

# All-pairs Testing and TestTrack TCM

by Keith Vanden Eynden

The common assumption about software testing is that more is better, and that testing all the possible states and variable combinations guarantees you will find all the bugs. In the real world, there is not enough time or enough testers to test every combination of every variable. Not all bugs will be found, making quality assurance a risk management discipline. How can you validate that your product is ready to ship within reasonable time and cost parameters? In other words, how can you manage the risk of not testing everything? One solution is to use testing methodologies, supported by proper tools, which help you quantifiably manage this risk.

Practically speaking, the role of quality assurance is to reduce the risk of these bugs ending up in the final product. Software complexity puts a huge burden on QA teams, which are typically much smaller than the development teams writing the software. It is also very easy for one developer to write a small amount of code that requires a significant amount of testing to ensure it functions properly in all situations.

For example, you have to test a dialog box with three drop-down lists to see if any of the combinations cause the program component to fail. The first list has five options, the second has eight options, and the third has three.

Option 1	Option 2	Option 3
A	A	A
B	B	B
C	C	C
D	D	
E	E	
	F	
	G	
	H	

To determine all the possible combinations, you can create a matrix like the following.

Test Run	Option 2	Option 1	Option 3
1	A	A	A
2	A	A	B
3	A	A	C
4	A	B	A
5	A	B	B
6	A	B	C
7	A	C	A
8	A	C	B
9	A	C	C

As you continue adding combinations, you discover that 120 test cases are required to cover all the possible combinations. You can also determine the number of combinations by multiplying the number of values available in each option ( $5 \times 8 \times 3 = 120$ ). If each test takes two minutes to perform, you are faced with four hours of testing on a simple dialog box. What if you need to test 100 dialog boxes? What if some dialog boxes contain fifteen options instead of three?

Now take the concept of complete coverage a step further and consider environmental variables such as operating system, database, and hardware components. How do you ensure that you find a bug that occurs only when the application is running on Windows XP and is using MySQL without testing all the possible OS and database combinations?

These examples demonstrate how quickly complete coverage becomes unmanageable. Luckily, you can find most bugs without testing all the combinations. The simplest bugs are single-mode faults, which occur when one option causes a problem regardless of the other settings. For example, a printout is always smeared when you choose the duplex option in the print dialog box regardless of the printer or the other selected options. Another type of bug is one that occurs when two options are combined—the printout is only smeared when duplex is selected and the printer is a model 394. These are called double-mode faults. Finally, multi-mode faults, which occur when three or more settings produce the bug, are the types of problems that make complete coverage seem necessary.

However, complete coverage is usually not necessary. A study by Telcordia Technologies found that "...most field faults were caused by either incorrect single values or by an interaction of pairs of values" (Cohen, et al. 1996). Another study of the software in medical devices showed that only three of the 109 failures resulted from the combination of more than two conditions (Wallace, 2000). If you have limited time and resources, you want to find the most common bugs and those that present the highest risk. Suppose the printer error only occurs when the operating system is Windows, the print option is set to duplex, the print quality is draft, and the Collate option is not selected. Is it worth your time to find that bug? Does the bug present a big enough risk to the user or application that it will even require a software fix?

Except in the rare cases where life and death are at stake, you can achieve a statistically acceptable level of quality by testing less than 100 percent of the combinations. One approach to doing this is called pair-wise or all-pairs testing.

### Implementing all-pairs testing

If testing all the possible combinations of values is not necessary, then the question becomes how to construct the all-pairs tests. There are several applications that allow you to enter variables and generate the tests. However, it is helpful to try constructing the test cases manually once or twice so you understand exactly how all-pairs testing works.

### Identifying the variables

Before you can implement all-pairs testing, you need to identify the variables. For example, single sign-on support was recently added to your company's sales management application, and the login screen was redesigned for this feature. The application runs on a variety of operating systems, supports several databases, and includes a cross-platform graphical user interface (GUI) and a Web-based client. The following table summarizes the variables that affect the tests.

Operating System	Database	Client Type	Browser
Windows	Access	GUI	Internet Explorer
Linux	SQL Server	Web	Firefox
Solaris	Oracle		
	MySQL		

(Note: For the sake of simplicity, the following examples assume that all the variable combinations are valid.)

How do you effectively test single sign-on support based on these different variables? A simple calculation shows that there are 48 possible combinations ( $3 \times 4 \times 2 \times 2$ ). All-pairs testing can significantly reduce that number.

### Creating the first pair of values

After identifying the variables, use a spreadsheet to combine the values from a pair of variables. Before starting, arrange the variables by the number of values they contain from greatest to least. In our example, the variables would be placed in the following order: Database, Operating System, Client Type, and Browser. Label the first column in the spreadsheet with the name of the variable with the most values.

Database

Label the second column with the name of the variable with the second-highest number of values.

Database	Operating System

Pair the first value in the first variable column with the first value in the second variable column.

Database	Operating System
Access	Windows

Then match the first value from the first variable column with the second value in the second variable column.

Database	Operating System
Access	Windows
Access	Linux

Continue the process until the first value of the first variable is paired with each of the second variable values.

Database	Operating System
Access	Windows
Access	Linux
Access	Solaris

Skip a row in the spreadsheet to improve readability and to allow for expansion. Then match the first variable's second value with all the values from the second variable.

Database	Operating System
Access	Windows
Access	Linux
Access	Solaris
SQL Server	Windows
SQL Server	Linux
SQL Server	Solaris

Repeat the steps until all values in the first two variables are paired up.

Database	Operating System
Access	Windows
Access	Linux
Access	Solaris
SQL Server	Windows
SQL Server	Linux
SQL Server	Solaris
Oracle	Windows
Oracle	Linux
Oracle	Solaris
MySQL	Windows
MySQL	Linux
MySQL	Solaris

### Matching the other pairs of values

To add a third variable, start by entering the values in order in a third column, repeating as necessary.

Database	Operating System	Client Type
Access	Windows	GUI
Access	Linux	Web
Access	Solaris	GUI
SQL Server	Windows	Web
SQL Server	Linux	GUI
SQL Server	Solaris	Web
Oracle	Windows	GUI
Oracle	Linux	Web
Oracle	Solaris	GUI
MySQL	Windows	Web
MySQL	Linux	GUI
MySQL	Solaris	Web

Next, compare the combinations and make sure you have all the possible pairs for the second and third variables. In our example, there is a pair for the Windows Operating System and each Client Type.

Database	Operating System	Client Type
Access	Windows	GUI
Access	Linux	Web
Access	Solaris	GUI
SQL Server	Windows	Web
SQL Server	Linux	GUI
SQL Server	Solaris	Web
Oracle	Windows	GUI
Oracle	Linux	Web
Oracle	Solaris	GUI
MySQL	Windows	Web
MySQL	Linux	GUI
MySQL	Solaris	Web

There is also a pair for the Linux Operating System and each Client Type.

Database	Operating System	Client Type
Access	Windows	GUI
Access	Linux	Web
Access	Solaris	GUI
SQL Server	Windows	Web
SQL Server	Linux	GUI
SQL Server	Solaris	Web
Oracle	Windows	GUI
Oracle	Linux	Web
Oracle	Solaris	GUI
MySQL	Windows	Web
MySQL	Linux	GUI
MySQL	Solaris	Web

Finally, there is a pair for the Solaris Operating System and each Client Type.

Database	Operating System	Client Type
Access	Windows	GUI
Access	Linux	Web
Access	Solaris	GUI
SQL Server	Windows	Web
SQL Server	Linux	GUI
SQL Server	Solaris	Web
Oracle	Windows	GUI
Oracle	Linux	Web
Oracle	Solaris	GUI
MySQL	Windows	Web
MySQL	Linux	GUI
MySQL	Solaris	Web

If a pair is missing, rearrange the values to create the necessary combinations. Notice that it only takes six test runs to cover all the Operating System and Client Type combinations.

Database	Operating System	Client Type
Access	Windows	GUI
Access	Linux	Web
Access	Solaris	GUI
SQL Server	Windows	Web
SQL Server	Linux	GUI
SQL Server	Solaris	Web
Oracle	Windows	GUI
Oracle	Linux	Web
Oracle	Solaris	GUI
MySQL	Windows	Web
MySQL	Linux	GUI
MySQL	Solaris	Web

Next, compare the combinations between the first and third variables to make sure you have all the possible pairs.

Database	Operating System	Client Type
Access	Windows	GUI
Access	Linux	Web
Access	Solaris	GUI
SQL Server	Windows	Web
SQL Server	Linux	GUI
SQL Server	Solaris	Web
Oracle	Windows	GUI
Oracle	Linux	Web
Oracle	Solaris	GUI
MySQL	Windows	Web
MySQL	Linux	GUI
MySQL	Solaris	Web

Notice that 10 test runs cover all the Database/Client Type and Operating System/Client Type pairs.

Database	Operating System	Client Type
Access	Windows	GUI
Access	Linux	Web
Access	Solaris	GUI
SQL Server	Windows	Web
SQL Server	Linux	GUI
SQL Server	Solaris	Web
Oracle	Windows	GUI
Oracle	Linux	Web
Oracle	Solaris	GUI
MySQL	Windows	Web
MySQL	Linux	GUI
MySQL	Solaris	Web

If there are more variables, continue the same procedure of creating pairs with the values from the last variable and the values for the other variables.

In our example, the Browser values are only valid when the Client Type is Web. The following table shows the Browser values added to the Web test runs.

Database	Operating System	Client Type	Browser
Access	Windows	GUI	
Access	Linux	Web	IE
Access	Solaris	GUI	
SQL Server	Windows	Web	FireFox
SQL Server	Linux	GUI	
SQL Server	Solaris	Web	IE
Oracle	Windows	GUI	
Oracle	Linux	Web	Firefox
Oracle	Solaris	GUI	
MySQL	Windows	Web	IE
MySQL	Linux	GUI	
MySQL	Solaris	Web	Firefox

Verify that all the Web pairs are present for the other variable values. There is a pair for each Browser value and the Web Client Type.

Database	Operating System	Client Type	Browser
Access	Windows	GUI	
Access	Linux	Web	IE
Access	Solaris	GUI	
SQL Server	Windows	Web	FireFox
SQL Server	Linux	GUI	
SQL Server	Solaris	Web	IE
Oracle	Windows	GUI	
Oracle	Linux	Web	Firefox
Oracle	Solaris	GUI	
MySQL	Windows	Web	IE
MySQL	Linux	GUI	
MySQL	Solaris	Web	Firefox

There is a pair for each Operating System and Browser value.

Database	Operating System	Client Type	Browser
Access	Windows	GUI	
Access	Linux	Web	IE
Access	Solaris	GUI	
SQL Server	Windows	Web	FireFox
SQL Server	Linux	GUI	
SQL Server	Solaris	Web	IE
Oracle	Windows	GUI	
Oracle	Linux	Web	Firefox
Oracle	Solaris	GUI	
MySQL	Windows	Web	IE
MySQL	Linux	GUI	
MySQL	Solaris	Web	Firefox

Finally, check the pairs for the Browser values and the Database values. Notice that the Oracle/Internet Explorer and Access/Firefox pairs are missing.

Database	Operating System	Client Type	Browser
Access	Windows	GUI	
Access	Linux	Web	IE
Access	Solaris	GUI	
SQL Server	Windows	Web	FireFox
SQL Server	Linux	GUI	
SQL Server	Solaris	Web	IE
Oracle	Windows	GUI	
Oracle	Linux	Web	Firefox
Oracle	Solaris	GUI	
MySQL	Windows	Web	IE
MySQL	Linux	GUI	
MySQL	Solaris	Web	Firefox

You could simply add two new test runs to cover the missing Database and Browser combinations, however, before you do that, check for duplicate pairs to see if you can incorporate the missing pairs into the existing runs. Notice that the Solaris/GUI and the Windows/GUI pairs are duplicated in the Access and Oracle groups.

Database	Operating System	Client Type	Browser
Access	Windows	GUI	
Access	Linux	Web	IE
Access	Solaris	GUI	
SQL Server	Windows	Web	FireFox
SQL Server	Linux	GUI	
SQL Server	Solaris	Web	IE
Oracle	Windows	GUI	
Oracle	Linux	Web	Firefox
Oracle	Solaris	GUI	
MySQL	Windows	Web	IE
MySQL	Linux	GUI	
MySQL	Solaris	Web	Firefox

You can change the Solaris/GUI pair in the Oracle group and add the Internet Explorer/Oracle pair to that run without affecting the other pairs. You can also change the Windows/GUI pair in the Access group and add the Firefox/Access pair to that run.

Database	Operating System	Client Type	Browser
Access	Windows	Web	Firefox
Access	Linux	Web	IE
Access	Solaris	GUI	
SQL Server	Windows	Web	FireFox
SQL Server	Linux	GUI	
SQL Server	Solaris	Web	IE
Oracle	Windows	GUI	
Oracle	Linux	Web	Firefox
Oracle	Solaris	Web	IE
MySQL	Windows	Web	IE
MySQL	Linux	GUI	
MySQL	Solaris	Web	Firefox

After pairing up the variables, add a column and number the test runs for easy reference. The following matrix summarizes the final test runs.

Test Run	Database	OS	Client Type	Browser
1	Access	Windows	Web	Firefox
2	Access	Linux	Web	IE
3	Access	Solaris	GUI	
4	SQL Server	Windows	Web	FireFox
5	SQL Server	Linux	GUI	
6	SQL Server	Solaris	Web	IE
7	Oracle	Windows	GUI	
8	Oracle	Linux	Web	Firefox
9	Oracle	Solaris	Web	IE
10	MySQL	Windows	Web	IE
11	MySQL	Linux	GUI	
12	MySQL	Solaris	Web	Firefox

Instead of 48 test runs, only 12 are required to provide the proper test coverage. If particular variable combinations have resulted in a higher number of defects in the past, you might want to include additional runs to cover those combinations.

## All-pairs testing and TestTrack TCM

TestTrack TCM, Seapine's test case management solution, manages all facets of the software testing process including test case creation, scheduling, execution, measurement, and reporting. TestTrack TCM's test variants allow you to define the variables that apply to the applications you are testing and also allow you to easily implement all-pairs testing. The sample project installed with TestTrack TCM contains the following variants and values.

Operating System	Database	Client Type
Windows	Native	Native Client
Mac OS	SQL Server	Web Client
Linux	Oracle	SOAP Client
Solaris	MySQL	Admin Client

## Customizing variants

After constructing all-pairs test runs, it is easy to implement test runs in TestTrack TCM using variants. The following example demonstrates how to set up the single sign-on tests.

Begin by customizing the Database and Client Type variant values and adding a new Browser variant.

Database	Operating System	Client Type	Browser
Access	Windows	GUI	IE
SQL Server	Linux	Web	Firefox
Oracle	Solaris		
MySQL			

With these variants in place, you can create one test case for the single sign-on feature and then generate test runs to cover different all-pairs combinations.

## Creating the test case

In TestTrack TCM, you can create a single test case and use different variant values to generate test runs and provide the desired test coverage. For example, you create one test case for the single sign-on feature that contains all the information about how to set up and run the single sign-on test.

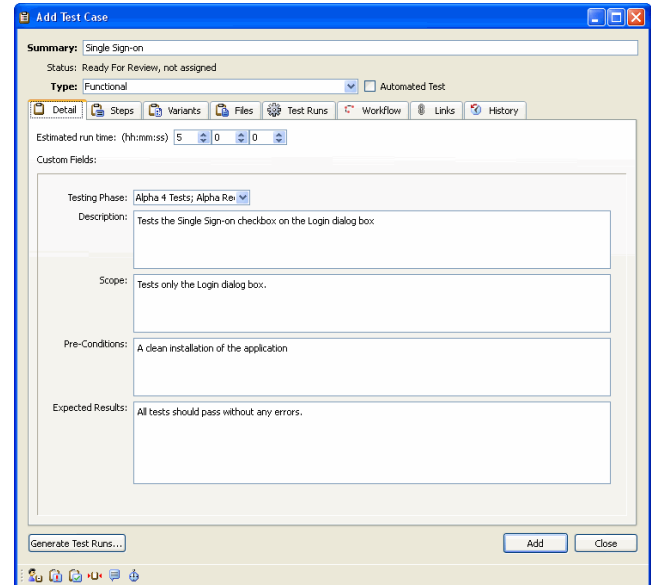


Figure 1: Completed Add Test Case window

Attach the all-pairs matrix to the test case to specify which variant value combinations to use when generating test runs.

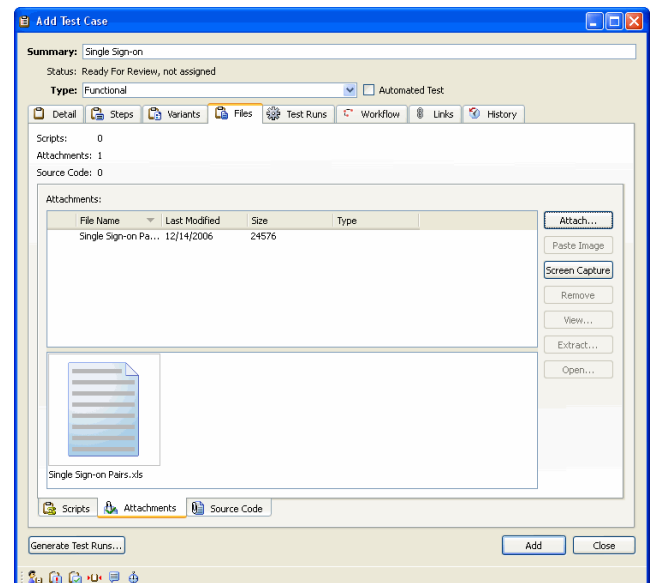


Figure 2: All-pairs matrix attached to the test case

## Generating the test runs

After the test case is reviewed and approved, you can generate the 12 test runs we identified earlier. Generate the test runs individually using the all-pairs matrix to complete the Generate Test Runs dialog box.

Test Run	Database	OS	Client Type	Browser
1	Access	Windows	Web	Firefox
2	Access	Linux	Web	IE
3	Access	Solaris	GUI	
4	SQL Server	Windows	Web	Firefox
5	SQL Server	Linux	GUI	
6	SQL Server	Solaris	Web	IE
7	Oracle	Windows	GUI	
8	Oracle	Linux	Web	Firefox
9	Oracle	Solaris	Web	IE
10	MySQL	Windows	Web	IE
11	MySQL	Linux	GUI	
12	MySQL	Solaris	Web	Firefox

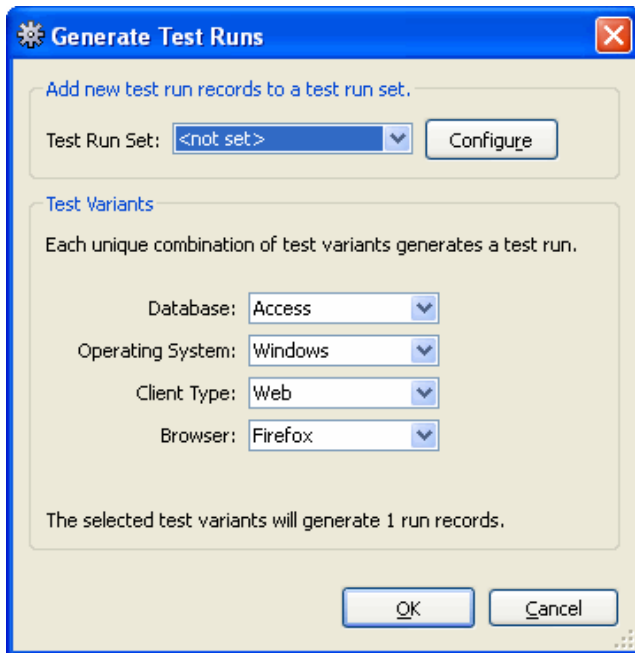


Figure 3: The Generate Test Runs dialog box for the first test run

After generating the first test run, repeat the process to generate all of the required combinations.

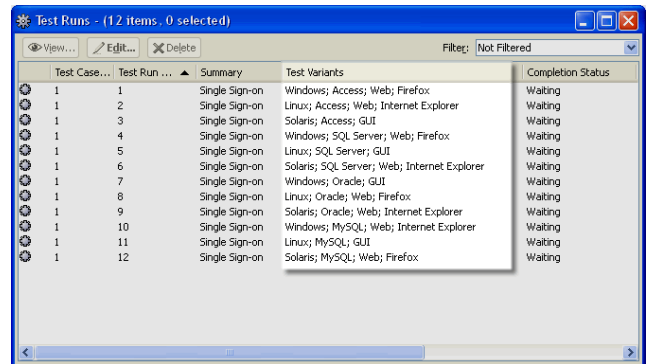


Figure 4: Test runs generated for all the pairs of variant values

All-pairs testing enables you to achieve suitable test coverage without testing every possible combination. All-pairs testing can also reduce the number of test runs by 75 percent or more. With that time savings, you will be able to spend more time developing complex tests and less time performing mundane tests on the combination of variables. By implementing all-pairs testing along with TestTrack TCM, you also improve your ability to manage your testing effort. You will spend more time focusing on actual testing and less time worrying about whether you are testing the right things.

## References

Berger, Bernie. "Efficient Testing with All-Pairs." International Conference on Software Testing 2003.

Cohen, David M., Siddhartha R. Dalal, Jesse Parelius, and Gardner C. Patton. "The Combinatorial Design Approach to Automatic Test Generation." *IEEE Software* September 1996.

Dustin, Elfriede. "Orthogonally Speaking: A Method for deriving a suitable set of test cases." *STQE* September/October 2001.

Wallace, Dolores R. and D. Richard Kuhn. "Converting System Failure Histories into Future Win Situations." Information Technology Laboratory, National Institute of Standards and Technology January 2000.

## About the Author

Keith Vanden Eynden is a senior technical writer at Seapine Software, Inc. with 15 years experience in performance improvement, training, and documentation.



www.seapine.com

5412 Courseview Dr., Suite 200  
Mason, OH 45040

TEL 513-754-1655  
FAX 513-754-1660

©2007 Seapine Software, Inc. TestTrack Pro, TestTrack, Surround SCM, QA Wizard, SoloBug, SoloSubmit, Seapine CM, Seapine SQA, and the Seapine logo are trademarks of Seapine Software, Inc. All other company products and company names are either trademarks or registered trademarks of their respective companies. All rights reserved worldwide. Information presented here is accurate as of the time of printing, but is subject to change or revision without notice. 7085.4 Whitepaper All-pairs Testing.indd 7/07