. A tester must determine the different possible values for each input in a relation. For thus, the AETG is not a tool, it is a methodology that be used a lot in the software testing field.

"This is the original AETG algorithm published in a technical paper by David M. Cohen [3]:

Assume that we have a system with k test parameters and that the ith parameter has li different values. Assume that we have already selected r test cases. We select the r + 1 by first generating M different candidate test cases and then choosing one that covers the most new pairs. Each candidate test case is selected by the following greedy algorithm:

1. Choose a parameter f and a value l for f such that that parameter value appears in the greatest number of uncovered pairs.

2. Let f1 = f. Then choose a random order for the remaining parameters. Then, we have an order for all k parameters f1, . . . , fk.

3. Assume that values have been selected for parameters f1, . . . , fj . For 1 _ i _ j, let the selected value for fi be called vi. Then, choose

(DOI: dx.doi.org/14.9831/1444-8939.2015/3-1/MAGNT.14)

a value vj+1 for fj+1 as follows. For each possible value v for fj , find the number of new pairs in the set of pairs {fj+1 = v and fi = vi for 1 _ i _ j}.

Then, let vj+1 be one of the values that appeared in the greatest number of new pairs. Note that, in this step, each parameter value is considered only once for inclusion in a candidate test case.

Also, that when choosing a value for parameter fj+1, the possible values are compared with only the j values already chosen for parameters f1,, fj" .[Cohen 011]

This algorithm has been modified to avoid the drawback for the AETG, because it has a limitation to upper limit for the test cases number.

All Combinations (AC):

All combinations (AC) ask to combine all values for each parameter to generate a test case. This is also the definition of the n-wise coverage.

For example, if a function has three parameters with these values: [female, male], [10, 20, 30], and [blue, white], then the AC will create the following twelve test cases: [(female, 10, blue), (male, 10, blue), (female, 10, white), (male, 10, white), (female, 20, blue), (male, 20, blue), (female, 20, white), (male, 20, white), (female, 30, blue), (male, 30, blue), (female, 30, white), (male, 30, white)]. The AC seems to be the foundation of combination testing. It can be used for small functions and procedures with limited parameters and values but not for the big one. It may be used for educational and academic purposes.

Conclusion:

Combination testing has received more attention from both academic and industries areas. Combination testing could reduce the number of tests and still remaining effective for catching software errors. This research paper presents six strategies for the creation of input parameter for test cases using combination testing strategies. Since software grows in complexity, quality issues must be met in each software requirement. Testers face a big challenge in doing their jobs because software is a group of different complicated paths. Combination testing are very useful when testing and examining paths of execution. Based on each software environment, scenario, risk and functionality, a tester can use the appropriate strategy.

Future work:

I found through my research that minimal research has been done on automating combination testing for websites, web application and electronic machines such as ATM or medical equipment. The big picture is how to apply these strategies, choose the best strategy, and manage conflicts? Since it is not correct to generalize results among all cases, how to perform exceptions between these combination testing strategies? The other issue that should be addressed in future research is: if some test cases have been edited after generation, can this editing affect the other test cases? Do testers need to recreate test cases again or do they have to use a different strategy?

(DOI: dx.doi.org/14.9831/1444-8939.2015/3-1/MAGNT.14)

References

[Bierbrauer 001]Bierbrauer, J, K Gopalakrishnan, and D.R Stinson. 1996. Orthogonal Arrays, Resilient Functions, Error-correcting Codes, and Linear Programming Bounds. SIAM Journal on Discrete Mathematics. 9, no. 3: 424-452.

[Cohen 001] Cohen, D.M, S.R Dalal, M.L Fredman, and G.C Patton. 1997. The AETG System: An Approach to Testing Based on Combinatorial Design. Software Engineering, IEEE Transactions on. 437-444.

Agarwal, B, S Tayal, and M Gupta. 2010. Software Engineering & Testing: An Introduction. Sudbury, Mass.: Jones and Bartlett.

Ammann, Paul, and Jeff Offutt. 2008. Introduction to Software Testing. New York: Cambridge University Press.

Andersson, Jonny. Automatic test vector generation and coverage analysis in model-based software development a Master's thesis in Vehicular Systems, 2005. Reg nr: LiTH-ISY-EX-3765-2005. 14th December 2005.

Andrews, Mike, and James A Whittaker. 2006. How to Break Web Software: Functional and Security Testing of Web Applications and Web Services. Upper Saddle River, NJ: Addison-Wesley.

Copeland, Lee. 2004. A Practitioner's Guide to Software Test Design. Boston, Mass.: Artech House.

Desikan, Srinivasan, and Gopalaswany Ramesh. 2006. Software Testing: Principles and Practice. Bangalore, India: Dorling Kindersley (India).

Grabowski, Jens, ed. , Brian Nielsen2005. Formal Approaches to Software Testing: 4th International Workshop, FATES 2004, Linz, Austria, September 21, 2004 : Revised Selected Papers. Berlin: Springer.

Hoffman, Daniel, Brett Stevens, Chien Chang, Kevin Yoo, and Gary Bazdell. 2010. Bad Pairs in Software Testing. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 6303, no. PAGE: 39-55.

Kaner, Cem, James Bach, and Bret Pettichord. 2002. Lessons Learned in Software Testing: A Context-driven Approach. New York: Wiley.

Lamancha, Beatriz Pérez, and Macario Polo Usaola. 2010. Testing Product Generation in Software Product Lines Using Pairwise for Features Coverage. ICTSS'10: Proceedings of the 22nd IFIP WG 6.1 International Conference on Testing Software and Systems.

Naik, Kshirasagar, and Priyadarshi Tripathy. 2008. Software Testing and Quality Assurance: Theory and Practice. Hoboken, N.J.: John Wiley & Sons.

Page, Alan, Ken Johnston, and Bj Rollison. 2009. How We Test Software at Microsoft. Redmond, Wash.: Microsoft.

Petrenko, Alexandre, ed. , Andreas Ulrich2004. Formal Approaches to Software Testing: Third International Workshop on Formal Approaches to Testing of Software : FATES 2003 : Montreal, Quebec, Canada, October 6th,

(DOI: dx.doi.org/14.9831/1444-8939.2015/3-1/MAGNT.14)

2003 : Revised Papers. Berlin: Springer-Verlag.

Rehman, A and Saba, T. (2012). Evaluation of Artificial Intelligent Techniques to Secure Information in Enterprises, Artificial Intelligence Review, DOI. 10.1007/s10462-012-9372-9.

Saba, T. Rehman, A. (2012). Machine Learning and Script Recognition, Lambert Academic publisher, pp:23-31.

Spillner, Andreas, Tilo Linz, and Hans Schaefer. 2011. Software Testing Foundations: A Study Guide for the Certified Tester Exam : Foundation Level, ISTQB Compliant. Santa Barbara, CA: Rocky Nook.

Srivastava, Praveen Ranjan, and Km Baby. 2010. Automated Software Testing Using Metahurestic Technique Based on an Ant Colony Optimization. Proceedings - 2010 International Symposium on Electronic System Design, ISED 2010. 235-240.

Tiuryn, Jerzy, ed. 2000. Foundations of Software Science and Computation Structures: Third International Conference, FOSSACS 2000, Held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2000, Berlin, Germany, March 25-April 2, 2000 : Proceedings. Berlin: Springer.

Whittaker, James A. 2003. How to Break Software: A Practical Guide to Testing. Boston: Addison Wesley.

Xu, Ruoan, Yoshimitsu Nagai, and Syohei Ishizu. 2011. Software Testing Method Considering the Importance of Factor Combinations in Pair-Wise Testing. 1865-0937. 173: 1865-0937.

Zamli, Kamal, Nor Ashidi Mat Isa, Mohammad F.J Klaib, Rusli Abdullah, and Mohammed Younis. 2011. Design and Implementation of a T-way Test Data Generation Strategy with Automated Execution Tool Support. Information Sciences. 181, no. 9: 1741-1758.

After the In-class Presentation:

A question came after the in class presentation: Is the combination testing considered as white box testing or a black box testing?

The answer is: it is considered as a white box testing on the unit, function or procedure level.

(DOI: dx.doi.org/14.9831/1444-8939.2015/3-1/MAGNT.14)