20XX-01-XXXX

Optimal test strategy for Clusters and CDCs in multi power train vehicle ecosystem

Author, co-author (Do NOT enter this information. It will be pulled from participant tab in MyTechZone)

Affiliation (Do NOT enter this information. It will be pulled from participant tab in MyTechZone)

Abstract

Innovation in energy storage and generation system will lead to multiple power train solutions across the vehicle categories in the Automative segment. With various options to the end consumer across different vehicle segments, the complexity associated with E/E Architecture and software engineering will be multi-fold both for the OEMs and Suppliers. Over the air updates shall become mandatory features to manage this complexity and to calibrate the vehicle features inline with changing trends and efficiency plus feature enhancements in post-market release scenarios. These upgrades are more common in digital clusters, in-vehicle entrainment and central digital cockpits. OEMs are introducing the vehicle platforms in multiple power train variants keeping comfort, instrument clusters and in-vehicle entertainment as core features across different power trains.

A well-defined and managed comprehensive optimal test strategy and infrastructure will be critical to ensure seamless release of the software solutions to different power trains during the product development phase and post-launch software upgrades.

In this novel work, we extensively explore the current practices of segregating features through common versus specific powertrains, managing the overall test strategy across varied test types, test data and test infrastructure; then address the advantages and challenges in our current practices. The paper would summarize the optimal test strategy in approaching the software development pipeline for a multi-powertrain architecture and focus specifically on early test-driven interventions for robust software deployment on production for both parallel and staggered release pipeline

Introduction

Multi-powertrain architectures are becoming more a norm these days, than they were a decade earlier. This, though brings in various options to the end consumer across different market segments, the complexity associated with engineering and especially Software engineering and its associated tests is ever growing. The scale and complexity associated with defining and managing a comprehensive test strategy, for a vehicle architecture across multi-powertrains is more profound in recent years

The thought flow in this paper would be first to introduce the various powertrains in automotive, the complexity of feature variations due to those powertrains, followed by overview of Software development lifecycle and its impact on products like instrument panel cluster and cockpit domain controllers, the current challenges in defining test strategy for such products and finally introduce the audience towards optimal test strategy during early development to production pipeline.

Automotive Powertrains

The performance of a vehicle heavily relies on the efficiency and condition of its powertrain. [4]. This system is central to vehicle performance, efficiency, and reliability, making it a core area of engineering focus for both development and testing in the automotive sector [2]. An automotive powertrain [1, 2] is the collective system of components in a vehicle that generates propulsion and delivers it to the wheels

**Components of powertrain**

The core components of powertrain are engine, transmission, driveshafts, differentials, final drive and control module. All these components can be selectively grouped into two major sets, namely engine [2] and drivetrain [2]. Engine is one crucial aspect, which converts the chemical energy from fuel / battery to mechanical energy to initiate motion [2] and drivetrains, which has the remaining components like driveshaft, transmission differential and axles, is just a way to describe all parts of the powertrain not related to the engine [1]. Each of the components is important, however lets briefly see the variations introduced by engine in specific and drivetrain. In the following section we would have a glance of those two at high level.

**Engines**: Engines are of one of the six types. They are

* Internal Combustion Engine (ICE) – Gasoline
* Internal Combustion Engine (ICE) – Diesel
* Hybrid Electric Vehicle (HEV)
* Plug-in Hybrid Electric Vehicle (PHEV)
* Battery Electric Vehicle (BEV)
* Fuel Cell Electric Vehicle (FCEV)

A good summary of their pros and cons are well described in multiple papers [5, 6], however for the context of out paper, we need to understand that many of those types are going to be part of vehicle architectures for few more years and as an industry we would evolve within those in improving the efficiency & regulatory needs, which would drive lot of software based features / functions.

**Drivetrains**:

Drivetrains are a sub-set of the whole powertrain, which helps the vehicle move. Drivetrains are one of the four following, which are

* front-wheel drive
* rear-wheel drive,
* all-wheel drive
* four-wheel drive.

Each of those are chosen for multitude of reasons by balancing some competing demands from, handling, fuel economy, traction control, road types, etc [1]. Example: For a normal city drive, where fuel economy, handling and sufficient traction is more important, front-wheel drive system is chosen. We are not going to dwell deeper in those aspects, except for mentioning that those types introduce different feature / functions at a Software level, to realize the benefit of those types.

**Feature variations for Cluster or CDC**

Each of the engine types, needs a list of supported features or functions to realize the end functionality of that type on the vehicle. If we were to group the engine types to three segments, ICE, Hybrid and Electric, we would notice that, for ICE, which is on one extreme we need a smaller battery, more in-car diagnostics for monitoring tail emissions and heavy transmission, however on the other extreme, a pure electric vehicle has a heavy battery pack, more in-car diagnostics for monitoring battery health and minimal transmission components. Most often everything that has to do with suspension, wheels, tires, steering, drive modes, lighting etc, is pretty similar between ICE and electric cars. However, there are areas, like powertrain connected components, along with safety, convenience, braking, where powertrain specific selection effects the features and functions within the vehicle. At high level the choice of engine & transmission, along with various components in the overall vehicle would define the software aspects of it. However not everything is different across those as well. The following section summarizes those portions of features, which are different and their impact on Software aspects on Instrument panel cluster and related products and those which are more of less similar across. To name a few:

* Features, which are same across powertrains. (Example: Odometer, Speedometer, Indicators, Drive modes etc)
* Features, which seemingly look similar, however the architecture & systems behind it are very different. (Example: Distance to Empty, Average Fuel Economy, etc)
* Features, which are new or closely coupled to powertrains. (Example: Acoustic Vehicle Alerting System (AVAS), Fuel leak detection, Auto-Battery cut-off, gear Indicator, Tachometer, Low battery warning, Regenerative Gauge, etc)

**Advancements in Powertrain Technologies**

Though we have seen the variations of features / functions across powertrains, this is not a status-quo, as innovation in automotive powertrains continues to accelerate through integration of software, electrification, and materials science. These advancements allow manufacturers to meet stricter efficiency standards and deliver new customer value [Opal]. These cab be broadly summarized into the following sections, as well explained in [Opal] and [[Link](https://www.mtwmag.com/innovations-in-powertrain-for-the-mobility-sector/#:~:text=One%20of%20the%20most%20significant,Combustion%20Engine%20(ICE)%20systems.)]

* Rise of electrification & Hybrid systems.
* Lightweight materials and modular design.
* Advancing combustion technology (ICE, ex: variable value timing, Turbocharged, VCI, HCCI, etc)
* Smart & Connected Powertrain.
* Advanced thermal management solutions. (Hybrid & Electric)
* Advanced EV Technology (ex: High efficiency motor, battery technology, two speed transmission, wireless charging, regenerative braking advancements)
* Integrated Control Systems: Use software to optimize torque delivery, shifting strategy, and energy recovery.

Each of these innovations improves the value proposition of modern vehicles, either through cost savings, performance enhancement, or reduced environmental impact. As we could see from above, lot of advancements have direct impact towards software part of the whole vehicle architecture.

Software Development Life cycle

The body of the paper should include detailed and structured description of the work performed, including (as appropriate) methodology, assumptions, hardware, observations, analysis, and a comparison of results with prior work. The information presented must be self-contained (in the sense that the reader is not assumed to have read prior papers) and provide an appropriate level of detail for the intended audience. Define all terms at first usage and apply them consistently.

The body section is not entitled “Body.” Rather it comprises multiple sections and subsections titled using topical headings in a four-level structure. Template styles [Head1] through [Head4] are used to tag and format titles of the different levels. No specific heading titles are mandated, but common examples include Methods, Results, and Discussion. The [Normal] style tag is used for paragraph text. Figures, tables, and equations fall under the body section. Found in this template are examples of a figure, a table, and equations (which must be kept to 3.5 inches wide).

Software Deployment

The history of software is a story of infrastructure and application code. Application code has changed in the sense of better frameworks and more developer-friendly tooling, but largely the process of writing code has remained the same. Infrastructure, on the other hand, has undergone some fairly large changes over time.[ <https://launchdarkly.com/blog/what-is-software-deployment/>]

Progresssion from VM’s to Containers to orchestration platforms, to application hosting platforms.

And there’s evidence to show that increased [deployment frequency](https://launchdarkly.com/blog/deployment-frequency/) leads to better organizational results. As illustrated in Google’s [State of DevOps Report](https://cloud.google.com/devops/state-of-devops), higher performing teams tend to deploy more frequently. Specifically, the highest-performing organizations deploy new code multiple times a day!

The goal of these [software deployment tools](https://launchdarkly.com/blog/software-development-tools/) is to automate the delivery of new code to different environments, e.g., development, [staging](https://launchdarkly.com/blog/what-is-a-staging-environment/), or production environments. Using infrastructure automation and CI/CD tooling, organizations can push new code into their version control systems, provision a new infrastructure environment, and deploy the changes to that environment with little to no developer or operator intervention.

Organizations have to make a decision on what their application architecture will look like ([microservices vs. monolithic](https://launchdarkly.com/blog/migrating-legacy-monolithic-applications-microservices/)), where their infrastructure will reside (on-premise vs. cloud), and a whole host of other details. Having a strong deployment strategy is the foundation of achieving that Continuous Delivery state because you need a way to create consistency across all teams within the organization

Deployment types:

<https://www.parasoft.com/blog/automotive-ci-cd-devops-test-automation/>

**How Containers Improve Automotive Software Development**

Development teams know that CI/CD workflows can function the same, regardless of whether they are containerized or not. But containers allow DevOps teams to deploy applications more easily, as well as patch and scale them to an organization’s needs. In essence, containers accelerate development, testing, and production as applied within Agile and DevOps utilization.

When it comes to managing complex development environments, especially in the safety-critical space, teams usually struggle with the following challenges.

* Synchronizing upgrades for the entire team to a new version of a tool like a compiler, build toolchain, and so on.
* Dynamically reacting to a new security patch for the library or software development kit (SDK).
* Assuring consistency of the toolchain for all team members and the automated infrastructure (CI/CD).
* Ability to version the development environment and restore it to service the older version of the product that was certified with the specific toolchain.
* Onboarding and setting up new developers.

All of these problems can be solved using containers.

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

Cost metrics On prime vs cloud : <https://bolt.earth/blog/cloud-based-development-in-automotive-software?srsltid=AfmBOopRChJq9rwYlaWHbm7GhTx4QiKwvc_FOYYzbkV-FV5y3rbFLWQH>

In 2022, the global automotive software market reached a value of [USD 21.7 billion](https://www.marketsandmarkets.com/Market-Reports/automotive-software-market-200707066.html) and is projected to reach USD 40.1 billion by 2027, with a CAGR of 13.1% during this period.

Vehicle software can be broadly categorized into:

* **In-vehicle software:** Directly impacts vehicle operations, including advanced driver-assistance system (ADAS), body controls, powertrains, infotainment, communication, remote monitoring, and [vehicle-to-everything (V2X)](https://5gaa.org/c-v2x-explained).
* [Industry software](https://www.matellio.com/blog/automotive-software-development-ultimate-guide/)**:** Applications that span the entire automotive value chain, including vehicle design and engineering, manufacturing execution, supply chain management, dealer management, and [fleet management](https://www.linkedin.com/pulse/navigating-electric-fleet-transition-boltearth) software and systems.

Embracing cloud technologies offers the automotive industry benefits, including cost-efficiency, increased reliability, faster time-to-market facility, and better scalability. In contrast, the drawbacks of an on-premise system entail substantial costs involving infrastructure, maintenance, storage, power, and licenses. Scaling poses complexities and expenses due to hardware prerequisites, and data backup necessitates investments in servers, network setup, and software, coupled with ongoing maintenance. These limitations underscore the constraints of on-premise systems in cost-effectiveness, scalability, and data management.

**Software System Complexities**

EVs require sophisticated software to coordinate their intricate systems seamlessly. Developing software for these complex systems poses challenges, mainly due to the need for domain knowledge and specialized expertise. Engineers must understand how components interact within the vehicle's ecosystem.

**Compatibility Issues**

A significant challenge in EV software development revolves around compatibility issues on two fronts:

* **Operational software compatibility:** EV software architecture comprises multiple layers, each interacting with distinct hardware components. Challenges arise when creating an operating system (OS) that accommodates various hardware, ensuring seamless interactions between layers.
* **Hardware compatibility:** EVs rely on software systems from varying manufacturers, demanding rigorous integration efforts. The challenge involves ensuring harmony and streamlined communication between software and hardware, with extensive testing necessary for reliable performance

**On-Premise Computing Limitations**

The predicted [dominance](https://www.digitaljournal.com/pr/news/xherald/electric-vehicle-software-market-size-growing-demand-growth-factors-opportunities-ongoing-trends-and-key-players-2023-2030-says-introspective-market-research) of cloud-based software from 2023 to 2030 raises questions about the challenges associated with on-premise computing — the practice of hosting software applications, data storage, and computing infrastructure within an EV manufacturer's physical premises. Several aspects of on-premise computing are currently limited. These include:

* **Performance optimization:** EV software optimizes complex vehicle aspects, requiring real-time data analysis and complex algorithms. On-premise computing constraints hinder efficient execution, affecting driving experience and energy efficiency.
* **Real-time processing:** Functions like regenerative braking need instant processing for safety. On-premise computing might struggle, risking delays and compromising EV safety and performance.
* **Over-the-air (OTA) updates and cybersecurity:** OTA updates enhance EV functionality. However, on-premise computing lacks the flexibility to manage updates for a diverse fleet, risking [cybersecurity](https://bolt.earth/blog/importance-of-cybersecurity-in-ev-ecosystem) and update deployment challenges.
* **Software updates and maintenance:** Routine updates are vital for bug fixes and security. On-premise systems may struggle to coordinate updates for a dispersed fleet, leaving vehicles outdated and vulnerable.

**Advanced Technological Innovations**

The world of cloud computing is constantly evolving, and the latest trends are shaping the future of this exciting field:

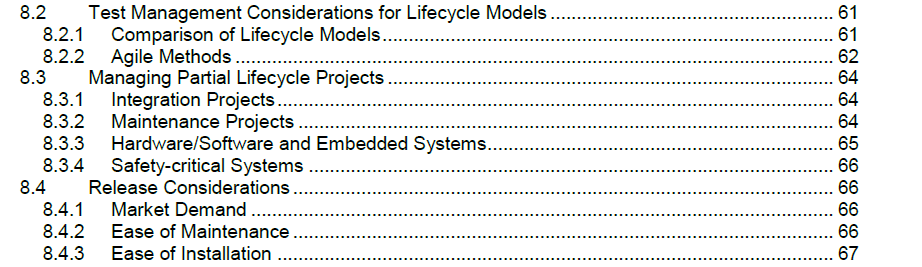
* **Multi-cloud and hybrid cloud solutions:** Empowers businesses to embrace diverse services for optimal results. Advantages include increased flexibility, improved disaster recovery options, and avoid vendor lock-in.
* **Edge computing and Internet of Things (IoT):** Allows swift data processing near the source and connected devices for data exchange, creating transformative potential across industries, fostering real-time decision-making, and elevating outcomes.
* **Artificial intelligence and machine learning in the cloud:** Enables efficient data management, informed decision-making, and personalized customer interactions. Their integration promises increased automation, data security, and individualized experiences, revolutionizing the cloud computing landscape.
* **Serverless technology:** Equips businesses to embrace cost-effective, scalable solutions without dedicated servers. It offers versatility, eradicating costs, and enables automatic scalability based on demand, reshaping cloud computing's landscape.

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

A [Head 2] style tag applied to the heading above is 11 point, initial caps, bold, italic. The [Normal] style tag should be used for paragraph text which is 9 point, regular.

Software Testing

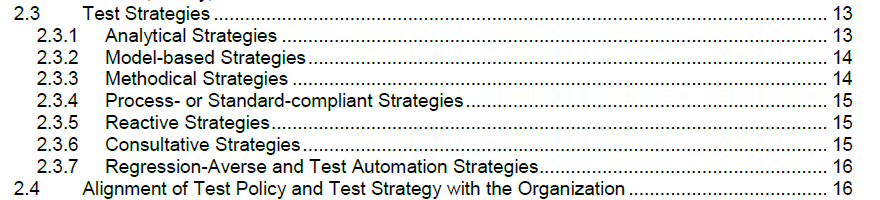
The history of software is a story of infrastructure and application code. Application code has changed in the sense of better frameworks and more developer-friendly tooling, but largely the process of writing code has remained the same. Infrastructure, on the other hand, has undergone some fairly large changes over time.[ <https://launchdarkly.com/blog/what-is-software-deployment/>]



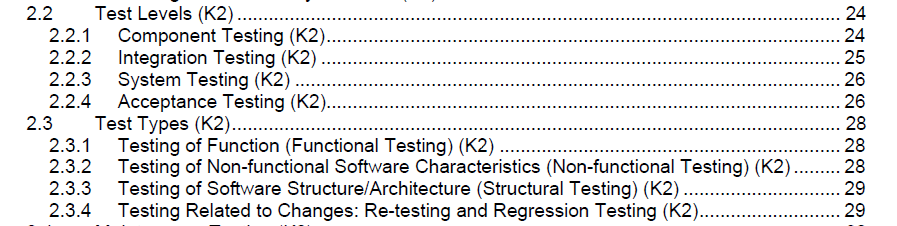
**Test Strategy**

A test strategy in software testing is a high-level document that outlines the overall approach and guiding principles for testing a software application. It provides a framework for planning and executing testing activities throughout the software development lifecycle. Essentially, it acts as a roadmap for the testing team, defining the objectives, scope, methods, and resources needed for effective testing.

An effective strategy for automotive software testing encompasses a **multifaceted approach that integrates various methodologies to ensure the safety, reliability, and performance of vehicles**.



**Test Type**

A software testing strategy outlines the approach to ensure a product meets quality standards. Key test types in a strategy include functional, performance, security, and usability testing, along with levels like unit, integration, system, and acceptance testing. These tests are crucial for verifying functionality, performance, security, and user experience, ensuring a robust and reliable software product. 

**Test Data**

Test data management (TDM) is the process for providing controlled data access to modern teams throughout the software development lifecycle.

Using data that faithfully simulates real-world circumstances is the main objective of TDM testing. Reliable and high-quality software releases are the result of regular and repeatable testing made possible by proper TDM.

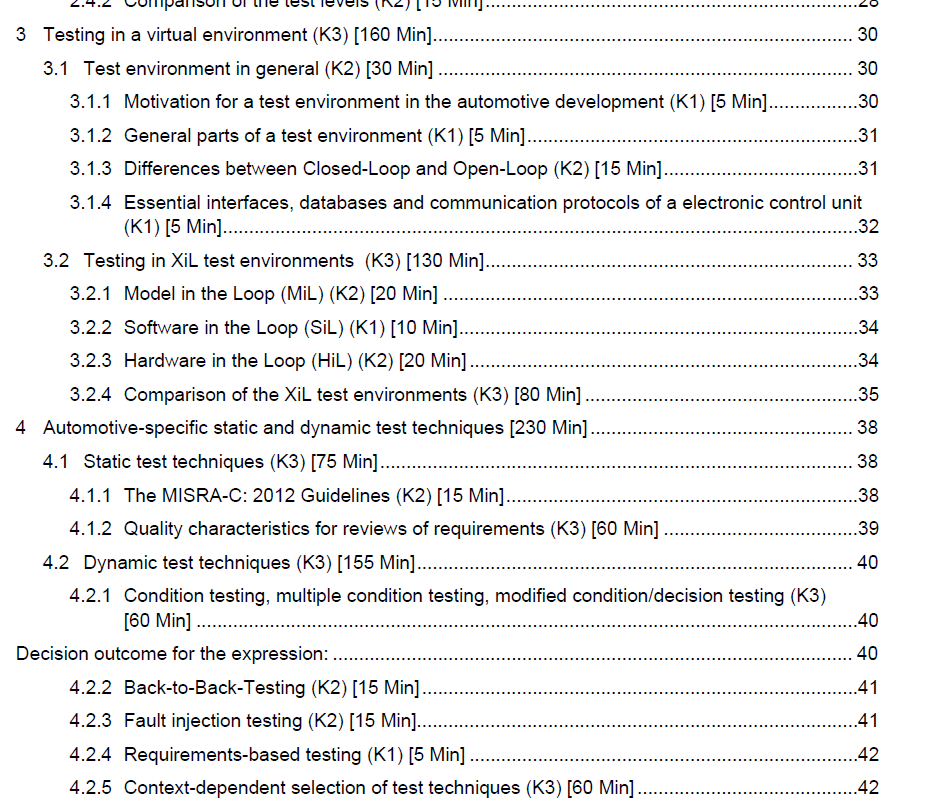
**Types of Test Data**

* **Live Data:** Real data from production environments, often used for manual software testing. While highly realistic, it poses significant privacy and security risks.
* **Synthetic Data:**Falsely generated data that mirrors real data without the related risks. This data is ideal for circumstances where real data cannot be used.
* **Masked Data:** Data that has been altered to protect sensitive information. This is often used when real data is necessary, but privacy concerns must be addressed.

**Test Infrastructure**

Test infrastructure encompasses the essential hardware, software, and resources needed to support the software testing process effectively. This includes test environments, tools, equipment, and test data, all crucial for planning, executing, and evaluating tests.

A well-designed test infrastructure plays a vital role in ensuring efficient and effective testing. It is typically established before testing begins and is often managed by a specialized team or individual. By optimizing testing resources and processes, test infrastructure enables faster test execution, shorter release cycles, and quicker time to market.



Challenges in managing test strategy in multi- Powertrains

The project test strategy guides all testing activities within a project and details objectives, resources, schedules, and responsibilities. This strategy must be tailored to the unique requirements of the project. Key decisions include the selection of test levels, test types, and test techniques for static and dynamic testing and other test practices (e.g., scripted testing, manual testing, back-to-back testing).

Choosing the best approach for a test strategy can be a complex process that can be influenced by organizational test strategy, project context, and other aspects. Selecting and combining test levels, test types and test techniques is therefore critical to an effective project testing strategy as it significantly influences the efficiency and effectiveness of testing.

Challenges based on the Product:

1. Derive the common test approach for Individual Customer, Software and system specifications with respect to power Trian
2. Maintaining Requirements traceability
3. Revisiting the testcases based on the change requests
4. Upgradation of generic framework based on the test approach for individual projects or complex requirements

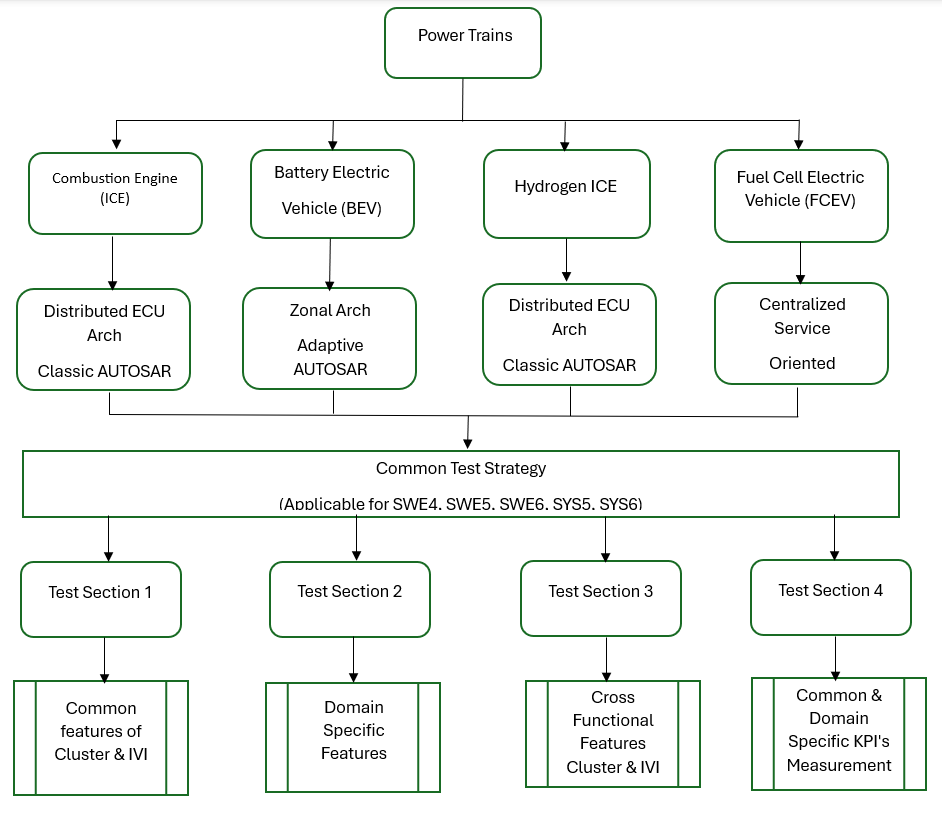
Challenges Based on the Project:

1. Common approach efforts should not impact individual project timelines
2. Strong Competent groups to be identified inside the project team members
3. Additional Efforts to have alignment with internal stakeholders and regular discussions

Proposed Optimal test strategy for multi- Powertrains

Aim to create the test strategy document which satisfies requirements of cockpit systems developed based on different types of powertrain vehicles, in this paper a well-defined test concepts and methodologies are mapped into traditional test strategy development stages and proposal of common test framework to achieve testability of multi powertrain architecture-based software implementations.

Key elements to define Test Strategy:

1. Master Test plan
2. Test strategy with Design
3. Software Architecture Differences
4. Test Approach
5. Developing Master Test Plan:
   1. Identify the Scope of the testing of Clusters and CDC’s with respect Multi Power terrain architecture
   2. Differentiate the common and Domain Specific Features to be tested
   3. Identify the common test environment across domains (ICE, BEV, Hybrid, Hydrogen, FCEV)
   4. Identify competent resources and Responsibilities
6. Common Test strategy design for Multi Power Terrain Architecture

Multi Powertrain Architectures:

1. Internal Combustion Engine (ICE)
   1. Distributed ECU Arch Classic AUTOSAR
2. Battery Electric Vehicle (BEV)
   1. Zonal Arch Adaptive AUTOSAR
3. Hydrogen ICE
   1. Distributed ECU Arch Classic AUTOSAR
4. Fuel Cell Electric Vehicle (FCEV)
   1. Centralized Service Oriented architecture

Common Test Strategy:

1. Test Section1:
   1. Group Common features of Cluster & IVI
2. Test Section2:
   1. Identification Domain Specific Features
3. Test Section3:
   1. Cross Functional Features Cluster & IVI
4. Test Section4:
   1. Common & Domain Specific KPI's Measurement

Software Architecture Changes:

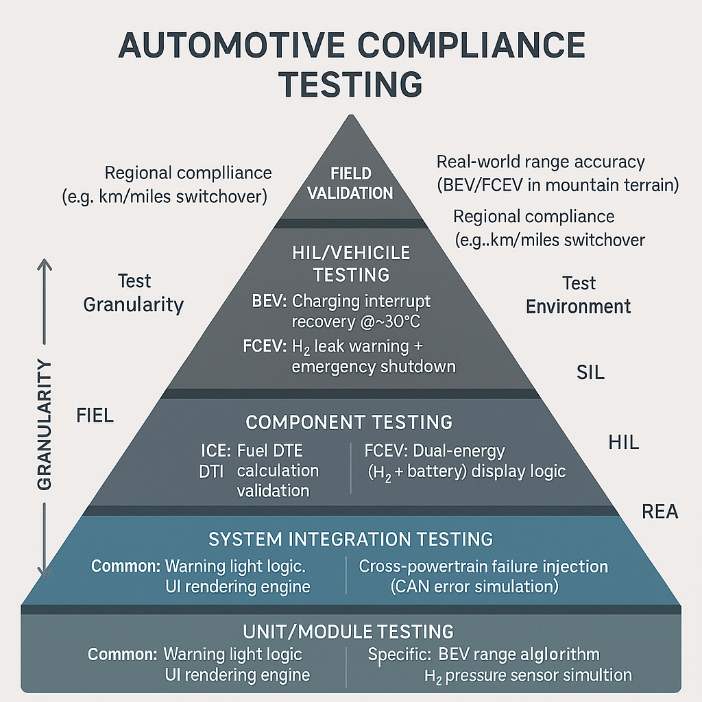
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Feature** | **ICE** | **BEV** | **H2 ICE** | **FCEV** |
| OTA Updates | Limited | Extensive | Emerging | Extensive |
| Middleware | Classic AUTOSAR | Adaptive AUTOSAR / POSIX | Classic AUTOSAR | Adaptive AUTOSAR / Linux |
| Cluster Type | Analog-Digital | Fully Digital | Hybrid | Fully Digital |
| Data Bus | CAN | CAN + Ethernet | CAN | Ethernet |
| Cloud Integration | Minimal | High | Low | High |

Choosing a Test Approach Based on Test Levels

An experience-based testing approach depends on the domain, knowledge, and past experiences of the testing team and their understanding of the system. The team employs a wide array of heuristics, domain-specific data, and testing techniques to anticipate any possible pitfalls, user interactions, or edge cases that might lead to failures.

The Test levels and Proposals towards multi powertrain architectures proposals mentioned below.

**Test Levels:**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Level** | **Focus Area** | **Tools Used** | **Example** | **Multi Train arch proposal** |
| Unit | Individual functions | gTest, CppUTest, pytest | Speed logic | Grouping of common logic function testcases and Domain specific logics function testcases |
| Integration | Module interactions | CANoe, Debuggers | GPIO Register level verification, Fault injection tests for UDS, | Maintain Test suits for Different architectures and reuse the |
| Software Qualification | Simulated Software input and output to cluster | CANoe, SIL benches | Diagnostics, Software update  Gear info, State of charge, HMI | 1.Segregate AutoSAR and HMI Features  2. Customize the test scripts  With General functions based on the power trains |
| System | Full cluster behavior | HIL, test benches, Labcar | HVAC , Engine Status check, Communication between ECU’s | High level Generic Test specifications to be deployed based on type of Vehicle level and labcar |
| Acceptance | User & business requirements | OEM tools, manual scripts | UX validation | 1.FOTA, Software update over UDS and USB common use cases to be defined and tested  2. Feature and Variant specific checklists can be maintained to identify the use cases |
| Regression | Stability after changes | CI/CD, automated suites | Re-run tests | Common automated framework with multi test sections can be proposed to reuse test scripts across multi power terrain architecture |

Compliance Coverage

|  |  |  |  |
| --- | --- | --- | --- |
| **Compliance Standard** | **Test Level** | **Powertrain Coverage** | **Key Test Cases** |
| **ISO 26262** *(Functional Safety)* | Unit → HIL | ALL | - ASIL-B: Critical warnings |
| - Priority hierarchy |
| **FMVSS 101** *(Controls/Displays)* | HIL → Field | ALL | - Speedometer tolerance ±3% |
| - Units compliance (mph) |
| **UN R79** *(Assist Systems)* | Component → Field | ALL | - Steering input feedback |
| - Lane-keep alert timing |
| **GDPR/CCPA** *(Data Privacy)* | Unit → System | ALL | - Infotainment data anonymization |
| - Log validation |
| **SAE J2402** *(Hydrogen Safety)* | HIL → Field | H₂-ICE / FCEV | - H₂ leak detection < 3s |
| - Tank pressure warnings |
| **IEC 62196** *(EV Charging)* | Component → Field | BEV / FCEV | - Charging status accuracy |
| - Connector fault display |

**Configuration Example:**

   "powertrain\_profiles": {

       "BEV": {

         "features": ["regen\_display", "charging\_status"],

         "compliance": ["IEC 62196", "ISO 6469"]

       }

     }

Summary/Conclusions

Deriving common test strategy is the challenging task with the consideration of evolving technologies in the automotive domain with respect to Architecture, Protocols, Specification’s, Vehicle Variants etc ,Its achievable with the right test approach , design and

References

1. SAE uses the [Chicago Manual of Style](http://www.chicagomanualofstyle.org/home.html) in formatting references. In the text of the paper the citations are numerically identified using square brackets [1]. Up to four authors should be listed; more than four, et al. should be used after the fourth author is listed. Refer to the SAE Technical Paper Style Guide for formatting of different types of references. Apply the List-Ordered-Numeric style tag to format references. Below are some examples.
2. Guo, Q. and Liu, B., "Simulation and Physical Measurement of Seamless Passenger Airbag Door Deployment," SAE Technical Paper 2012-01-0082, 2012, doi:10.4271/2012-01-0082.
3. Kunkel, S., Zimmer, T., and Wachtmeister, G., "Friction Analysis of Oil Control Rings during Running-In," SAE Technical Paper 2011-01-2428, 2012, doi:10.4271/2011-01-2428.
4. Morgan, R., Scullion, P., Nix, L., Kan, C. et al., "Injury Risk Investigation of the Small, Rear-seat Occupant in Side Impact," SAE Technical Paper 2012-01-0092, 2012, doi:10.4271/2012-01-0092.
5. Kimura, Y. and Murakami, M., "Analysis of Piston Friction - Effects of Cylinder Bore Temperature Distribution and Oil Temperature," *SAE Int. J. Fuels Lubr.* 5(1):1-6, 2012, doi:10.4271/2011-01-1746.
6. SAE International Surface Vehicle Recommended Practice, “Laboratory Measurement of the Composite Vibration Damping Properties of Material on a Supporting Steel Bar,” SAE Standard J1637, Rev. Aug. 2007.

Contact Information

Contact details for the main author should be included here. Details may include mailing address, email address, and/or telephone number (whichever is deemed appropriate).

Acknowledgments

If the Acknowledgments section is not wanted, delete this heading and text.

Definitions/Abbreviations

|  |  |
| --- | --- |
| SA | sample abbreviations |
| UBT | Use borderless table ≤ 3.5 inches wide. |
| test vector | Don’t capitalize term unless an acronym or proper noun. |

Appendix

The Appendix is one-column. If you have an appendix in your document, you will need to insert a continuous page break and set the columns to one. If you do not have an appendix in your document, this paragraph can be ignored and the heading and section break deleted.