System Debug and Validation: Use case Based Perspective

Bhushan Naware
MIG, Intel Technologies India Pvt Limited
Bangalore 560103 India
Email: bhushan.g.naware@intel.com

Arun A Pai
WSS, Intel Technologies India Pvt Limited
Bangalore 560103 India
Email: arun.a.pai@intel.com

Ravinder Singh
WSS, Intel Technologies India Pvt Limited
Bangalore 560103 India
Email: ravinder.m.singh@intel.com

***Abstract* – Concept and Design of systems with sheer complexity at various abstraction levels is becoming tedious and time consuming process. To comply with the expected requirements, subsequent validation and verification becomes even more time consuming and expensive. When it comes to platform level validation and debug, there are various fronts that are to be looked at with great depth. In case of laptops/desktops the system stack includes hardware, silicon, firmware, bios, operating system, various drivers and applications. In complex systems, finding root cause of issues caught at platform validation is challenging and increases debug throughput. In this paper, we will introduce a methodology for validation and debug that could be applied across similar systems. This methodology is bound to shorten the life span of test plan creation, early identification, debug and root cause of issues. This will result in cost saving and shorter time to market.**

Keywords: Test Plan; Use case; Scenario; Win-DBG; JTAG; Silicon Debug.

I. INTRODUCTION

Every year the computing systems are becoming more complex and as a result there is an increase in overall product life cycle time starting from concept, design, development and validation. The validation and platform debug needs to be very efficient and test cases needs to be derived from real use cases in-order to exercise all corner scenarios. The other possibilities can also include stress testing of the systems that has to be done with existing and newer features. Stress and Stability testing consumes the maximum duration of validation as the test duration spans across days, these test increases the workload of the platform validation teams exponentially due to sporadic failures which takes more time to repro. The methodology that is proposed in this paper can be used by extensive number of teams/customers that are working on platform validation; be it original design manufacturer or original equipment manufacturer or the in-house validation teams that are responsible for product readiness and deployment. This paper provides an introduction to the concept of use cases as one of the obelisk of validation metric in order to scale the validation and also make the entire coverage robust and more adaptive. Using the concept of use cases to bolster the system validation, there is a preeminent advantage in the issue debugging and also gauging of coverage across platform which can provide status for overall product readiness with respect to the quality requirements.

Currently the state of the art validation methodologies that are used by original equipment manufacturers and original device manufacturers and also the in-house validation teams is based on feature based approaches and the one proposed here currently is being used for the first time in broad system level context.

The paper is outlined as follows. In section II, the concept of use cases is explained. In section III the generic approach of platform testing and debug via use cases is proposed. In section IV with one of the running examples the concept of use case and usage revelation is brought forward, also ease of debugging of issue is explained in section V. The paper finishes with conclusion in section VI.

II. HYPOTHESIS

The conventional way in which the system validation is performed revolves around the new feature debut, in a particular platform whether it is a hardware or software, then checking if the standalone operation of feature matrix is proper, and if the answer to that is yes; then subsequently it has to be validated and substantiated for the different flows at the platform level. Considering there is sprouting feature list and also the new evolutions of system use patterns; platform test plan intricacy & validation cycle time increases multifold.

In order to mitigate and get the details on the above list of features sorted out, there is a need of mechanism that would give us portable and more systematic way of tracking features at platform, which would eventually touch all the underlying sub-features. Hence, instead of looking at the system from new features standpoint we look at the system from the usage scenarios.

The end-user when aims to use the system, what is the way in which the system is used. Complexity of such flows is taken into account and once that use case list is in place, we have more or less a constant feature list that would cover a particular domain. Once the list of use cases is fairly constant platform over platform there is more transparency in terms of tracking. Further to that depending on a new feature introduction, be it architectural or any software dependent it can be knit into the particular use case; solving both the purposes, keeping the main test tracking list constant and also incorporating the new testing scenarios. We are following the inverted edifice approach to solve the test case intricacy and other allied issues.

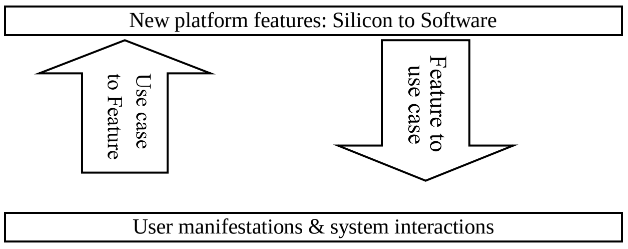

New platform features: Silicon to Software
Use case to Feature
Feature to use case
User manifestations & system interactions

Figure. 1 Inverted edifice: use case vs feature based approach.

Right hand side of Figure 1 represents the Conventional mode while the left side represent the proposed methodology. As paper progresses, it would be more relevant and clear how can we have ease of validation and debug addressed by new use case based approach.

III. TEST AND DEBUG TRIGGERED VIA USE CASE

Having understood the use case approach, we need to check the details on the usage of the same with respect to the validation and debug on real world platform. The following section takes a look at both the validation and debug of platform spanning out with the use case based approach from an idealist system standpoint. In-order to understand about applicability of use case approach for debug and validation strategies, we need to first observe on type of abstraction layers we have and then pertaining to abstraction layers, what kind of validation problems and debugging complexities can precede. Identification of problem, with anticipated complexities & trying to address it with proposed approach would help, in reduction of both the validation as well as debugging related issues. This is seen in the following section with hibernate system state example.

Taking into account the "outside in approach" we normally see what a silicon (CPU) offers, after that we design the features, as well as other supported customaries. When we have the requisite hardware, i.e., silicon and board, it comes to the BIOS, Firmware's & device drivers. When all these things are good, we then move towards the choice of "OS" and the subsequent test requirements that are needed in-order to perform our validation. From the representation in Figure 2, it becomes much evident on what complex level the System validation happens.

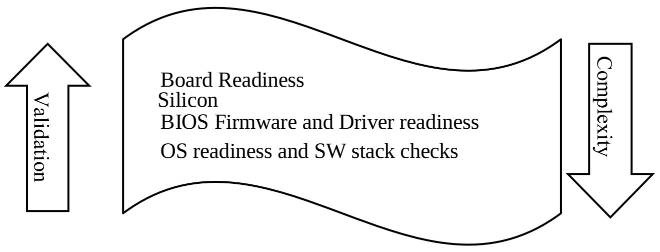

Validation
Board Readiness
Silicon
BIOS Firmware and Driver readiness
OS readiness and SW stack checks
Complexity

Figure. 2 Silicon to Software features depiction and traditional testing methods

Validating a scenario with proper use case defined becomes easier to test & articulate. Let us take a look at small example to explain how a use case definition can help to ease the complexity of validation. Goal is to validate system states that a system under test supports, and see what the test coverage is attached to the particular use case, map it back and get information about supported power states. Using various tools we can get re-confirmation that indeed these are the power features that we are expecting on the system. Once the existing use case is available amalgamating any new power feature onto the system power state matrix, becomes quite a simple task. Figure 3 provides an insight into one of the systems and the various power states that it supports in particular. This is a toned down version of multiple states via which the system can navigate through in different phases of validation and actual usage.

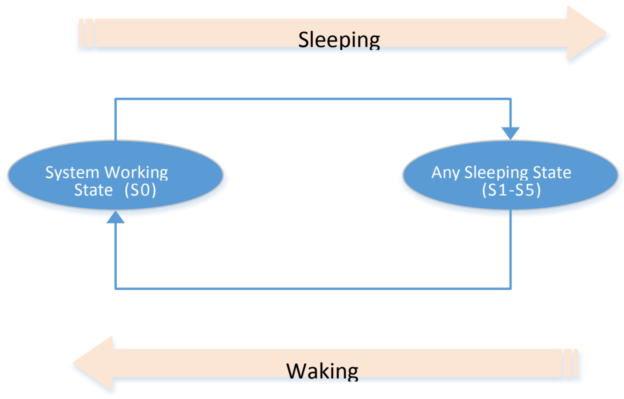
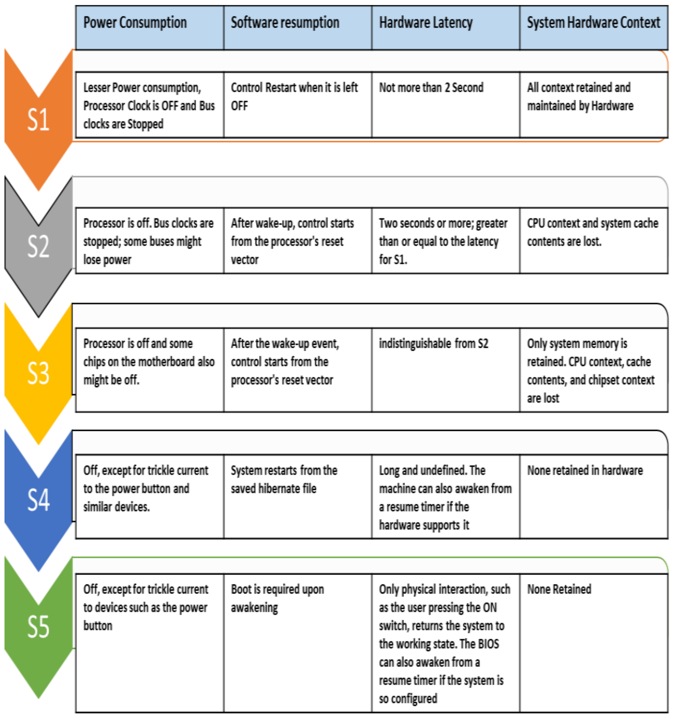


Figure. 3 System and corresponding supported and un-supported power states.

In addition to the validation strategies of available power states & checking on the coverage gap of existing power states or completely abstaining power state, we can also look at the use case definition as one of the major pillars for debug.

Consider one of the debugging scenarios here with platform power policy and the way failure condition is debugged. We expect S4 (Hibernate) state as a default platform state present, but for some reason we have the S4 (Hibernate) not mentioned in the available states list. Then, from the use case lists we can take starting point as absence of S4 (Hibernate) state & what are the platform use cases that would be hindered and in-turn what feature lists cannot be tested. Then from Figure 2. Silicon to Software depiction, we can check that what can be the probable cause of system state failure, whether it maps to silicon abstraction layer or due to any other failure. Once the leads are generated the debug process direction is decided accordingly. Say we have issue with the OS then follow up software debugging is done and subsequent resolutions would get us the issues sorted out.



	Power Consumption	Software resumption	Hardware Latency	System Hardware Context
S1	Lesser Power consumption, Processor Clock is OFF and Bus clocks are Stopped	Control Restart when it is left OFF	Not more than 2 Second	All context retained and maintained by Hardware
S2	Processor is off. Bus clocks are stopped; some buses might lose power	After wake-up, control starts from the processor's reset vector	Two seconds or more; greater than or equal to the latency for S1.	CPU context and system cache contents are lost.
S3	Processor is off and some chips on the motherboard also might be off.	After the wake-up event, control starts from the processor's reset vector	indistinguishable from S2	Only system memory is retained. CPU context, cache contents, and chipset context are lost
S4	Off, except for trickle current to the power button and similar devices.	System restarts from the saved hibernate file	Long and undefined. The machine can also awaken from a resume timer if the hardware supports it	None retained in hardware
S5	Off, except for trickle current to devices such as the power button	Boot is required upon awakening	Only physical interaction, such as the user pressing the ON switch, returns the system to the working state. The BIOS can also awaken from a resume timer if the system is so configured	None Retained

Figure. 4 System power state details

IV. VALID USE CASE ANALYSIS

This section demonstrates the usage of this model to get more robust understanding of use case application. For explanation purpose, system state Hibernate a.k.a. S4 is considered. Validation plans & tests are derived from use case scenarios. Various Combination and tests are planned with S4 entry/exit criteria kept in mind. Coverage is quantified with features planned and validation matrix created subsequently for an end to end use case and flow.

Basic understanding of Hibernate use case, provides information such as the condition of system, power consumption during hibernate, input wake mechanism, how system should behave after wake and what to restore after hibernate exit. Additionally various user scenarios which include multi domain interactions are also covered. Table I enlists some of the features/test that needs to be checked and covered during Hibernate use case validation.

TABLE I. FEATURE/TEST CASE CHECKS DURING HIBERNATE USE CASE

SI No	Condition in Hibernate	
	Scenario	When
1	System in off condition	Yes during entry
2	Power consumption status	While in S4 system should measure the lowest power
3	Wake scenario	Wake using USB, LAN or any other input source based on the system feature.
4	Context Saving	While entry, hiber file should be generated with all the active context stored and on exit it should retrieve all the context from the Hiberfile.sys
5	Battery Management	System should trigger Hibernate based on the amount of time the system is in idle.
6	Responsiveness	Involves Time taken for entry and exit for hibernation
7	Memory Management	Check System memory Decomposition when resumed from Hibernation
8	Video/Audio Resume after Hibernation	Context shouldn't be lost and user should be able to resume the MM content
9	Input Sequence	What type of input sequence need to be planned for entry such as power button, via OS , using scripts etc.

Understanding few of the scenarios would help in gauging usage of this model on real system cases. Gradually starting with the functional test then moving to inter operational tests, stress test and finally to the reliability checkouts is the methodology of this use case model.

In every stage of checkouts, various tests are performed and result is measured against expected outcome. Starting with functional checks where the basic entry/exit of hibernation is tested and expectation is to have all the precondition met. Entry to S4 when initiated, triggers following processes all apps drivers and services are notified, all the system context is saved on the boot media. Resumption from S4 is determined by OS boot manager by detecting a valid hibernation file, after that it directs the system to resume, restoring the contents of memory and all architectural registers, the contents of the system memory are read back in from the disk, decompressed, and restored back to its original state [4].

After functional tests, few inter-operability scenarios are run to make sure system memory is checked properly with all active context saved and resumed. The example scenarios for system memory check would be video player run resuming from where it was paused or YouTube streaming window reloaded and paused, etc. Consecutive Hibernate cycles, with counts gradually increasing from 100, 500 to 1000 are checked. This provides the overall stability and confidence on the system reliability. Additionally inclusion of traffic along with hibernate cycle is done where, scenarios such as Bluetooth file transfer and S4 cycles simultaneously, are run for multiple iterations.

TABLE II DIFFERENT HIBERNATION FILE BASED ON FAST STARTUP OPTION

Hibernation File Type	Default Size	Supports
FULL	40% of physical memory	Hibernate, Hybrid Sleep, fast Startup
REDUCED	20% of physical memory	Fast Startup

V. ISSUE DEBUG ON HIBERNATION

Debugging any issue from platform remains challenging and most troublesome due to the sheer number of variables that can affect the flow. It becomes more tedious and cumbersome process if any power flow is involved as there are numerous transitions which increases failure probability & debug complexities. Of the all power state flow Debug of Hibernation use case remains is one of the toughest.

For any power flow it has two contexts, 1st being entry and 2nd being exit or resume from concerned power state. Issues mostly prevail among these two context. The issues seen can be categorized into following failure buckets.

1. Context not getting saved after resuming from S4
    a. The system is not taking the S4 flow and entering into other alternative Power flow path such as S3 or S5.

2. Devices not recognized after resuming from S4
    a. Multiple devices gets lost or doesn’t detect after resume such as storage devices like USB, Yellow bangs to various modules such as Connectivity modules, IO or any controllers.

3. Soft Hangs
    a. Recoverable Hangs which can be due to issues in device driver loading after resume.
    b. Unrecoverable Hangs or device lost while resuming resulting in Memory dump such as Blue Screen of Death (BSOD), Green Screen of Death (GSOD).

4. Hard Hangs
    a. Non Recoverable error or hang observed resulting in system not responding towards any of the user commands
    b. These issues can be due to IP hangs or Silicon hangs. Debugging done via Joint Test Action Group (JTAG).

5. Responsiveness
    a. Time taken to enter the Hibernate is more compare to the Target specified.
    b. Time taken to resume from Hibernate fails to meet the target specified.

6. Auto Wake Issues
    a. Systems wake as soon as it enters S4.
    b. System wakes from S4 after certain duration.

All failure needs a different debug approach, in order to achieve the best results. Each failure above needs dedicated effort and support to root cause and narrow down upon the exact problem causing component. At high level we can understand the debug strategy using following flow chart given in Figure 5 and aligning it with the use case based model gives us the flexibility of getting things done at much faster, organized and streamlined way. We start the debug to check if it is a hard hang and if it is the case we need to use hardware mechanisms and tool like JTAG to scan inside the silicon using its Design for Test (DFT) capabilities. If there are only soft hung seen, then we need to get it bucketed in sub-category and pursue a different method of debug regarding the same. This flow chart explains at a very high level of abstraction of a well knitted and branched out debug tree. With more and more defects the tree would fan out to utmost complexities and that is where the use case scenarios will come in handy to identify the feature dependencies and debug them as applicable.

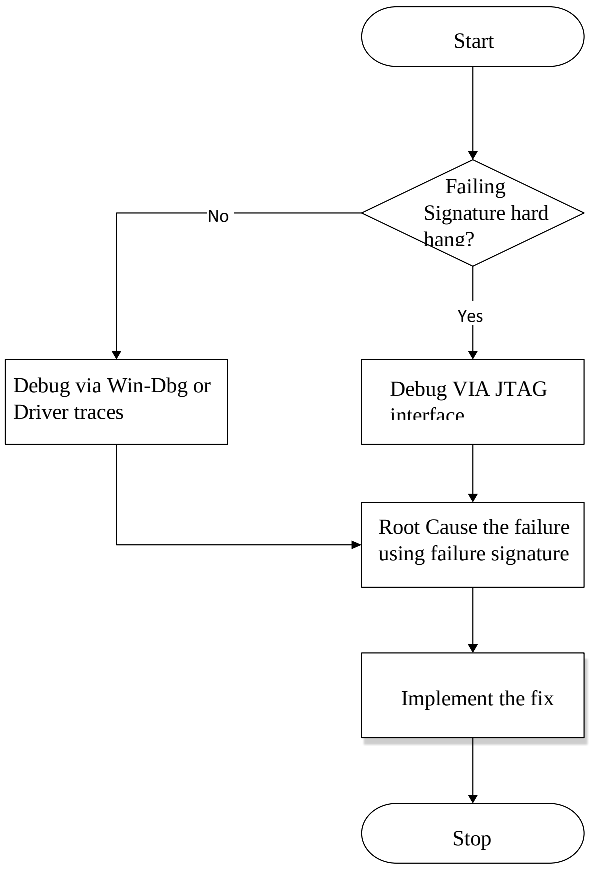

Start
Failing Signature hard hang?
No
Yes
Debug via Win-Dbg or Driver traces
Debug VIA JTAG interface
Root Cause the failure using failure signature
Implement the fix
Stop

Figure. 5 Debug flow for Hibernation issues

For any platform, the flow defined can help and pin point what are the use cases impacted in power management domain and also the effect of same on the corresponding overlapping domains. Issue debug with the flow chart provided helps in promptly resolving and nailing the issue. This would get us also the details on the overall system coverage. Also with the methodology that we are following, it becomes easier for us to check for the hard hangs or soft hang and also follow the proper bucketing procedure as stated earlier.

Let us take a peculiar example of failure and then map it back in out flow chart and then subsequently take it further down to the abstraction level where we see the failure being pin pointed. After that we would also check for the interpretations from the coverage viewpoint what can be removed. As per the description from section IV the major thing that we have failures with hibernate flow is entry and exit from it.

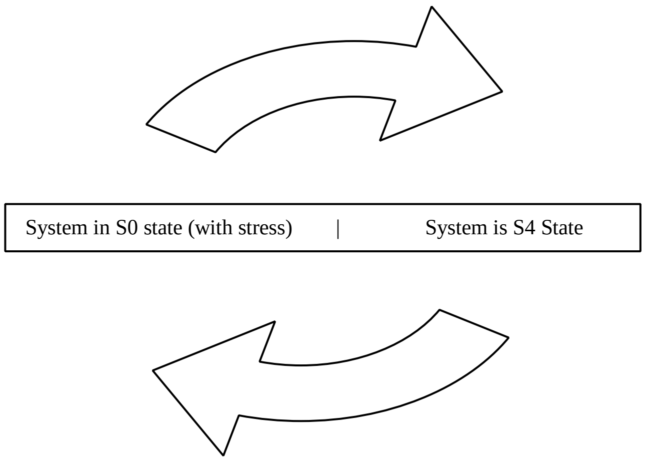


Figure 6. Representation of iterative stress validation scenario

We have a stress testing scenario as shown in Figure. 6 where there are back to back S4 cycles needed for qualification of platform release. During overnight stress run we see the failure, depicting a display off symptom, while system has power and other peripherals are properly in place. Then, we need to start debugging from the point where we need to identify whether the failure was while resuming or entering to the S4 state. After initial level of analysis as per the debugging flow say we zero down to the conditions saying it is soft hang with indication that while entering to S4, system went into corrupt state. After some more analysis we get to know that because of an issue with inbuilt OS drivers we are having a suspected failure.

Inferring from the above information at hand we can tell that we do not have issues with the hardware per say, be it silicon or board specific or any other third party hardware IP. We could also say that there are bunch of probable causes from the software side when we are in process of debugging. Deeper dive in the debug can then in-turn reveal, what is the exact component / entity failing and would help in getting what features are blocked owing to this failure. Once that information is available, we could then get a reduced platform test coverage as the tests involving resuming from S4 in any way would all get blocked. To quantify it we can explicitly say approximately a test plan would see reduction

For better understanding the Figure 7 depicts the pictorial representation of the use case and also gives an idea about inferences at various levels.

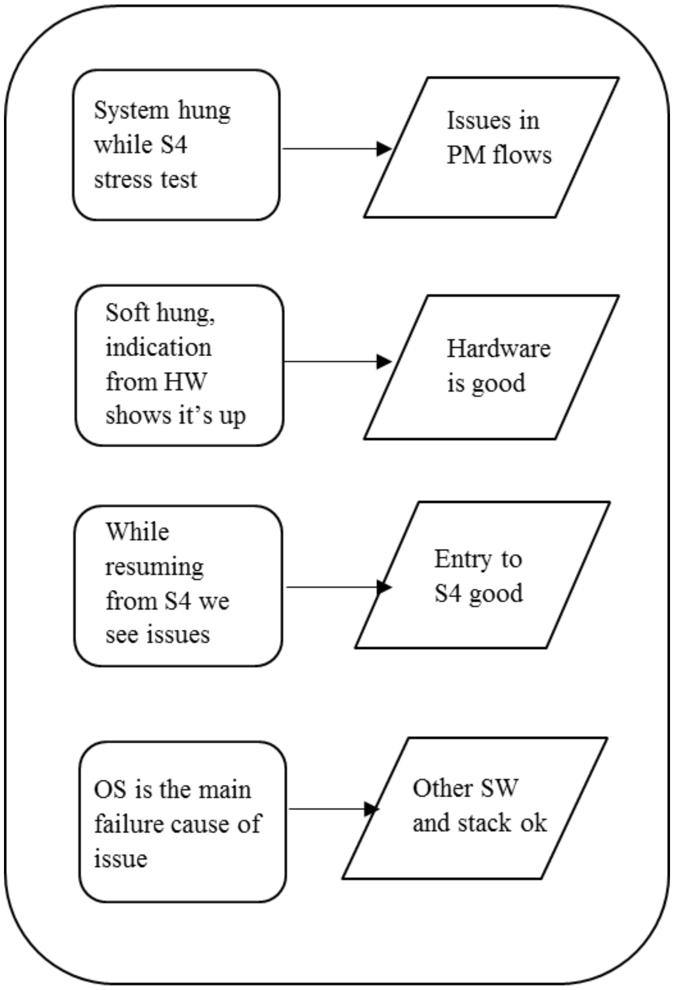

System hung while S4 stress test
Issues in PM flows
Soft hung, indication from HW shows it's up
Hardware is good
While resuming from S4 we see issues
Entry to S4 good
OS is the main failure cause of issue
Other SW and stack ok

Figure 7. Observations and inference matrix generated from S4 fail analysis

From a single issue we could get many inferences and also a fair bit of idea as to what would be main features blocked. In similar fashion we would be having information from all issues coming in and giving the validation teams a clear picture of what is status of platform health, where there are more bugs and more focus is needed and also we could get information on redundant tests that are not yielding bugs. Basically dynamically changing the test plan. So all in all this would emphasize and enhance the test plan quality and also the way system validation is done providing all the necessary aids and opportunities of improvement.

VI. RESULTS & CONCLUSION

This paper discusses the methodology for alignment of system test content and gauging of the details of system coverage, how issues can be debugged efficiently and effectively. The overall essence of paper is to move from the feature centric validation and debug to the use case based approach & intuitive debug of the issues targeting platform agnosticism. The use case approach eventually helps in the easier feature test additions and also the validation at system level. Debugging and error categorization also becomes way easier if we follow this methodology. The proposed methodology can be extended to any of the systems use case wherein we can perform the respective scaling of test plans and other features checks depending on the user scenarios. A running use case example and the debug fan outs for the erroneous conditions are also presented as part of the paper.

As part of deployment of this methodology internally we have used the same approach for the previous two platforms for validation and have seen 20% reduction in validation cycle times overall. If we translate it to direct $$ savings it would be around the 20% budget saving given for the platform validation. This when clubbed with various OS where individual platform validation cycle is performed amounts for a considerable amount of money. Additionally taking this methodology and furthermore AI based algorithms we have developed tools internally which takes the platform defects as inputs and provides us with the requisite test plan generated dynamically, which is a reduced set list depending on the defect trends seen in the earlier runs.

REFERENCES

[1] N. Kumar, "IoT architecture and system design for healthcare systems," *2017 International Conference on Smart Technologies for Smart Nation (SmartTechCon)*, Bengaluru, India, 2017.

[2] A. V. Ramesh, S. M. Reddy and D. K. Fitzsimmons, "Airplane system design for reliability and quality," *2018 IEEE International Reliability Physics Symposium (IRPS)*, Burlingame, CA, USA, 2018.

[3] Advanced Configuration and Power Interface Specification, Version 5. November 2013.

[4] IEEE Standard for Test Access Port and Boundary-Scan Architecture - Redline," in IEEE Std 1149.1-2013 (Revision of IEEE Std 1149.1-2001) - Redline, vol., no., pp.1-899, May 13 2013.

[5] J. Ryser, M. Glint A Scenario-Based Approach to Validating and Testing Software Systems Using Statecharts

[6] I. Jacobson: Basic Use Case Modeling; Report on Object Analysis and Design, vol. 1, n° 2, pp. 15-19, 1994