2015 International Conference on Industrial Instrumentation and Control (ICIC)
College of Engineering Pune, India. May 28-30, 2015

Effective Test Strategy for Testing Automotive Software

Sujit Sopan Barhate
Head Test Laboratory
Hella India Automotive Pvt. Ltd.
Pune, India
Sujit.Barhate@Hella.com

***Abstract*— Electronic content is increasing in automobiles day by day. Functionalities like Air Bags, Anti-lock Braking, Driver Assistance Systems, Body Controllers, Passive entry Passive Start, Electronic Power Steering etc. are realized electronically with complex software. These functionalities are related to automobile system safety. Hence, safety is one of the key issues of future automobile development. Risk of system failure is high due to increasing technological complexity and software content. The software shall be tested well to arrest almost all the defects. This paper explains a test case development and execution strategy based on practical implementation. It explains how test case reduction using Taguchi method, prioritization of test execution and automation help to make testing effective. It also demonstrates how maximum defects are discovered in short time.**

Keywords—Test Strategy; Test Prioritization; Taguchi Method; Test Automation

I. INTRODUCTION

Recently, news buzzed in automotive world: Five deaths and more than 30 injuries due to software defect in air bags. It caused, recall of 8 million vehicles from 10 automakers [21]. Many such incidences happened in recent past [22], [23], [24], [25]. Software defect in complex automotive systems are causing fatal accidents. Hence, automotive software needs to be defect free. Exhaustive testing is required to ensure defect free software. However, exhaustive testing is impossible. Number of test cases needed for exhaustive testing is extremely large [1], [10].

Testing is very critical component because any single fault could potentially have devastating consequences. Testing is defined as the process of executing a program with the intent of finding errors [10]. However it is impractical to find all the mistakes/errors in the program. Moreover, projects neither have infinite resources nor time to carry out exhaustive testing [3]. Therefore, we need to think of economics of testing and come up with strategies to combat the challenge.

In order to achieve a goal of automotive software testing, a test strategy has to be developed. The strategy shall take care of following points:

- Write "good" test cases. A good test case is one that has a high probability of detecting an as yet undiscovered error [10].
- Prioritize test case execution
- Ensure requirement coverage
- Ensure equivalence partitioning and Boundary Value Analysis (BVA)
- Update test cases to increase testing depth with increase in software maturity
- Avoid pesticide paradox [12]
- Automation in writing test cases and test execution

Test coverage is very important to validate functioning of software for various combinations of inputs. It is often taken care in structural testing. Most of the static testing tools provide metrics of the test coverage like statement, decision/branch, MC/DC (Modified Condition/Decision Coverage) [11]. However, equivalence portioning and BVA are often not tested well. Hence, careful attention towards these aspects is necessary. Testing depth shall increase as maturity of software increases. Testing shall be intensified by adding more test scenarios for boundary values when software had proved on basic test cases. This will also help in avoiding pesticide paradox of test cases. Usually, test cases written once are used again and again for future software releases as part of regression. Such regressions will not discover new defects in the software. Defects will be immune to the test cases. It is the similar phenomenon which is observed in pesticides. Pests get immune to particular type of pesticide when used over a period of time. Frequent test case revisions are required to avoid this paradox. It may generate huge number of test cases. However, executing all the test cases may not be possible every time. So, test cases shall be prioritized. High priority tests must be executed first to find maximum defects. Lower priority test cases can be executed later point of time. Lower priority test cases may need more time for execution with lesser yield in terms of defect detection. Test prioritization techniques are explained by many authors. Coverage based prioritization [8], MC/DC pairs based prioritization [11], 4C (Customer Requirement-based techniques, Coverage-based techniques, Cost Effective-based techniques and Chronographic History-based techniques) classification of prioritization [20] etc. are few of the examples. They have also predicted the defect rate [7], [9], [17], [18]. However, this paper describes, test prioritization based on test case complexity and Orthogonal Arrays (OA) test cases. The paper demonstrates defect detection per priority level.

978-1-4799-7165-7/15/$31.00 ©2015 IEEE

In order to comply everything mentioned so far cannot be achieved without a high degree of automation. Automation will not only save time but also ensure uniform implementation. It will also help in reducing person dependency in the process.

This paper describes a test strategy. Approach to write test cases, test case reduction, prioritization and test automation are explained in the paper. Effectiveness of the strategy is discussed based on results obtained from the implementation.

In the remainder of this paper, test strategy is described in section II. Results and observations obtained from the project in terms of defect detection, time saving are discussed in section III. The paper is concluded in section IV based on the results and resource requirement.

II. TEST STRATEGY

At Hella India Automotive, a test strategy was developed for complex automotive electronics projects. It was implemented in one of the very challenging projects. The project had stringent cost and time constraints. Testing needs 50% of the budget in a project development phase [15]. Hence, this part had to be efficient. Fig. 1 shows the process steps for writing test cases effectively to get high yield in short time. The process steps and its implementation are explained as follows:

A. Requirement Analysis

Understand complete functionality from the requirements. Analyze requirements and understand interrelations of inputs which affect the outcome.

B. List inputs & outputs

Make a list of Inputs and Outputs (IOs) in the given requirement. Write the data range for the IOs and identify values to be tested to validate the requirement. For example, the requirement is:

"*Activate ABS motor if brake pedal is pressed and vehicle speed is above 25 km/hr when the ignition is ON.*"

Inputs in the requirement are brake pedal (*Brake_Pedal_Sw*), Ignition (*Ign_SW_Status*) and vehicle speed (*Veh_Spd*). Output in the given requirement is ABS motor (ABS_Motor).

- *Brake_Pedal_Sw* is a Boolean variable hence can hold two values 0 or 1. Where 0 is "not pressed" and 1 is "pressed".
- *Ign_Sw_Status* is an enumerated data type having five values 0 to 4. Ignition ON is enumerated to 4.
- *Veh_Spd* is an integer and its values are limited from 0 to 255.

Basic functionality of the software can be tested with following input scenarios:

- *Brake_Pedal_Sw* shall be tested for both 0 and 1.
- *Ign_SW_Status* shall be tested for 4 and any value other than 4
- *Veh_Spd* shall be tested for two values, less than 25 and greater than 25.

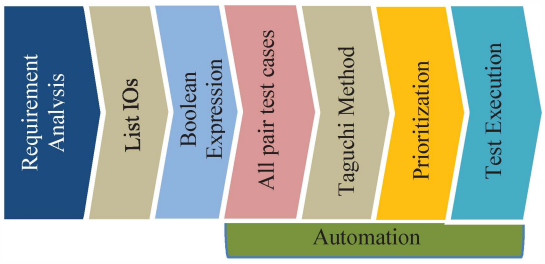

Fig. 1. Test strategy process steps

When the software satisfied the tests, testing depth shall be increased. Testing depth can be increased by testing more input scenarios. Next step could be to add input scenarios to test equivalence partitioning [12], [15], [17]. In order to test equivalence partitioning, *Veh_Spd* shall be tested for additional values like 10, 100, 200.

If the software proves to be mature in these scenarios then it shall be tested for BVA [12], [15], [17]. Following additional input scenarios can be used for BVA.

- *Ign_SW_Status* can be tested for -1, 0, 3, 4, 5
- *Veh_Spd* can be tested for -1, 0, 24, 25, 26, 255, 256.

C. Boolean expression

Convert requirement in a Boolean expression. The requirement mentioned in an example can be converted in Boolean expression as:

$$((Ign\_Sw\_Status == IGN)\ \&\&\ (Brake\_Pedal\_Sw == 1)\ \&\&\ (Veh\_Spd > 25)) = ABS\_Motor\_ON$$

The expression can be represented as:

$$(A\ \&\&\ B\ \&\&\ C) = ABS\_Motor\_ON$$

Where, "A" is condition *(Ign_Sw_Status==IGN)*, "B" is *(Brake_Pedal_Sw == 1)* and "C" is *(Veh_Spd > 25)*.

D. All pair test cases (Truth Table)

Write a truth table for the Boolean expression. Values to be tested depend on the depth of the testing as discussed in point B. Various possible combinations of all inputs will be an all pair truth table. Solve the Boolean expression for all input possibilities to get expected Boolean output. However, expected outcome for other data type output variable shall be analyzed manually. Every row of the truth table is a test case. It will generate huge number of test cases. Test cases generated for above example for BVA input scenario is 100. Number of test cases increases exponentially for complex expressions. According to counting principle, if out of all variables n_1, n_2, n_3 ... have values m_1, m_2, m_3 ... respectively then possible Number of Test Cases (N_{TC}) will be:

$$N_{TC} = {m_1}^{n_1} \times {m_2}^{n_2} \times {m_3}^{n_3} \times \ldots\ldots$$

In the example, variable *Ign_Sw_Status* has 5 values, *Brake_Pedal_Sw* has 2 values and *Veh_Spd* has 10 values then the number of all pair test cases is

$$N_{TC} = 5^1 \times 2^1 \times 10^1 = 100$$

E. Test Case Reduction using Taguchi Method

Execution of large number of test cases is impractical as time and resources are limited in a project. Hence, test cases shall be reduced to ensure practicality. Taguchi method was used to reduce the test cases.

Taguchi method was developed by Genichi Taguchi, a Japanese engineer. He confined statistical Design of Experiments (DoE) in research and development environments. The method is based on OA. Basic of OA is explained in Fig. 2. Basics of OA and its applications are explained by many authors [2], [4], [6], [7]. OA testing strategy is a systematic statistical way of testing pair-wise testing [7]. It is used in product engineering and manufacturing. The technique is also used in software testing [4], [7], [13], [19]. In the discussed example, OA results in 50 test cases. Hence, number of test cases reduced by 50%.

F. Test Case Prioritization

Number of test cases is huge in a project for all requirements. Tremendous time is required to execute these test cases. Hence, considering economics of the test cases they are prioritized in four levels. Table 1 describes the priorities and Fig. 3 shows the test pictorial view for better understanding.

"Prio 1" is the highest priority and "Prio 4" is the lowest. Tests are executed according to the priorities to maximize the objective of finding defects early. "Prio. 1" test cases are the simple combination of inputs which affects the output. It needs short time for execution. Yet it will be effective to find most obvious defects in the software. "Prio. 2" test cases ensure validation of critical requirements. It tests requirements for more scenarios and complex input combinations which affect the output. "Prio 3" is remaining OA test cases. "Prio 4" test cases are remaining test cases, usually from 'All Pair'. These test cases shall be executed to achieve maximum possible exhaustive testing with BVA.

"All pair" test cases are a superset of test cases. OA test cases are subset of all pair test cases. Most of the OA test cases are prioritized at level 2. Few important input combinations not selected by OA. Such combinations are added to the priority level from "All pair" test cases. It ensures all critical combinations of inputs are tested. Very simple input combinations which affect outcome are selected as part of priority level 1. Priority level 1 is subset of OA test cases. Test cases remaining from OA are prioritized 3. Remaining test cases from "All pair" are prioritized 4.

Run \ Factor	A	B	C
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

Levels

$L_{Runs}(Levels^{Factors}) = L_4(2^3)$ Taguchi Orthogonal Array

Fig. 2. Basics of Taguchi Method

TABLE I. PRIORITY DESCRIPTION

Priority	Description/selection
Prio 1	Small set of test cases. Simple input combinations which affect output.
Prio 2	Test cases which are covering all the requirements. Majority of the test cases will be from OA.
Prio 3	Remaining set of test cases from OA.
Prio 4	Remaining set of test cases from all pair.

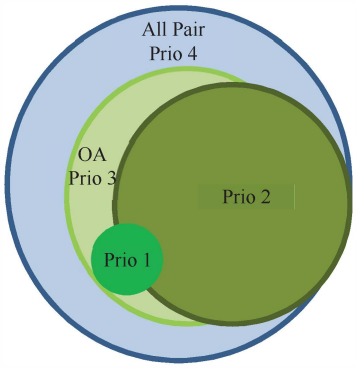

Fig. 3. Test prioritization

G. Automation

High degree of automation is a key for successful implementation of the test strategy. Automation was developed for writing test cases and execution of the test cases.

- Test Case Development Tool

  Fig. 4 shows "Test Vector Generation Tool". Requirements are converted into logical (Boolean) expressions. Inputs and outputs involved in those requirements are listed in an excel sheet with the test values. The tool generates test vectors for "All pair" test cases. Orthogonal arrays are used to reduce the test set automatically in the tool. Fig. 5 shows another tool called "Test Case Generation Tool". It converts test vectors into test cases in the prescribed format. The test cases have to be executed on Hardware in Loop (HIL) test setups. Hence, the test cases are converted in executable format.

- Test Execution

  Test cases are executed on HILs using "Hella Test Framework" (HTF) tool. HTF is a test case scheduler. It executes test cases on fully automatic HILs. Every step in the test is evaluated automatically for "Pass" or "Fail". Failed test cases are analyzed by test engineers on manual test setups and issues are raised for confirmed failures.

III. Results & Discussion

The test strategy was implemented successfully in the project. Following are the results and observations:

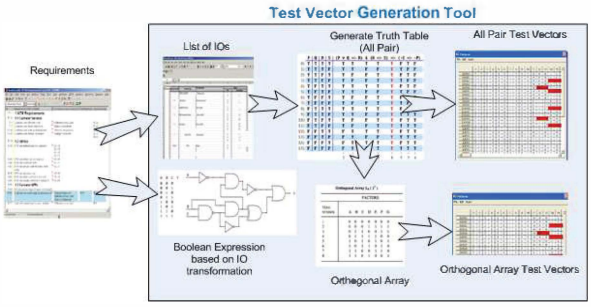

Fig. 4. Test Vector Generation Tool

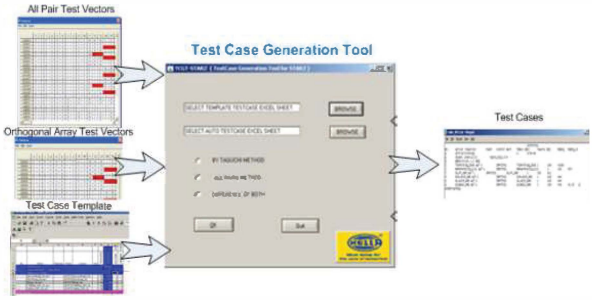

Fig. 5. Test Case Generation Tool

A. *Reduction in efforts*

More than 12000 test cases were developed for around 4000 requirements. Usually, 20 minutes are required to write a test case manually. Instead, due to automation less than 5 minutes are required to write a test case. Hence, around 75% efforts are saved in writing test cases.

More than 95% automation in test execution is achieved using HTF. Hence, manual test execution needed only 5% efforts. Test result analysis time was dependent on number of failures reported by the automated test execution. Failures were analyzed to confirmed defects. False failures were fixed in the automation to achieve continuous improvement in the automation level.

B. *Reduction in test cases*

Around 40% overall test cases reduction is observed due to OA in a complete test set. 50-70% test case reduction is observed in complex requirements having more number of inputs. However, there was not significant reduction in test cases for simple requirements having less number of inputs.

C. *Test case distribution*

Fig. 6 shows distribution of test cases according to the priority levels. Only 5% tests were selected for priority 1. Priority 2 is the biggest test set having 46% test cases. But it covers all critical scenarios for the testing. Priority 3 is remaining OA test cases contributing to 15%. Priority 4 is the second biggest test set. Test execution of all the priority levels ensures almost exhaustive testing as it covers all possible combinations of input variable and their values under test.

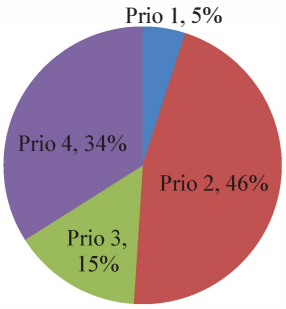

Fig. 6. Test case destribution

D. *Effective defect discovery*

600 defects were discovered by executing the test cases. A radar chart in Fig. 7 showing relation of defect discovery with number of test cases executed for each priority. Following are the key observations:

- Defect line (red) is above test cases line (blue) for priority 1 and 2. This indicates more defects are discovered in lesser number of test cases.
- Defect line (red) moved below test cases line (blue) for priority 3 and 4. This indicates, more number of test cases is executed but defects discovery is lower.
- Priority 1 detected 20% of defects in 5% test cases. It takes only 6 hours to execute the test cases. It helped in reporting defects quickly hence, it helped in reducing time for rework.
- Priority 2 test cases discovered 70% defects in 46% test cases. Hence, proved to be effective test cases.
- Priority 3 test cases discovered 4% defects after executing 15% test cases.
- Priority 4 test cases could discover 6% defects after executing 34% test cases.
- Priority 1 test cases proved to be most efficient test cases in terms of defect discovery versus number of test cases and time required for test execution.
- Priority 1 and 2 higher yield in terms of defect defection than priority 3 and 4.

E. *Avoid pesticide paradox*

To overcome pesticide paradox, test cases need to be regularly reviewed and revised, and new tests need to be written to exercise different parts of the software to potentially find more defects [12]. It is achieved by increasing depth of testing as discussed in section II, B. Regularly revising test cases by changing input scenarios was possible due to the high degree of automation.

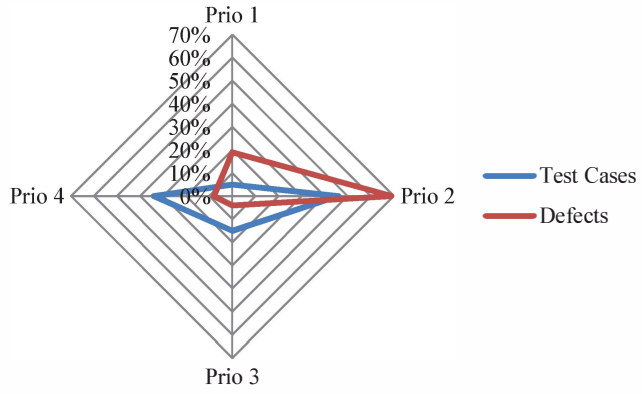

Fig. 7. Test Cases and Defects relation

IV. Conclusion

- Advantage of systematic approach in writing test cases, use of statistical method OA and high degree automation demonstrated efficient testing.
- OA test cases demonstrated very important role in defect detection. It could discover around 90% defects in around 60% test cases.
- Test case prioritization logic demonstrated the effectiveness. 20% of defects discovered in 5% test cases within 6 hours of test execution. Priority 1 and 2 proved to be *good* test cases as they discovered more defects in lesser test cases hence in lesser test execution time. Whereas, lower priorities 3 and 4 tested software for many more scenarios and could discover defects related equivalence partitioning and BVA.
- A systematic approach of developing test cases for equivalence partitioning, BVA and taking care of pesticide paradox helped discovering new defects.
- High degree of automation in all the phases made this possible. Increasing testing depth as per increase in maturity of software and writing new test scenarios to avoid pesticide paradox lead to huge number of new test cases. It was achieved due to automation in writing test cases. 75% efforts were saved in this phase. 95% automation in test execution made it possible to execute all the test cases.
- More than 50% increase in productivity due to lesser number of resources as compared to similar projects which did not use the test strategy. Five test engineers worked in the project whereas minimum ten are required in similar projects. The tests were executed on two HILs compared to five in similar projects.

References

[1] Hsu Mon Maung, Kaythi Win, "An Efficient Test Cases Reduction Approach in User Session Based Testing", International Journal of Information and Education Technology, Vol. 5, No. 10, October 2015

[2] Lilly Raamesh, G. V. Uma, "An Efficient Reduction Method for Test Cases", International Journal of Engineering Science and Technology, Vol. 2(11), 2010, 6611-6616

[3] Naveen Gautam, Ratna Babu Chinnam, Nanua Singh, "Design reuse framework: a perspective for lean development", Int. J. Product Development, Vol. 4, No. 5, 2007

[4] Kedar Phadke, Madhav Phadke, "Utilizing Design of Experiments to Reduce IT System Test Cases", Crosstalk, The Journal of Defense Software Engineering, Vol.24 No.6, Nov/Dec 2011

[5] Kuo-Chung Tai, Yu Lei, "A Test Generation Strategy for Pairwise Testing", IEEE Transactions on Software Engineering, Vol.28 No.1, January 2002

[6] Robert Mandl, "Orthogonal Latin Squares: An Application of experiment Design to Compiler Testing", Communications of the ACM, Volume 28 Issue 10, Oct. 1985

[7] L. Lazic, N. Mastorakis, "Orthogonal Array application for optimal combination of software defect detection techniques choices", WSEAS Transactions on Computers, Issue 8, Volume 7, August 2008

[8] S. Yoo, M. Harman, "Regression T esting Minimization, Selection and Prioritization : A Survey", Software Testing, Verification and Reliability, 2007, 00:1-7

[9] Ahmed M. Salem, Kamel Rekab, James A. Whittaker, "Prediction of Software Failures Through Logistic regression", Information and Software Technology 46(2004) 781-789.

[10] Glenford J. Myers, "The Art of Software Testing, Second Edition", Published by John Wiley & Sons Inc.

[11] James A. Jones and Marry Jean Harrold, "Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage", IEEE Transactions on Software Engineering, Vol 29, No. 3, March 2003

[12] Dorothy Graham, Erik Van Veenendal, Isabel Evans, Rex Black, "Foundations of Software Testing", ISTQB Certification

[13] K. Gopalkrishnan, D. R. Stinson, "Applications of Orthogonal Arrays to Computer Science", Proc. of ICDM 2006, Pages 149-164

[14] Mirko Conrad, Ines Fey, "Systematic Model Based Testing of Embedded Automotive Software", Electronic Notes in Theoretical Computer Science 111(2005) 13-26

[15] Anupriya Jain, Sachin Sharma, Seema Sharma, Deepti Juneja, "Boundary value analysis for non-numerical veriables: Strings", Oriental Journal of Computer Science & Technology, Vol. 3(2), 323-333 (2010)

[16] Huelya Bayrak, Aslihan Alhan, "On the Construction of Orthogonal Arrays", Hacettepe Journal of Mathematics and Statistics, Volume 31 (2002), 45-51

[17] Mohd. Ehmer Khan, "Different Approaches to Black Box Testing Technique for Finding Errors", International Journal of Software Engineering & Applications, Vol.2, No.4, October 2011

[18] Shaikh Umar Farooq, S.M.K. Quadri, "Evaluating Effectiveness of Software Testing Techniques With Emphasis on Enhancing Software Reliability", Journal of Emerging Trends in Computing and Information Sciences, Vol.2, No.12, December 2011

[19] R. K. Gupta, Tapan Bagchi, "Harnessing Taguchi Methods in Software Development", Production and Operations Management International Conference, Dallas, USA, May 2007

[20] S. Roongruangsuwan, J. Deangdej, "Test Case Prioritization Techniques", Journal of Theoretical and Applied Information Technology, 2010

[21] http://www.usatoday.com/story/money/cars/2014/11/13/takata-airbags-honda-death-five/18963827/

[22] http://www.autonews.com/article/20131009/OEM11/131009855/toyota-speed-up-defect-caused-death-injury-lawyer-tells-jury

[23] http://www-odi.nhtsa.dot.gov/owners/SearchResults?searchType=ID&targetCategory=R&searchCriteria.nhtsa_ids=14V054000

[24] http://www.bizjournals.com/triangle/morning_call/2014/02/toyota-recalling-19m-prius-cars-for.html

[25] http://www.autonews.com/article/20131102/OEM11/131109934/honda-to-recall-344000-minivans-in-u.s.-due-to-braking-glitch