Volume 4, Issue 10, October 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Verification and Validation of Embedded Systems - The Good, the Bad and the Ordinary

Adnan Shaout*
The University of Michigan – Dearborn
The Electrical and Computer Engineering Department
Dearborn, United States

Cassandra Dusute
The University of Michigan – Dearborn
The Electrical and Computer Engineering Department
Dearborn, United States

Abstract - Verification and validation are two major parts of a product's life cycle. Combined, they ensure that the product meets requirements and that the software and hardware are built correctly. For embedded applications, it is important that the systems pass verification and validation not only for quality purposes, but for safety as well. The objective for this paper review existing verification and validation methods, discover what works and what doesn't work and propose a new process for verification and validation for embedded systems using those working processes called VVResPCT.

Keywords - real-time embedded systems, verification, validation, GM ignition switch, software, hardware.

I. INTRODUCTION

An embedded system is a combination of hardware and software that is made to fulfill a dedicated function. For example, a car's airbags are controlled by an embedded system. Some embedded systems depend on real-time accuracy to operate correctly. It is important that an embedded system is built with quality in mind to prevent serious consequences [1].

How is quality defined? Quality can be broken down into many parts. Is the system reliable? Is it safe and functional? Is it usable? These questions need to be addressed throughout a system's lifecycle. They all can be answered by applying validation and verification activities from the start.

What is validation and verification? The following can quickly sum up validation and verification:

- **Validation** - Am I building the right product?
- **Verification** - Am I building the product right?

Validation determines if the final software meets the needs and requirements of the user. Verification determines if the product has been built according to the design specifications and requirements after each phase of development. Validation is usually run at the end of development, however it is suggested that both validation and verification be done throughout development [5]. Since the cost of making a change increases the further into a project you are, detecting any discrepancies between requirements and the product early will lower this cost.

A company can use in-house teams to conduct validation and verification activities. However, it is suggested that an independent validation and verification team is used to run through a product. In some situations where safety is involved, companies are required to conduct certified tests on the product they are building. These teams, who are often certified, should be involved in the development early on, so that they are familiar with the system. The team will help remove bias from tests and the familiarity with the system will ensure that nothing goes untested, and the final product is of high quality [7].

Functionality and usability can be easily addressed with basic validation and verification activities. Safety and reliability, however, need more extensive testing.

A. Common misconceptions

During development, there are three fundamental concepts applied to achieve successful software. These three can often be confused for one another. Although they serve to achieve the same goal, they are all different and meaningful in their own way. The three concepts are:

- Verification
- Debugging
- Testing

As stated above, verification proves that a program meets a given requirement. This differs from debugging and testing slightly. Debugging looks into already known errors to determine the cause of the error and how it can be fixed. Testing identifies these errors, looking at the differences between expected and actual results. While both verification and validation are often mislabeled as testing, it is important to remember that testing is a part of validation and verification, not the other way around.

Testing is a dynamic activity. It can take real-world conditions and apply them to the system as a whole. Since it is dynamic, testing usually can't be used in early development. However, static testing is a good replacement. In fact, validation and verification activities start off static early in development, and become more dynamic as development progresses.

The paper is organized as follows: section 2 presents some of the current software processes used in industry, section 3 presents validation and verification processes used, section 4 presents a new lifecycle called VVResPCT, section 5 presents where did GM went wrong and section 6 presents concluding remarks.

II. CURRENT PROCESSES

Currently, there is one well-known lifecycle model that stresses the importance of both verification and validation during hardware and software development. During the V-model lifecycle, each development phase has a corresponding testing phase that is done in parallel [9]. Figure 1 shows an example of a V-Model Lifecycle.

The verification track, or development, is boiled down into four main phases:

- **Requirements** - important phase of the development track. All requirements are understood based on the customer's point of view. During this phase, the design for Acceptance tests is created.
- **System Design** - during this phase, the complete system should be designed. This includes hardware design as well as how components will communicate. During this phase, the design for System tests is created.
- **Architecture Design** - the system design from the previous phase is taken and broken down into more specific modules and is more clearly understood in this phase. During this phase, the design for Integration tests is created.
- **Module Design** - individual modules are designed, keeping in mind that they need to be compatible with other system modules. The architecture of the system is used to make sure that modules will work nicely together. During this phase, the design for Unit tests is created.

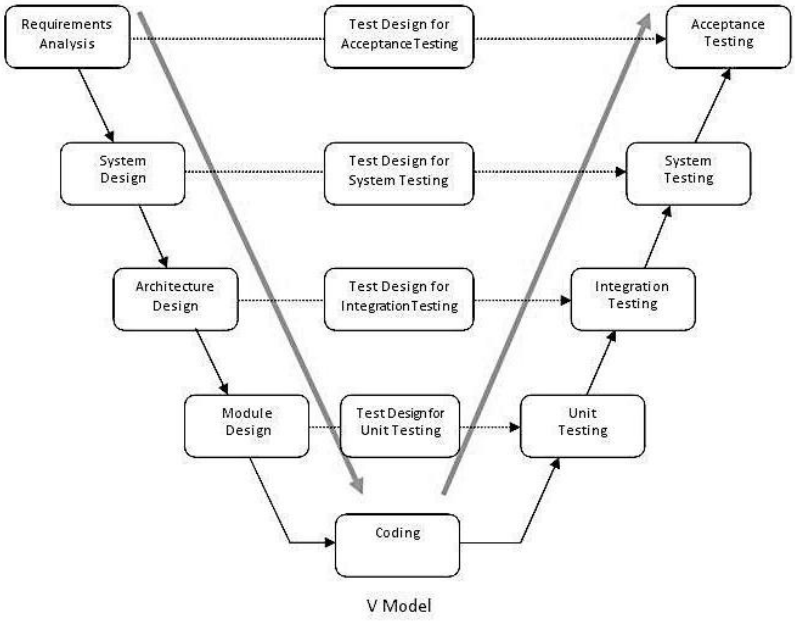

Figure 1. Example of a V-model Lifecycle.

The corresponding validation side includes these phases:

- **Unit Testing** - these tests are run at the code level. They help to prevent bugs in the future.
- **Integration Testing** - these tests are used to test how the system communicates with each internal module.
- **System Testing** - these tests check the compatibility between the hardware and software used. They are also used to see how the system works with external modules.
- **Acceptance Testing** - these are the tests performed by the customer to make sure the product meets all of their requirements. This is usually the phase that uncovers compatibility issues with the end-users environment.

The v-model lifecycle is good because it is easy to use and manage. However, the v-model does not work well with large projects that are ongoing or have changing requirements. Also, the v-model doesn't produce any working program until the end of the lifecycle.

If the V-model isn't the lifecycle currently being used, there is a validation and verification process that can be added to whatever testing phase that is currently being used by the company. That process includes six steps and changes depending on how far into development the system is.

- **Planning and control** – how will the validation and verification process (activities) be managed and performed.
- **Preparation** – what will be achieved by the activity and what techniques should be used for the activity?
- **Specification** – define activities (for example, what environment is the activity in, what test cases should the activity use?)

- **Implementation** – test data and tools for running activities.
- **Execution** – run activity.
- **Wrap-up** – evaluate any results you get after running the activity.

Implementation and execution will change depending on where the system is in development. If validation and verification activities are running during the beginning of development, implementation will be lacking test data.

III. VALIDATION AND VERIFICATION METHODS

In this section we will present a review of the validation and verification methods use for software systems.

A. Static vs. Dynamic

Validation and verification methods can be summarized into two main categories: static and dynamic. While both have their advantages, the best choice depends on what type of coverage needs to be added. Static methods, like model checking, check the requirements and design implementation for correctness without testing time constraints. While this is good for checking logic, these methods don't cover timing, which is crucial in real-time embedded systems. Here's where dynamic methods come in. Dynamic methods use time to model how a system will behave. A combination of both methods will provide full coverage for any system.

B. Static Analysis

This method, more formally known as Code review, is a static method that helps uncover missing, redundant, deficient or non-required functionality. The method analyzes the program without actually running it. When used early in development, it helps to ensure quality programming and reduces unnecessary bugs.

There are two forms of Static Analysis: walkthrough and inspection. During a walkthrough, the object execution is simulated; then, test data is walked through step by step to discover awkward algorithmic solutions. Inspection is a more technical code review. Static analysis is often applied to testing requirements and the design of a system.

C. Formal Methods

For more logic based validation and verification methods, the use of formal methods is suggested. Formal methods use math to demonstrate the correctness of a system [3]. The two main formal methods are model checking and theorem proving.

While these methods are very useful, they can be hard to use since they require an extensive knowledge of advanced mathematical techniques. However, as computer systems advance, both methods are becoming more and more popular within leading software companies. This is probably because they are becoming increasingly user-friendly and are some of the best when it comes to detecting logical errors.

D. Model Checking

In model checking, a model – for example, a prototype or finite state automata – is used to describe a working system. Then, a specific property of that system is turned into a logical function, and the model is explored to verify that property. Model checking is good for testing requirements, finding deadlock within the system, and discovering unreachable or orphaned modules.

Model checking usually consists of three main phases:

- Create a model of the system
- Provide a system property that needs to be proved
- Run a model checking tool to prove that property

Figure 2 shows a more detailed model checking workflow.

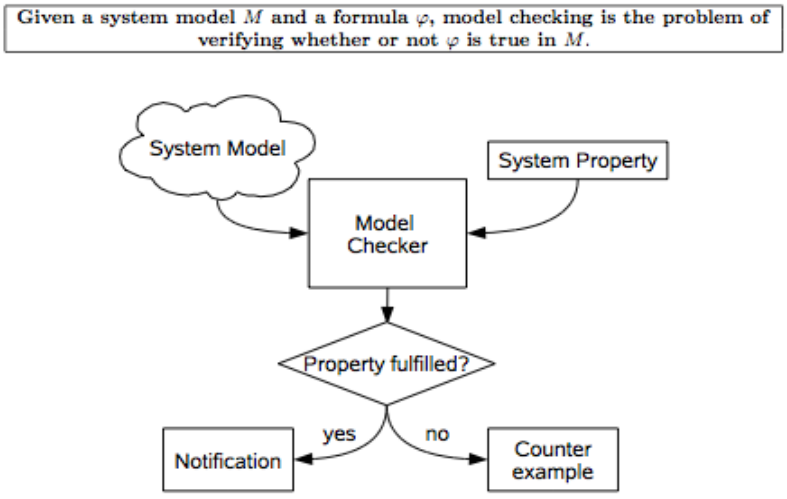

Figure 2. Model Checking workflow.

Although model checking is the easier of the two formal methods, it does have a dark side. In large systems, model checking faces the danger of state-space explosion, where there are too many states that are rapidly forming. This prevents the user from seeing every possible interaction between states. With advances in technology, however, the model checking method is capable of checking many more states than it has in previous years.

Model checking is sometimes used with Kripke Structures to represent reachable states and transitions in a system. An example of a Kripke Structure is the ever-popular coffee-vending machine. Figures 3 and 4 show the coffee vending machine example.

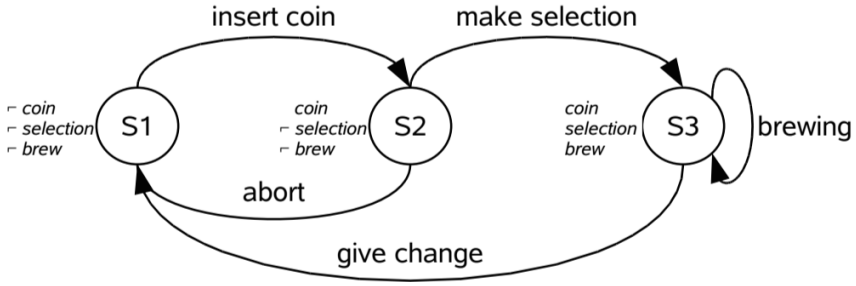

Figure 3. Coffee Vending Machine Example as a Kripke Structure [4].

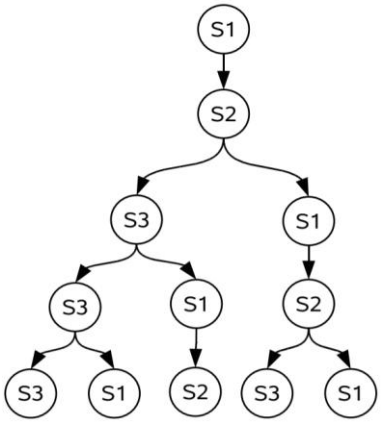

Figure 4. First Few Reachable States of Coffee Vending Machine.

As you can see in Figure 4, the number of states can quickly become overwhelming. In fact, this diagram can continue for infinite states. Since it is repetitive, this example is excluded from state-space explosion, as we know what types of interactions are possible.

E. Theorem Proving

Developed in the 70's by Tony Hoare and Edsger Dijkstra, theorem proving proves the correctness of a system by meeting a theorem [4]. Theorem proving uses a precondition and executable code to find a post condition. The theorem states that if a precondition is true before the program is executed, then the post condition must also be true.

Since theorem proving uses such extreme math and logic, it is rarely used as a verification method. It is also a time consuming method, because the user has to guide the process along. For example, the user has to continuously think about what proof should apply to the given step, and then wait for the result.

F. Simulation

Dynamic methods begin with simulation. Simulations accurately depict the software, and allow a user to test that the software is correct. They can be used on large models without worrying about state-space explosion. Simulations are one of the easier methods to use and one of the most popular; however they don't offer as robust coverage as other methods. For example, validating a simulation is a manual process instead of automatic. Also, simulations are capable of exposing erroneous behavior, but there's no guarantee that it will point out bad behavior.

G. Prototyping and Emulation

Prototyping and emulation take simulations to the next level. They are a close representation of the actual program, and provide close to accurate results. They are executed in real-life speeds and are put into real-life environments and situations.

Prototyping and emulation do have some disadvantages. They are time consuming to make, and may be expensive since they need to match closely with the actual system. Also, they can't be created until the design of the original system is done, putting off important testing. These disadvantages can be countered, though. Using cheaper hardware will make prototyping less expensive, and waiting until the design is complete will ensure that the prototype matches the system. Since dynamic methods aren't executed until later in development anyway, the timing of creating the prototype isn't a great deterrent.

H. Integration Testing

While integration testing isn't usually classified as a verification or validation method, it's still an important part of testing an embedded system. For this reason, it is listed as a current process. Integration testing is important because it not only makes sure that hardware and software work together, it also makes sure that the system as a whole is integrated properly. It also exposes many different types of errors. For example, integration testing can detect inadequate timing, which is a huge factor for some embedded systems. It can also detect when interrupt handling is being done wrong, which may also affect some embedded systems.

IV. PROPOSED LIFECYCLE

A new lifecycle, called Validation Verification Research Prototyping Coding and Test (VVResPCT) software process method, which involves more robust validation and verification coverage, is proposed in this paper. Figure 5 shows how that new process is separated into different phases. To further the new process, we integrated it into our own personal process, ResPCT [10].

A. Embedded System Life cycle

The ResPCT life cycle [10] consists of four main phases: research, prototype, code, and test. We have broken down the new process for embedded systems and combined it with the ResPCT process. The new process has been broken down as follows:

1. *Research*
    - Define Product Requirements
        - understand customer requirements
        - create test plan and test cases
    - create Validation and Verification activity plan

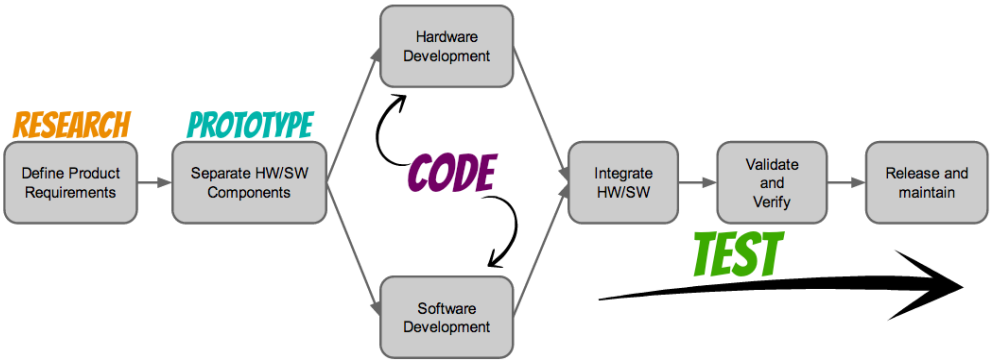

Figure 5. New Validation and Verification Process Integrated with ResPCT (VVResPCT).

2. *Prototype*
    - Separate HW/SW Components
        - design the system architecture including hardware and software components
        - distinguish what needs to be done for the next phase
3. *Code*
    - Software Development
        - see Figure 7.
    - Hardware Development
        - see Figure 6.
4. *Test*
    - Integrate Hardware and Software
        - run integration tests
    - Validate and Verify
        - prepare techniques - Model-checking and prototyping
        - gather test data
        - run activities
        - analyze
    - Release and Maintain
        - user acceptance testing

After reviewing current validation and verification methods, a method was chosen from each dynamic and static activity that proves to work the best. Those are model checking and prototyping methods. Both static and dynamic methods were chosen because of the way that validation and verification migrate between the two. At the beginning of development, a more static approach (model-checking) should be used to keep development on course and prevent any unnecessary bugs. As development continues, prototyping will help in making sure that the system is accurate and error free.

B. Hardware Development Life Cycle

As you can see in Figure 6, the hardware development life cycle adds more steps to the coding phase of ResPCT [10]. In fact, you'll find that both the hardware and software life cycles use their own ResPCT process to add more coverage as follows:

1. *Research*
    a. review requirements
    b. if contracting out, determine manufacturer
2. *Prototype*
3. *Code*
    a. build product to specs
4. *Test*
    a. certified testing
    b. HW/SW integration

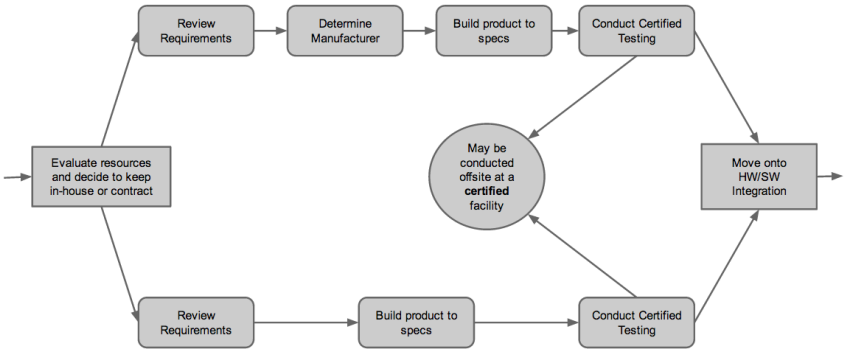

Figure 6. Hardware Development Life Cycle.

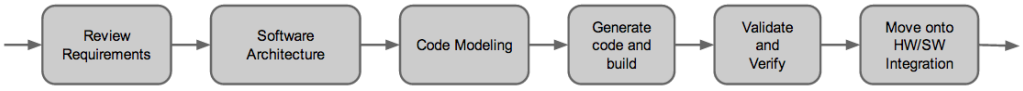

Figure 7. Software Development Life Cycle.

C. Software Development Life Cycle

Similar to the hardware life cycle, the software life cycle uses its own iteration of ResPCT. Here's how it is broken down:

1. *Research*
    a. review requirements
2. *Prototype*
    a. software architecture
    b. code modeling
3. *Code*
    a. generate code
4. *Test*
    a. validate and verify (run same process as above).
    b. HW/SW integration

V. WHERE GM WENT WRONG

The recent GM ignition switch recalls was researched in this paper. More specifically, we wanted to look into what their validation and verification plan was for the ignition switch, as well as what their process for approval was. Since the ignition switch was manufactured by Delphi, we will look into their approval process for the switch.

The ignition switch was originally built with the ease of use in mind. GM had received complaints about their ignition switches being too hard to turn, so for the newer early 2000 models, they decided to lessen the torque required to turn the key. They didn't realize that the decrease in torque required would increase the chances of the key turning into Accessory instead of run, thus turning off power to the airbags and power steering. Thirteen years - and 13 deaths - later, they finally recalled the faulty switch.

A. GM Validation Plan

GM's Validation Plan for the ignition switch includes three major parts: performance tests, validation tests, and component specifications. Figures 8, 9 and 10 will show each of these validation plans. As you can see in Figure 10, GM had specific requirements for the torque of the ignition switch. The torque - or how much force is required to turn the key in the ignition - required for the ignition switch should be between 15 and 25 N-cm (Newton centimeters). Based on the validation report in Figure 11, the ignition switch did not fall in that range. In fact, only a handful of switches made it above the minimum requirement.

General Motors Corporation North American Operations

4.0 VALIDATION

4.1 GENERAL

The Validation Plan consists of the test procedures required to satisfy component validation. The tests are to be performed by the supplier, unless otherwise indicated.

4.2 PERFORMANCE TESTS

The tests referred to in this section are quantitative or semi-quantitative in nature. Performance tests are to administered as specified in the validation matrix. Tests shall be performed as specified in Columns 1 and 2 of Table 4.2

STANDARD	SECTION	SPECIFICATION REQUIREMENT	TEST
		3.1.2.2	P0 - Simple Function Check[1]
GM 9110P	5.1	3.2.1.3	P1 - Voltage Drop/Circuit Resistance
GM 9110P	5.2	3.2.1.5	P2 - Open Circuit Resistance
GM 9110P	5.3	3.2.1.6	P3 - Isolation Resistance
GM 9110P	5.6	3.2.1	P4 - Function Check
GM 9110P	5.7, 5.8	3.2.2.3	P5 - Force Displacement
GM 9110P	5.12	3.2.1.8	P8 - Contact Bounce
GM 9110P	5.15	3.2.2.5	P9 - Rattle Evaluation
GM 9110P	5.11	3.2.2.4	P10 - Audible Sound
GM 9605P	2.0	3.2.1.4	P11 - Contact Resistance
GM 9110P	5.10	3.1.1	P12 - Appearance

TABLE 4.2

1. Quickly check the continuity or circuit resistance on a bench top while operating each control. The intent is to establish a minimum level of functionality between tests.

Figure 8. GM Ignition Switch Performance Tests [8].

General Motors Corporation North American Operations

4.3 VALIDATION TESTS

All the tests specified in this section and the following sections apply to all switch types. Each switch type shall be validated independently according to the test sequences and procedures specified.

GM 9110P SECTION	SPECIFICATION REQUIREMENT	TEST	COMMENTS
6.6		Visual (Teardown) Inspection	
9.1	3.1.5.1	Durability	
	3.1.3.1.1	Storage Temperature	
8.2	3.1.3.2	Humidity Test	
	3.1.3.2.1	Moisture Susceptibility	
7.7.4.1	3.1.3.3	Vibration	
8.5	3.1.3.4	Thermal Shock Test	
8.3	3.1.3.5	Corrosion (Salt Fog)	
8.4	3.1.3.6	Fluid Compatibility	
8.1	3.1.3.7	Dust Test	
GM6090M	3.1.3.8	Flammability Requirements	
	3.1.3.11	Ozone	
	3.1.3.13	Mixed Flowing Gas	
	3.1.3.14.1	High Temperature Endurance	Components with electronics only.
	3.1.3.14.2	Low Temperature Endurance	Components with electronics only.
	3.1.3.14.3	Power Temperature Cycling	Components with electronics only.
7.4	3.2.1.7	Continuous Current Overload	
	3.2.1.7.1	Short Circuit Test	
GM9100P Series	3.2.1.9	EMC	Components with electronics only.
7.2	3.2.1.10	Reverse Polarity	
7.3	3.2.1.11	24-Volt Jump Start	
9.10	3.2.1.12	1000 Hour Load Soak Test	
7.5	3.2.2.6	Mechanical Drop Test	
7.6	3.2.2.7	Mechanical Shock Test	
6.2	3.2.2.8	Terminal Retention	
6.3	3.2.2.9	Connector Insertion	
	3.2.2.10	Connector Retention	
6.4	3.2.2.11	Case Integrity	
	3.2.2.12	Switch Retention/Insertion	
6.5	3.2.2.13	Mechanical Overload	
GM9604P, Sect. 5.5	3.2.2.14	Switch Surface Temperature	
	3.2.2.15	Theft Deterrence	

Figure 9. GM Ignition Switch Validation Tests [8].

IGNITION SYSTEM TORQUE REQUIREMENTS

Actual curve to be furnished by supplier after GM Engineering approval.

Column Torque Requirement:
The maximum allowable torque of the Lock Housing/ Key Lock Cylinder interface (excluding the ignition switch) MUST not exceed 10 N-cm.

Ignition Switch Torque Requirement:
The minimum torque required by the switch, on the return side of the ignition switch from CRANK to the RUN position MUST be 15 N-cm.

NOTE:
Torque Curve allowable tolerance shall not exceed +/- 5 N - cm.

Figure 10. GM Ignition Switch Torque Requirements [8].

GM — ANALYSIS/DEVELOPMENT/VALIDATION PLAN & REPORT (ADVP&R) FOR SUPPLIERS

SECTION I – DESIGN EVALUATED

SECTION II – PLAN AND REPORT DATA

SECTION III – VALIDATION PLAN / SEC IV – VALIDATION REPORT

ITEM #	PROCEDURE #	PROCEDURE TITLE	REQUIREMENT # & TITLE	REQMT VALUE	RESPONSIBILITY	EVALUATION PHASE	SAMPLE QTY	SAMPLE TYPE	TIMING START	TIMING COMPL.	SAMPLES TESTED: QTY	SAMPLES TESTED: TYPE	SAMPLES TESTED: STAGE	RESULTS	NOTES
53	3.2.2.3	Torque-Angle	Actuation Torque	All In N-cm	Delph i	PV	6	D	1-7-02	1-8-02	6	D	Pil		See PPAP for Measured values.
			Off-ACC	25/15										Not OK	5/3 N-cm
			ACC-RUN	25/15										Not OK	7/4 N-cm
			Max Travel	65/55										Not OK	38/31 N-cm
			Return Torque	15 Min										Not OK	15/12 N-cm
			RUN-ACC	25/15										Not OK	5/4 N-cm
			ACC-OFF	25/15										Not OK	6/5 N-cm
			Displacement	All In Deg											Displacements-
			ACC Make	33/27										OK	33/31 Deg
			Key-In Break	40/34										Not OK	43/41 Deg
			RC Make	67/61										OK	67/65 Deg
			KRC Make	61/55										OK	60/58 Deg
			Diff KRC-RC	10/2										OK	9/7 Deg
			ACC Break	99/93										Not OK	102/100 Deg
			KRC Resist	95/82										Not OK	98/96 Deg
			Diff ACC-KRC	2 Min										OK	6/3 Deg
			KRC												

Figure 11. GM Ignition Switch Validation Report [2].

B. Process for Approving the Switch

When Delphi works with larger customers, like GM, they use their own process for approval known as Production Part Approval Process (PPAP) [6]. The steps for PPAP are as follows:

1. Purchaser (GM) provides the design and specifications for the part
2. The product is built to specifications and tested against those specs
3. The results of the testing are then given to the purchaser for final approval

During the PPAP, Delphi had noted multiple times that the ignition switch did not meet GM's minimum torque requirements. However, if the purchaser approves a product that has known errors, Delphi is obligated, per their contract, to deliver those products. Such was the case with the GM ignition switch. From GM's side, the approval process seemed to involve only one man, the lead engineer. That man, Ray DeGiorgio, approved the switch for production because he was not aware of any performance issues or effects on the safety of the car due to the underperforming switches.

C. Changes

While there are many ways a recall could've been prevented, we've narrowed it down to the three easiest and most obvious solutions. First, the faulty ignition switch should have been fixed early in development so that the torque fell within the acceptable range. Second, GM should have performed better test coverage on how other components within the cars reacted when the ignition switch changed states. Finally, the approval process for the switch should include more than just Delphi's testing results and a one-person sign-off.

The trouble with using an outside manufacture to create products is that they often don't have complete access to see how their product will integrate with other components. It is up to the purchaser to perform integration tests on all modules. For example, if GM had thoroughly tested the ignition switch against other components, they would have known earlier that, the ease of turning the key might be cause for concern.

VI. CONCLUSION

In many situations, verifying and validating a product could mean the world. In the case of a real-time embedded system, it could mean someone's life. Throughout this paper, validation and verification have been defined, current methods and processes for each validation and verification have been discussed, and a new process, called VVResPCT, using methods that seem to work best has been proposed. GM's ignition switch was also discussed, including their approval process and where the product engineers went wrong.

REFERENCES

[1] Laplante, P. A., Ovaska S. J. (2012). Real-time Systems Design and Analysis. Hoboken, NJ: John Wiley & Sons, Inc.

[2] Svoboda, T., General Motors. (2000). Analysis/Development/Validation Plan & Report (ADVP & R) for Suppliers: Delta Z Ignition Switch. http://democrats.energycommerce.house.gov/sites/default/files/documents/GM-Analysis-Validation-Report-Torque-2002-5-21.pdf

[3] Geilen, M. C. W. (2002). Formal Techniques for Verification of Complex Real-Time Systems. Eindhoven: Eindhoven University of Technology.

[4] Reinbacher, T. (2008). Introduction to Embedded Software Verification. Vienna: University of Applied Sciences Technikum Wien.

[5] Herrman, J. (2001). Guideline for Validation & Verification Real-Time Embedded Software Systems. http://www.dess-itea.org/deliverables/ITEA-DESS-D162-V01P.pdf

[6] Waxman, H. DeGette, D. Schakowsky, J. (2014). Democrats Request Details about GM Approval of Faulty Ignition Switches. http://democrats.energycommerce.house.gov/index.php?q=news/democrats-request-details-about-gm-approval-of-faulty-ignition-switches

[7] Aldec, Inc. (2010). Meeting Growing Verification Demands. http://www.aldec.com/

[8] DeGiorgio, Ray. (2001). Component Technical Specification. http://democrats.energycommerce.house.gov/sites/default/files/documents/GM-Component-Technical-Specification.pdf.

[9] Tutorials Point. (2014). SDLC V-Model. http://www.tutorialspoint.com/sdlc/sdlc_v_model.htm

[10] Adnan Shaout and Cassandra Dusute, (2013)," ResPCT – A new Software Engineering Method", International Journal of Application or Innovation in Engineering & Management (IJAIEM) 12/2013; Volume 2(Issue 12): Page 436 – 442, Impact Factor: 2.379.

