International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 12– No.33, September 2020 – www.ijais.org

A Method and Apparatus to Design Optimal Scenario based Test Model for System Software

Susanta Dutta
Sr. Software Engineer
Hewlett Packard Enterprise
Bangalore, Karnataka,
India-560048

Sathesh Babu Rangaraj
Sr. Software Engineer
Hewlett Packard Enterprise
Bangalore, Karnataka,
India-560048

Mahantesh Chiniwar
Sr. Software Engineer
Hewlett Packard Enterprise
Bangalore, Karnataka,
India-560048

ABSTRACT
System software is a type of computer program that is designed to run on a computer hardware, such as server, storage or network devices. Some of the examples of system software are operating system, firmware, device drivers, management software etc. To ensure high level of quality of these system software, only functional and non-functional testing (reliability, compatibility, performance and scalability) may not be sufficient.

To uncover errors in the system software that can only occur in certain conditions and situation, a scenario based test model should be derived based on underlying hardware configurations, state of the hardware, user actions being performed on it, etc.

The present solution is a method for designing optimal test model considering multi-dimensional aspects. As an example, three dimensions are considered to describe the concept in this paper.

General Terms
This solution relates in general to software test engineering to ensure quality of a system software, such as Operating system, Firmware, device drivers, hardware management software or utilities. The present solution is directed to the designing of an optimal scenario based test model for the system software.

Keywords
Software Engineering, System Software, Defects, Quality, Testing, Scenario based testing, System test. Test cases

1. INTRODUCTION
Scenario based testing simulates hypothetical stories involving multiple complex test steps. This testing can be included along with functional and non-functional testing. To ensure quality of system software, it's important to develop a test model to derive those scenarios and execute them to uncover defects that are observed when different states of the product intersects and conflicts with user actions on a specific hardware configuration.

This technique solves the problem of determining a systematic approach to design optimal test model for a system software:

The test model for a system software consists of followings:

a) Configuration of the underlying hardware. This can be different model, add-on hardware component or any specific hardware configuration.

b) State of the hardware. It can be either running some processes or jobs, or got into a partial failure or degraded state. The state of the hardware can be one or more in combination.

c) User action on the hardware using the software. This can be due to usages of the product. It can be either manual or automatic action. There can be one or more user actions in sequence or in parallel.

2. BACKGROUND
While analyzing some of the customer found issues for a storage management software, found that all the issues were not caught during regular functional, reliability, compatibility, performance and scalability testing. Majority of the issues occurred only in certain conditions and situations when storage system is in particular state and user performs an action on it.

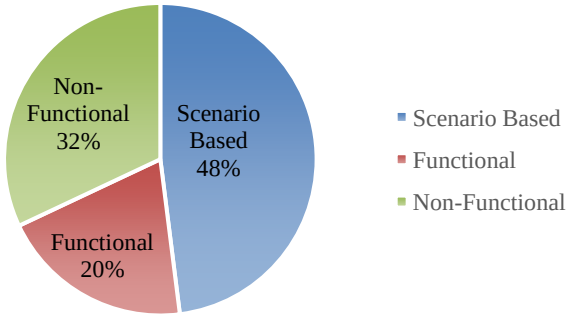


Figure-1: A Case Study Data

A case study, shown in Figure-1, was performed on customer found issues (CFIs) on a selected module, the interesting pattern shows 48% of the issues were due to usages of the system software in some specific scenarios.

Some example of real customer scenarios are:

- Formatting a storage volume hangs when underlying storage logical disk is initializing. Customer had to wait for hours till initialization is completed.
- Graceful rebooting of a NAS storage system gets stuck when one of the network link is in failed state. As a result NAS services are not available for client access file shares.

- Running defrag on a degraded thin provisioned volume causes memory leak. Customer system used to crash intermittently.

The pattern among above scenarios is each scenario has elements of user action, specific state and configuration

These conditions and situations are known only after customers reported, but challenge is to determine such conditions and situations during test design phase with right priority.

A method can be derived to design an optimal test model from all possible states and user actions on a system software [1] [2] [3]. This technique solves the problem of determining a test model for a system software with a systematic approach.

3. DETAILED DESCRIPTION OF THE SOLUTION

The following detailed description is directed to concept for designing optimal test model for effective testing of a system software.

This method provides a systematic approach to derive all test cases from various configuration, models, versions etc., different state and user actions on it.

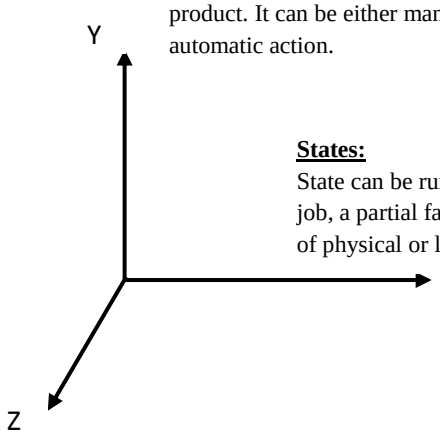
Figure-1 is a case study data that shows percentage of customer found bugs – 48% bugs were encountered due to some specific scenarios, others are functional or non-functional issues.

Figure-2 shows aspects of an illustrative operating environment along with usages of the hardware product, which creates scenarios for a product. XZ-plane represents the operating environment that consists of various configurations and different states [4] of the hardware product. In the approach of designing scenario based test cases, creation of operating environment is an important aspect. This can also be referred as test environment.

An action is performed on a product as per its usage of features and functionalities. Y-Axis represents user actions on it, one of the important aspect of scenarios, according to the embodiments presented herein.

Now, considering operating environment and user actions, XYZ – Space, is an illustration for the scenarios of a system software considering all three aspects of a hardware product.

User Actions:
User actions can be due to usages of the product. It can be either manual or automatic action.



States:
State can be running any process or job, a partial failure or degraded state of physical or logical object.

Configurations:
Configuration can be different model or version of a product, large or medium configuration, along with any specific configuration option that is turned on or off

Figure-2: Scenarios for a product

3.1 Preparation Phase

Figure-3 shows template table illustrating list of possible hardware configurations, states and user actions along with weight value.

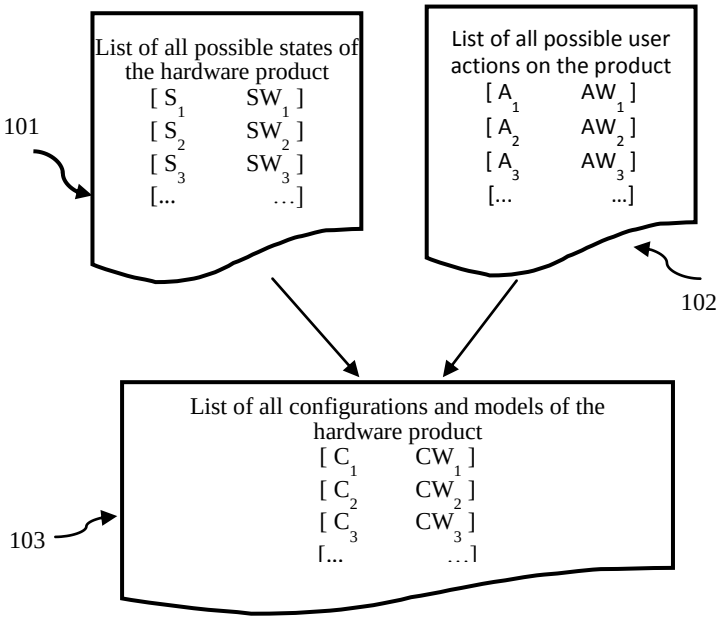


Figure-3: List of possible hardware configurations, states and user actions along with weight value

As a first step of preparation phase, identify and list all possible states that product can get in. This is shown in block 101. The state of the product can be either running some processes or jobs, or got into a partial failure or degraded state. States are denoted in the block 101 as S1, S2, S3 and so on. Some of the example of states for computer hardware or software can be, link down, remote system is unreachable, a hardware component failed. Each state can have a weightage against other states, considering probability of occurrence,

impact and criticality of it. A numeric value in the scale of 1 to 10 with 10 being the most critical and commonly observed state, can be assigned as a weightage to each state. State weighted are denoted in the block 101 as SW1, SW2, SW3 and so on.

Similar to states, identify and list all possible user actions for the given product. The actions are generally due to usages of the product. This is shown in block 102. For example of computer hardware can be rebooting, installing software, refreshing software graphical user interface etc. Based on criticality, assign weightage value to each of them. Those are denoted in the block 102 as AW1, AW2, AW3 and so on.

Product engineer, technician or subject matter expert (SME) with working experience with the product is the right person to list all of the above.

Last step in preparation phase is to list all configurations, models of the hardware that are available for qualification. In the block 103, configurations are denoted in the block 103 as C1, C2, C3 and so on and configuration weightages are denoted as CW1, CW2, CW3 and so on.

As depicted in Figure-4, there can be some combination of any two or three type of parameters which are unsupported or invalid. For example, when a server link is down, command cannot be sent to it or a specific configuration may not support some user action. These combination can be called out in a separate table for exclusion from final test set.

For a given product, a library file or database tables with all these parameters can be maintained for future test design.

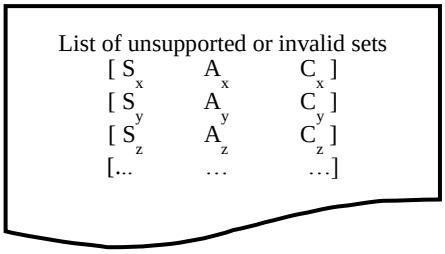


Figure-4: Exclude Table

3.2 Test Case Design Phase

Figure-5 is a logical flow diagram illustrating the method for designing optimal test model [5] [6], according to embodiments presented herein

Logic in flow diagram depicts the selection of each configuration, state of the hardware product along with the actions. Block 201 is the outer most loop for each configurations that was listed in listed in 103 during preparation phase. Other two inner loops, block 202 and 203, are for states of hardware (block 101) and list of actions (block 102). While executing all three loops in the flow diagram, there are two other important considerations need to be made (block 204):

a) Exclude the set that matches with combinations that are listed in exclude table during preparation phase.

b) Multiply respective weight values together to calculate resultant weightage for each set [6]: $SW_i * AW_j * CW_k$

$$SW_i * AW_j * CW_k \rightarrow \{S_i, A_j, C_k\}$$

As last step of the test case design, sort all final sets based on resultant weightage (block 205). This is to determine sort the sets based on their priority.

Each set can be converted to test case.

Here is an example of data set:

Configurations and their weightage:

C1	3
C3	4

Hardware states and their weightage

S1	3
S2	4
S3	2

User Actions and their weightage

A1	3
A2	4
A3	2

Exclude sets:

A2	C1	
S2	A3	
C2	S3	
C1	A1	S1

Logic described in flow diagram creates resultant test sets:

64: C3 -> S2 -> A2
48: C3 -> S2 -> A1
48: C3 -> S1 -> A2
36: C3 -> S1 -> A1
36: C1 -> S2 -> A1
32: C3 -> S3 -> A2
24: C3 -> S3 -> A1
24: C3 -> S1 -> A3
18: C1 -> S3 -> A1
16: C3 -> S3 -> A3

Any test case consist of three fundamental components – Test Environment, Test Steps and Verification steps. From the data generated in block 205, create test cases having at least three parameters in each:

1) Test Environment (Product Configuration with one or more States)

2) Test Steps (User Actions)

3) Verification Steps with expected results

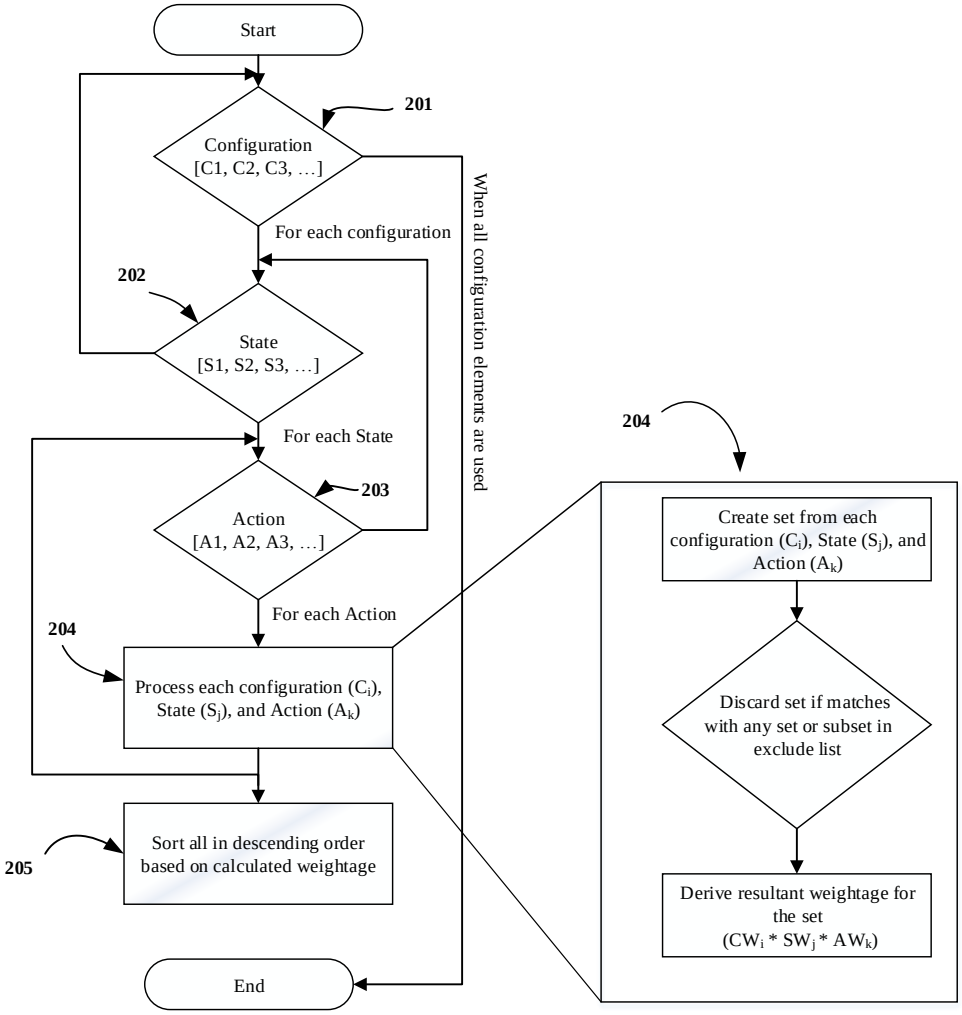


Figure-5: A logical flow diagram illustrating the method for designing optimal test model

4. APPLYING THE TECHNIQUE

This method has been applied in several projects and observed positive results. For the preparation phase and described in Figure-3, one of the storage management software projects data is presented here:

Table 1. Example data for configuration

Configurations	Weightages
RAID_5_volume	4
RAID_1_volume	2
Volume_on_SSD	3

Table 2. Example data for states

State	Weightages
Volume_degraded	2
Disk_failed	3
Network_link_down	2
RAID_rebuild	3

Table 3. Example data for user actions

User action	Weightages
Snapshot_creation	4
Adding_disk	2
Firmware_upgrade	1
Graceful_reboot	1
Defrag	3

The exclude data set identified and presented in the Table 4:

Table 4. Example data for user actions

Configurations	State	User Action
	RAID_rebuild	Adding_disk

When applied the algorithm described in Figure 5, following 48 new test scenarios were generated. Those are listed based on their resultant weightage in descending order:

Table 5. Derived scenarios along with resultant weightage

	Scenarios
48	RAID_5_volume : Disk_failed : Snapshot_creation
48	RAID_5_volume : RAID_Rebuild : Snapshot_creation
36	RAID_5_volume : Disk_failed : Defrag
36	Volume_on_SSD : RAID_Rebuild : Snapshot_creation
36	Volume_on_SSD : Disk_failed : Snapshot_creation
32	RAID_5_volume : Network_link_down : Snapshot_creation
32	RAID_5_volume : Volume_degraded : Snapshot_creation

27	Volume_on_SSD : Disk_failed : Defrag
24	RAID_5_volume : Network_link_down : Defrag
24	Volume_on_SSD : Volume_degraded : Snapshot_creation
24	RAID_1_volume : Disk_failed : Snapshot_creation

- - - - - - - - - - - -

[Rows are truncated for better view]

- - - - - - - - - - - -

8	RAID_5_volume : Volume_degraded : Firmware_upgrade
8	RAID_5_volume : Network_link_down : Graceful_reboot
6	Volume_on_SSD : Network_link_down : Firmware_upgrade
6	Volume_on_SSD : Network_link_down : Graceful_reboot
6	Volume_on_SSD : Volume_degraded : Graceful_reboot
6	Volume_on_SSD : Volume_degraded : Firmware_upgrade
6	RAID_1_volume : Disk_failed : Firmware_upgrade
6	RAID_1_volume : Disk_failed : Graceful_reboot
4	RAID_1_volume : Network_link_down : Graceful_reboot
4	RAID_1_volume : Network_link_down : Firmware_upgrade
4	RAID_1_volume : Volume_degraded : Graceful_reboot
4	RAID_1_volume : Volume_degraded : Firmware_upgrade

Top 30 out of 48 test scenarios selected for test execution. There were total additional 11 critical bugs uncovered in the software from this method of test scenario:

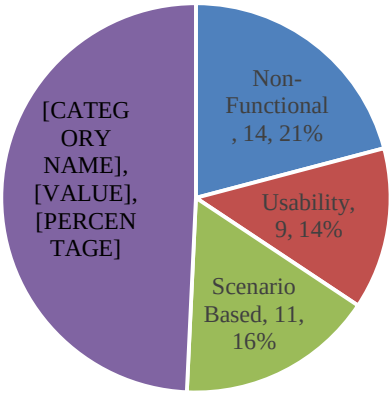


Figure-6: Test result after applying the technique

5. CLAIMS

A method for designing optimal test model for system software from multi-dimensional aspects, of a hardware product configuration, state of the product and user action on the product, said method comprising the steps of:

1) Identify and list all possible states that a hardware product can get in along with their weightage
2) Identify and list all possible user actions as part of product usages along with their weightage
3) List all models and configurations of the product along with their weightage
4) A unique logic to generate valid set of states and user actions for given configuration along with resultant weightage set.
5) An automatic test model design based on logic described in this paper.

6. CONCLUSION

The above described technique guides to design optimal test model for system software. Though this paper explained the method only using three dimensions, for a complex product, scenario based test cases can be designed with even more number of dimensions.

To improve quality of system software, these test cases can be executed to uncover errors in the system software that can occur only in certain conditions and situations.

7. ACKNOWLEDGMENT

We would like to express our sincere gratitude to Mr. Suhas Shivanna, Distinguished Technologist, Hewlett Packard Enterprise, for his constant support, guidance and motivation. It would never have been possible for us to take this analysis and research to completion without his incredible support and encouragement.

8. REFERENCES

[1] Thillaikarasi Muthusamy, and Seetharaman.K. 2014 A New Effective Test Case Prioritization for Regression Testing based on Prioritization Algorithm

[2] R. Krishnamoorthi and S. A. Mary.2009.Factor Oriented Requirement Coverage Based System Test Case Prioritization of New and Regression Test Cases. Information and Software Technology. Vol. 51.No. 4. pp. 799-808

[3] S. Raju and G.V. Uma, 2012. An Efficient Method to Achieve Effective Test Case Prioritization in Regression Testing using Prioritization Factors. Asian Journal of Information Technology, 11: 169-180.

[4] B. Korel, L. H. Tahat and B. Vaysburg, "Model based regression test reduction using dependence analysis," *International Conference on Software Maintenance, 2002. Proceedings.*, Montreal, Quebec, Canada, 2002, pp. 214-223, doi: 10.1109/ICSM.2002.1167768

[5] Phadke, Madhav S. "Planning Efficient Software Tests". Phadke Associates, Inc. Numerous articles on utilizing Orthogonal Arrays for Software and System Testing.

[6] Dustin, Elfriede "Orthogonally Speaking".