Pairwise Testing

Anuruddha Jayasekara Pathiranage

02/19/2015

Pairwise Testing

Pairwise testing has become an indispensable tool in a software tester's toolbox. The technique has been known for almost 20 years (Tai & Lei, 2002), but only last five years there has a tremendous increase in its popularity. Pairwise testing is a wildly popular approach for combinatorial testing problems. Pairwise testing is an economical alternative tool to testing all possible combinations of a set of variables. In pairwise testing a set of test cases is generated that covers all combinations of the selected test data values for each pair of variables. Pairwise testing is also referred to as all pairs testing and 2-way testing. Also we can do all triples (3-way) or all quadruples (4- way) testing, of course, but the size of the higher order test sets grows very rapidly. Generally, pairwise testing begins by selecting values for the system's input variables. These individual values are often selected using domain partitioning. The values are then permuted to achieve coverage of all the pairings(Bach & Schroeder, 2004).

Example of Pairwise Testing

An mobile application with simple drop down box with 5 elements (Let's say 0,1,2,3,4) along with a checkbox, radio button, in put Text field and Save Button. The Constraint for the Text field is, it can accept values only between 1 and 50. Below are the values that each one of the GUI objects can take :

Drop down Box - 0,1,2,3,4

Check Box - Checked or Unchecked

Radio Button - ON or OFF

In put Text field - Any Value between 1 and 50

Exhaustive combination of the product,

Drop Down Box = 5

Check Box = 2

Radio Button = 2

Text Field = 50

Total Number of Test Cases using Cartesian Method : 5*2*2*50 = 1000

Total Number of Test Cases including Negative Cases will be > 1000

Now, the idea is to bring down the number of test cases. We will first try to find out the number of cases using the conventional software testing technique. We can consider the drop down box values as 0 and others as 0 is neither positive nor negative. Radio button and check box values cannot be reduced, so each one of them will have 2 combinations (ON or OFF). The Text field values can be reduced into three inputs (Valid Integer, Invalid Integer, Alpha-Special Character).

Now, we will calculate the number of cases using software testing technique, 2*2*2*3 = 24 (including negative cases).

Now, we can still reduce the combination further into All-pairs technique.

Step 1: Order the values such that one with most number of values is the first and the least is placed as the last variable.

Step 2: Now start filling the table column by column. Drop down box can take 2 values.

Step 3: The Next column under discussion would be check box. Again Check box can take 2 values.

Step 4: Now we need to ensure that we cover all combinations between drop down box and Check box.

Step 5: Now we will use the same strategy for checking the Radio Button. It can take 2 values.

Step 6: Verify if all the pair values are covered as shown in the table below.

In put Text field	Drop Down Box	Check Box	Radio Button
Valid Int	0	check	ON
Valid Int	others	uncheck	OFF
Invalid Int	0	check	ON
Invalid Int	others	uncheck	OFF
AlphaSpecialCharacter	0	check	ON
AlphaSpecialCharacter	others	uncheck	OFF

Result of Pair-Wise Testing:

Exhaustive Combination results in > 1000 Test Cases.

Conventional Software Testing technique results in 24 Test Cases.

Pair Wise Software Testing technique results in just 6 Test Cases.

When Pairwise Testing should be used

Pairwise testing can be used in a testing approach taken for testing the software using combinatorial method. This method can be used to test all the possible discrete combinations of the parameters involved.

Types of software testing problems that pairwise testing addresses

The pairwise approach is popular due to the generation of small test sets that are relatively easy to manage and execute. But it is not just the significant reduction in test set size that makes pairwise testing appealing. It is also focus on testing on a richer source of more important bugs. So this nicely address following testing problems;

1) Pairwise testing protects against pairwise bugs

2) Reducing the number tests to perform

3) Pairwise bugs represent the majority of combinatoric bugs

4) The bugs are a lot more likely to happen than ones that only happen with more variables.

5) The availability of tools means you no longer need to create these tests by hand (Bach & Schroeder, 2004).

Key assumptions of pairwise testing

Pairwise testing requires that for each pair of input parameters of a system, every combination of valid values of these two parameters be covered by at least one test case(Lei & Tai, 1998).

Limitations of pairwise testing

1) Pairwise testing can only protect against pairwise interactions of the input values that we happen to select for testing. The selected values are normally an extremely small subset of the actual number of possible values.
2) And if we select values individually using domain analysis, we might miss productive test cases that contain special-case data combinations and error-prone combinations. The weaknesses of domain partitioning are carried over into pairwise testing and magnified.
3) Pairwise testing is argued that the probability of more than two particular values coinciding cannot be greater than and is probably a lot less than the probability of only two coinciding values(Bach & Schroeder, 2004).

References

Bach, J., & Schroeder, P. J. (2004). Pairwise testing: A best practice that isn't. In *Proceedings of 22nd Pacific Northwest Software Quality Conference* (pp. 180–196). Citeseer. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.159.5244&rep=rep1&type=pdf

Lei, Y., & Tai, K.-C. (1998). In-parameter-order: A test generation strategy for pairwise testing. In *High-Assurance Systems Engineering Symposium, 1998. Proceedings. Third IEEE International* (pp. 254–261). IEEE. Retrieved from http://ieeexplore.ieee.org/abstract/document/731623/

Tai, K.-C., & Lei, Y. (2002). A test generation strategy for pairwise testing. *IEEE Transactions on Software Engineering*, *28*(1), 109–111.