ISSN 2347-4289

Software Product Testing Using Orthogonal Array (OA) Testing Technique

Sreenivasa Pisupati

Vice President – QA Reliance Jio, Navi Mumbai, India
Email: s_pisupati@hotmail.com

Abstract: This document describes the Orthogonal Array (OA) technique used in software testing. Orthogonal Array help in improving the Software product testing.Orthogonal Array employs design of experiments (DOE), which is one of the most important statistical tools of Total Quality Management (TQM) for designing high quality systems at reduced cost. Some of the benefits of Orthogonal Array techniques are the productivity improvement, quality including high code coverage, reduction in the test execution cycle time and enhanced customer satisfaction. The actual essence of testing lies in designing the minimal set of test cases which can uncover the maximum number of bugs in the system and at the same time gives a comfortable feeling about the quality of the system. Orthogonal Array Testing Strategy is a kind of "Dream Come True" for the test designers to design their test cases.

Keywords: OA, DOE, Parameters, Levels,TQM

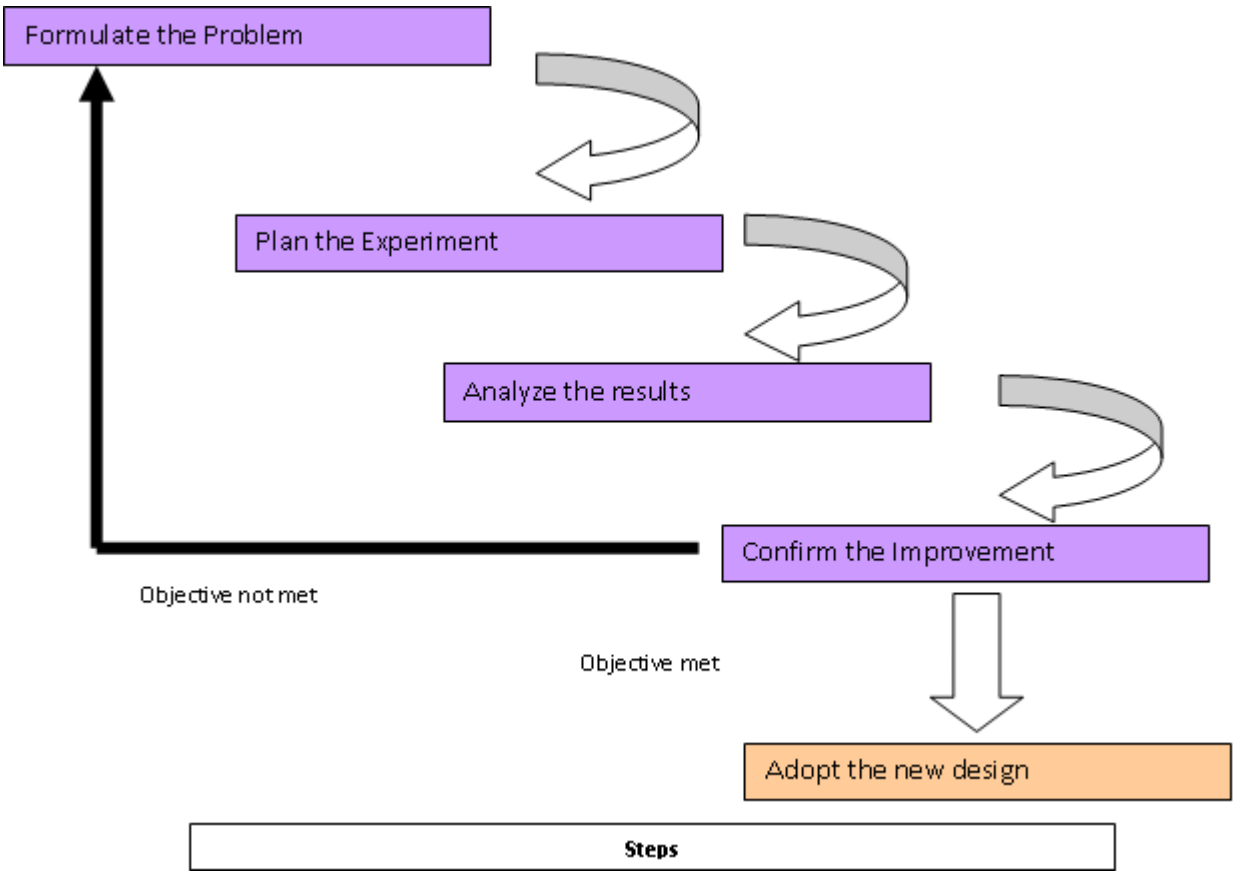
Introduction

Orthogonal Array Test Strategy is a systematic and statistical way of testing pair-wise interactions. In simple terms, Orthogonal Arrays are special set of Latin squares, constructed by Taguchi to design the test sets. It provides representative (uniformly distributed) coverage of all variable pair combinations. This technique has evolved based on the assumption that "Interactions and Integrations are the major sources of defects". Most of the defects in the software are not a result of complex interactions; they arise from simple pair-wise interactions. If the defects arose due to the simple pair-wise interactions have been arrested, then the confidence in the system will increase. It is practically not possible to test all combinations of all the variables. With so many possible combinations, it is very easy to miss one. OATS provides a test set that guarantees, testing the pair-wise combinations of all the variables. It creates an efficient and concise test set with very few test cases rather testing all the combinations of all the variables. It also exercises some of the complex combinations of all the variables. The more the product evolves the more complicated the test suite becomes. This makes the test result analysis, DR reporting and DR analysis inefficient. The result is pouring more and more costly resources in to the test activities. This document describes how this pain area can be addressed using the famous Six Sigma methodologies which provides tools like Orthogonal Arrays (OA). This whitepaper introduces Orthogonal Arrays, explains the process of using OA concepts to improve the testing process.Orthogonal Array is a method of choosing a set of tests from a universe of tests, to make the testing efficient and effective. It is based on creating Parameters (control Variables) and Levels (values) for the parameters, which are the inputs to testable functions. There are different algorithms to choose the parameters and levels from the Universe of available values. One such method was proposed by Dr Taguchi. This document deals with the OA method what Taguchi has suggested.Taguchi methods provide an efficient and systematic way to optimize designs for performance, quality, and cost.

Approach

There are five major steps in the designing process. The steps take the test team from formulating the test problem to creating a good test design and refining the test design. The steps are as follows:

Formulate the problem
Plan the experiment
Analyze the results
Confirm the experiment
Adopting the new design



Formulating the problem

It is very much required for the design engineer to think the entire system in terms of parameters that decide the outcome. Ideally, the problem definition should include two important things

The control variables

The values each variable can take.

In this, the former is called as “PARAMETER” and the later is called as “LEVELS”. One more important factor in this approach is that all the methodology is applicable only if the parameters identified are independent.

Planning the Experiment

While planning the experiment, which is essentially the design of test case, Taguchi analysis uses a concept called Matrix experiment using Orthogonal Arrays. It is an efficient way to study the effect of several factors simultaneously. Orthogonal arrays offer many benefits. First, the conclusions arrived at from such experiments are valid over the entire experimental region spanned by the control factors and their settings. Second, there is a large saving in the experimental effort. Third, the data analysis is very easy.

Matrix Experiment:

A matrix experiment consists of a set of experiments where we change the settings of the various product or process parameters we want to study from one experiment to another. After conducting a matrix experiment, the data from all experiments in the set taken together are analyzed to determine the effects of the various parameters. Constructing matrix experiments using special matrices, called “orthogonal arrays” allows the effects of several parameters to be determined efficiently and is an important technique in Robust Design. To actually construct an orthogonal array, control parameters or design variables must be assigned to the columns of an array, and the integers in the array columns are translated into the actual settings of the assigned parameters. The unassigned columns are deleted from the array.For instance, in a hypothetical design study, to evaluate effects of three design variables (PARAMATERS) on a product or process response, a designer selects two levels or settings (LEVELS) for each design parameter: Then as per Taguchi Approach

Parameter : 3
Level : 2

A suitable matrix experiment in this case uses the L4 orthogonal array as follows:

Experiment Number	Column 1	2	3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

As depicted above, the L4 experiment consists of four rows and three columns where each row corresponds to a particular experiment and each column identifies settings of a design parameter. In the first run, for example, the three design variables are set at their low level (level = 1). In the second run, the first parameter is set at level 1 and the remaining two variables are set to high (level 2), and so on... The following list contains some of the widely used orthogonal arrays, which use two and three level design parameters:

L4: *Accommodates three two-level design variables*

Experiment Number	Column 1	2	3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

i.e. 4 experiments have to be conducted for three factors two levels. Without OA we need to conduct $2^3 = 8$ experiments.

L8: *Accommodates seven two-level design variables*

Experiment Number	Column 1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	1	1	2	2	2	2
3	1	2	2	1	1	2	2
4	1	2	2	2	2	1	1
5	2	1	2	1	2	1	2
6	2	1	2	2	1	2	1
7	2	2	1	1	2	2	1
8	2	2	1	2	1	1	2

i.e. 8 experiments have to be conducted for 7 factors with 2 levels. Without OA we would have to conduct $2^7 = 128$ experiments.

L9: *Accommodates four three-level design variables*

Experiment Number	Column 1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

i.e. 9 experiments have to be conducted for 4 factors with 3 levels. Without OA we would have to conduct $3^4 = 81$ experiments

How to use this technique?

The following steps describe how to use the Orthogonal Array for arriving at the number of test cases.

1. Do a thorough analysis of the requirements document and understand the usage of the software.
2. Divide the functionalities of the software into multiple categories.
3. For each category, identify the variables on which the functionality is dependent on.
4. Identify the number of possible values for each of the variables.
5. Prepare the factor-level table. A table with factors as rows and level numbers as columns and the values of the factors in each of the cells.
6. Select the suitable Orthogonal Array from the list of available Orthogonal Arrays.
7. Prepare the test cases for each of the combinations in the selected array.

Verification Suite Development using OA

A good analysis is required before deciding on the usage of OA for Test Suite development with respect to identifying the parameters and levels. If the feature or the product, for which the test case being developed, do not have a possibilities of defining the Parameters and the respective Levels in a better way, it is suggested to look for alternatives than using OA. It will be very effective and time saving if the test development team is closely interacts with the feature or product development team from the RS phase with respect to defining the PARAMETERs and LEVELs. In case of test case re writing, the Parameters and the Levels could be arrived at by understating the feature/product itself. Help of the feature/product developer may be obtained as applicable. Defining the Parameters and the respective Levels is the crucial activity in the entire process. Once the Parameters and Levels are defined, the next step is to come out with the optimized combinations of test cases. Help of the tool called Minitab can be obtained for this purpose. Minitab is a statistics tool from the company Minitab Inc. For Minitab

the input is the Parameters and levels and the out put is the optimized test case combination. Generally the code coverage is >90% by this optimized test combination, if the parameters and levels selection is effective. At times it may not be practically possible to define parameters and levels for all the functions defined in the product or feature and also there would be requirements to have higher code coverage for products like mission critical applications. In such cases manual test cases could be developed to support the optimized test combination. This is advised since the effort what is required to identify the parameters in these cases would be very high when compare to developing a test case in the conventional way. Thus this method can give a very high code coverage which would ensure that the testing of all functionality in a minimum time and cost. In general the phase wise effort breakup in an OA test case development cycle could be as given in the table below. As discussed earlier it is assumed that the test developer will closely work with the feature/product developer from the RS defining stage to have the right set of Parameters and the Levels which will make the test case more effective. In case of test re writing, there are chances that the study phase would be slightly higher since the test developers would have to understand the feature/product on their own.

Phase	Effort
Studying the feature and identifying the parameters	15 % of the test development cycle time
Getting the optimum test combinations	5 % of the test development cycle time
Developing the test cases	60% of the test development cycle time
Merging of the set-up scripts	5 % of the test development cycle time
Executing the tests and collecting data	5 % of the test development cycle time
Code coverage analysis, test enhancement and test re-run	10 % of the test development cycle time

The table below provides the comparison between the conventional method and OA method with respect to lines of test code, test execution time and code coverage. It is assumed that the feature/product will be of medium complexity and code size.

Description	Conventional Test case	OA test case
Lines of code	100%	30-40% of the conventional test case
Test execution time	100%	~ 30% of the conventional test case
Code coverage	Industry norm is 60-70%	Can be as high as 100% (with the support of manual test cases)

An experience using OA

Example1: Consider a system which has three modules called A, B and C. Module A is the main module which calls the modules B and C depending on the logical state combinations of three parameters X, Y and Z. The parameters X, Y and Z can take the two possible logical states say P and Q. We need to come up with test cases to test whether module A calls the correct module. Generally it requires 2^3 = 8 combinations as shown below, to be tested and hence it requires 8 test cases.

Parameter X	Parameter Y	Parameter Z
State P	State P	State P
State P	State P	State Q
State P	State Q	State P
State P	State Q	State Q
State Q	State P	State P
State Q	State P	State Q
State Q	State Q	State P
State Q	State Q	State Q

If we closely look at the combinations listed, the pair-wise combinations of Parameter Y and Parameter Z get repeated for each of combination of Parameter X. Hence there are totally four pair-wise combinations of Parameter Y and Parameter Z, which become redundant. As per the basic assumption, complex interactions are not the sources of defects and hence these redundant combinations can be removed by using the two levels and three factors Orthogonal Array. Following table lists the combinations arrived after using the OATS.

Parameter X	Parameter Y	Parameter Z
State P	State P	State P
State P	State Q	State Q
State Q	State P	State Q
State Q	State Q	State P

If we closely look at the combinations listed, this has covered all the possible pair-wise combinations of Parameter X and Parameter Y, Parameter Y and Parameter Z and also Parameter X and Parameter Z. It also covers 4 complex combinations. The total number of combinations used is only 4, which is only half of the overall possible combinations thus reducing the testing effort by 50%. As the number of variables involved in the test increases, this percentage tends to go down even further. The combinations that are particularly suspicious i.e. the combinations that have strong possibility of errors can be included into the pair-wise combinations.

OA Test case sustenance strategy

After the product/feature reaches the market, generally, the field errors, customer feed back, product/feature enhancement etc carried out either through DR (defect report) process or RFC (request for change) process. This results in addition of test cases to test the new code. Very often the newly generated test cases to verify the code changes in the feature/product are added to existing optimized test suites directly without any optimization. This practice may defeat the purpose of optimized test concept using OA at times. It is time consuming to identify Parameters and Levels for small fixes, use tools like Minitab to define the optimization etc. In such cases it is advised to revisit the test case in a fixed interval, based on the field error rate or the feature enhancement, to address the newly added test for optimization. Though it looks like taking additional effort, in practical it is very small when compare to the effort and time what is saved because of the optimization.

Benefits of OA

Productivity Improvement:

Implementation time is less: The implementation time is less as the test cases are less. Even though the test suit count is less, the combination will be optimum and covers the basic feature completely. This is evident from the fact that only 38 test cases of 1,367 LOC were written compared to the existing 117 test cases of 28,122 LOC to test the join index feature.

Execution time is less: Since the test cases are less in new test suite compared to the conventional methods, test execution time is less. The existing tests used to take 8 hours to complete where as the new test cases runs in just 25 minutes to test the join index feature.

Result analysis takes less time: Since the number of test cases and the LOC for the test cases is less, the result file size is going to be less compared to the old tests. Hence time taken to compare the result files with the standard files is going to be less. This will further reduce the overall test execution time.

Increase in overall productivity: All the above points provide evidence to the fact that the OA methodology increases the overall productivity since lesser number of test cases are written. Implementation time, test execution time and the result analysis time is less compared to the other methods. Hence, OA concept increases the overall productivity by 40%-50%.

Quality: High code coverage: As evidenced in the join index experience, OA methodology covers close to 95% of the feature code compared to the usual methods. This is achieved with less number of test cases and with less execution time. For the remaining 5% of the code, tests have to be written manually. In this way, close to 100% of the feature code is covered.

Right the first Time: In a conventional method of Test case development, the Test Design mostly depends on the capabilities of the test developer. This might result in redesign, large review defects and thus impact the over all quality of the Test cases. This is avoided to a large extent if OA is used. Since the parameters and levels are defined after good analysis and input from the feature/product developers themselves, this can result in Right the First Time delivery and thus reduce the probabilities of redesign, and review rework.

Improvement in Capacity Constraints: Saving in Infrastructure resources: Since the adoption of OA methodology results in reduced code size, the result files and the disk space required to store them will be considerably less. As the reduced code size results is reduced execution time, the machines resource requirement time is very less. These resources can be ploughed back to do more tests and more analysis. The

capacity constraint due to Infrastructure availability on QA activities is relaxed.

Saving in Human resource: The labor required for development, execution and result analysis would very less because of the reduced code base, execution time and result analysis time. This helps in increasing the frequency of the test cycles.

Quicker To Market: Since there are possibilities in the reduction of over all test cycle duration, the product can reach the market quickly. This also helps in Time to Market.

Helps in Meeting customer expectation: Since the test execution time is less, the bug fix cycle would be considerably reduced. This will drastically increase the customer response time and thus result in high customer satisfaction. This helps product companies plan their releases in tune with customer needs, rather than based on QA capacity.

Conclusion

It is beneficial to the Software product testing to make use of the Orthogonal Arrays. The only constraint would be the applicability of the concept to certain types of products.
There are 3 areas in a product life cycle management we can get benefited by adopting OA.

Test case development for New Feature/Product Re designing the existing test cases Sustenance of the test cases thus developed.

In all the three areas, Software product companies can enjoy the benefits of OA, which were detailed throughout this document. The most critical portion of the adoption process is the product understanding and modelling the product/ feature as Parameters and levels. The care and efforts put in to develop this model will drive the success of the test suite design. If the Software product company does not have enough experience with OA and Six Sigma concepts, it will be worthwhile investing in hiring professionals trained in these concepts and experienced in Software testing.

References

- Design of Experiments using the Taguchi Approach
- A primer on the Taguchi Method
- Shishank Gupta, “Parametric Test Optimization”, Software Testing Conference, 2002.
- B. Beizer. “Software Testing Techniques.” Van Nostrand Reinhold, 2nd edition, 1990.
- http://www.paiwise.org/tools.asp
- http://www.developersense.com/pairwiseTesting.html

Standardized and used by any company for their data cleaning and sanitizing operations. It scales well, is modular and follows software engineering principles. The framework was proved to work on a large dataset in an industrial setting.

References

[1] Dean, J. and S. Ghemawat (2008). Mapreduce: simplified data processing on large clusters. Commun. ACM, 51(1), 107–113. ISSN 0001-0782. URL http://doi. acm.org/10.1145/1327452.1327492.

[2] Agrawal, D., S. Das, and A. El Abbadi, Big data and cloud computing: current state and future opportunities. In Proceedings of the 14th International Conference on Ex-tending Database Technology, EDBT/ICDT ’11. ACM, New York, NY, USA, 2011. ISBN 978-1-4503-0528-0. URL http://doi.acm.org/10.1145/1951365. 1951432.

[3] Jacobs, A. (2009). The pathologies of big data. Commun. ACM, 52(8), 36–44. ISSN 0001-0782. URL http://doi.acm.org/10.1145/1536616.1536632.

[4] El Akkaoui, Z., E. Zimànyi, J.-N. Mazón, and J. Trujillo, A model-driven framework for etl process development. In Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP, DOLAP ’11. ACM, New York, NY, USA, 2011. ISBN 978-1-4503-0963-9. URL http://doi.acm.org/10.1145/2064676. 2064685.

[5] Shvachko, K., H. Kuang, S. Radia, and R. Chansler, The hadoop distributed file system. In Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), MSST ’10. IEEE Computer Society, Washington, DC, USA, 2010. ISBN 978-1-4244-7152-2. URL http://dx.doi.org/10.1109/MSST. 2010.5496972.

[6] Hecht, R. and S. Jablonski, Nosql evaluation: A use case oriented survey. In Pro-ceedings of the 2011 International Conference on Cloud and Service Computing, CSC ’11. IEEE Computer Society, Washington, DC, USA, 2011. ISBN 978-1-4577-1635-5. URL http://dx.doi.org/10.1109/CSC.2011.6138544.

[7] Chaudhuri, S. and U. Dayal (1997). An overview of data warehousing and olap tech-nology. SIGMOD Rec., 26(1), 65–74. ISSN 0163-5808. URL http://doi.acm. org/10.1145/248603.248616.

[8] http://en.wikipedia.org/wiki/Extract,_transform,_load