

Using an Integrated Test Framework To Reduce Mechatronics Life-Cycle Costs

Chris Domin
MicroMax, Inc.

Copyright © 2008, 2009 MicroMax, Inc.

ABSTRACT

Mechatronics products contain significant amounts of software. Most advances in mechatronics software development focus on specific phases of the development process. However, very little emphasis has been placed on maximizing reuse of artifacts across the life cycle. This paper explores the potential of reducing overall costs of developing and testing mechatronics software by systematically increasing efficiencies.

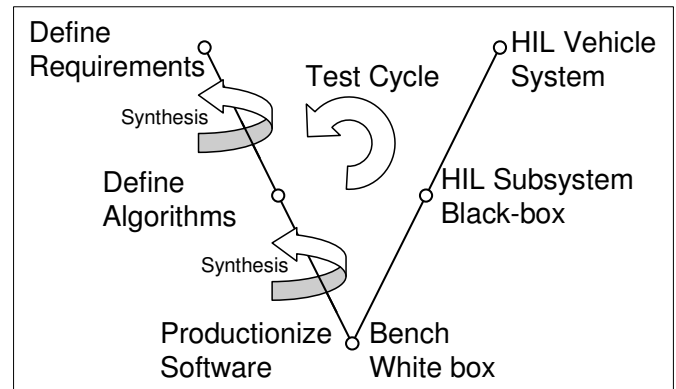
INTRODUCTION

Automotive organizations are aggressively searching for ways to reduce costs of capital and labor for production-grade mechatronics software, and, at the same time, increase test coverage and overall quality of the final product. Partial solutions have been developed over time, moving most software organizations towards model-based methods for developing of algorithms and software for controls applications. A robust, process-centric solution that fully realizes benefits of complete reuse has not been widely deployed across the OEM and supplier network.

Current best practices

The traditional development process is generally based on using the well-known V process (see Figure 1). Model-based tools have increased the efficiency of design and development by providing an economy of expression and standardized notations. In addition, the practice of applying continuous integration techniques (simulation and testing) to iteratively explore, design, and refine implementations within a particular phase of

development has reduced life-cycle costs. However, significant opportunities exist to reduce life-cycle costs even more by focusing on testing as an integrated process. Significant gains are realized as a result of increasing human efficiency, automation of repeatable



tasks, and leverage of artifacts.

Figure 1- Current Best Practices

A number of optimizations are already in use by the majority of companies in industry, notably:

1. Using simulation models to express executable behavior as use-cases.
2. Using co-simulation to validate system behavior.
3. Iteratively refining modeled behavior as requirements are validated.

4. Using automatically generated code from the simulation models to execute on rapid prototyping hardware.
5. Using automatically generated code for on-target applications.
6. Graphically expressing test stimuli for increased human efficiency.
7. Using scripts to automate testing efforts.

While these approaches have reduced development costs for organizations, there still exist significant opportunities for cost reductions in the complete product life cycle.

Efficient point solutions do not mean an optimized life cycle

It is well substantiated that continuous integration (concurrent testing) provides the most cost-efficient method of producing mechatronics software. Engineers test what they develop throughout the development process. They validate requirements and verify design of data flow models, state driven models, UML models, vehicle simulation models, 'C/C++' code, breadboards, ECU's, sub-systems, networks, complete virtual vehicles, hybrid systems (part virtual, part real), environmental robustness, EOL, and so on.

To date, tools suppliers have addressed the individual phases of the development process and provided a choice of products for each phase of development. However, the methods used to validate and verify the artifacts during each phase are particular to each vendor. There is not a consistent, integrated approach that optimizes for labor costs when transferring test cases and test data between tools or phases of development.

Consider the table in Figure 2. Representative tools vendors who have deployed state-of-the-industry products are examined for their ability to directly interchange test information such as test stimuli, pass/fail criteria, test results.¹

Each development and test tool has its own execution environment, with interfaces that support scripts and/or application programming interfaces. The services each tool provides are, for all intents and purposes, unique to the tool itself.

The consequence is that the tools are well coordinated with each other, and tend to take on a life of their own; due to the organization structure that supports them.

¹ The information shown here is representative only. There is no intent by the author to endorse any tool listed or discredit any tool not listed. The information shown has NOT been verified by as to its accuracy or validity and therefore all users are cautioned to use at their own discretion

The unfortunate outcome is the overall process is not optimized for cost or schedule.

Examining the tool interfaces further, one realizes that some tools import and export test data as comma separated values (CSV). While CSV provides connectivity to the tool, it neither describes the syntax of the data nor the semantics associated with the test cases.

Many tools allow import and export of test data and results in XML format. This is superior to CSV and allows syntax and partial semantics to be understood: that is, the interface exchanges *information* (meaning) rather than *data*. Some automotive OEMs have invested in programs to standardize information exchange in XML. Some have even chosen to build the Integrated Test Framework in-house.

So, as a stop-gap measure, many organizations increase their tool cohesion by adopting a data-driven approach to test, with MS-Excel as the repository for test data. By opting for a data-driven, as opposed to language-based approach to test, a step is taken in the right direction, but MS-Excel is certainly not the optimal data-driven enterprise solution for test.

		Interfaces						
		Exec			Data		Info	
		COM Automation	API	Script	CSV/Excel	Proprietary	XML	Proprietary
Representative Tool Vendors		Aonix					✓	✓
		ARTISAN			✓		✓	
Modeling		Excel Software		✓		✓		
		IBM-Telelogic	✓				✓	
		IBM-Rational	✓		✓		✓	
		Mathworks	✓	✓	✓			✓
		Microsoft	✓	✓	✓		✓	✓
Unit Test		IBM-Rational		✓				
		IPL					✓	
		LDRA					✓	
		Nunit					✓	
		ParaSoft					✓	
		Tessy					✓	
HIL Test		VectorCast	✓					
		add2	✓	✓				
		Agilent		✓				
		dSPACE	✓	✓		✓	✓	
		ETAS				✓		
		National Instruments		✓				
		Opal-RT		✓	✓			
Vector		✓						

Figure 2 - Execution, Data, and Information Exchange

Considering Figure 2 again, it shows the lack of portability of test information between development tools:

The result is that no matter how good the individual products are, it is still desirable to have a structure that supports a more cohesive test strategy because test data are created, captured and analyzed in so many different environments. This lack of cohesion disrupts how an organization operates by requiring specialists in each tool area. In addition, the artifacts produced during different phases of development cannot be leveraged by the organization. .

DESIRED EXTENSIONS TO CURRENT ENVIRONMENTS

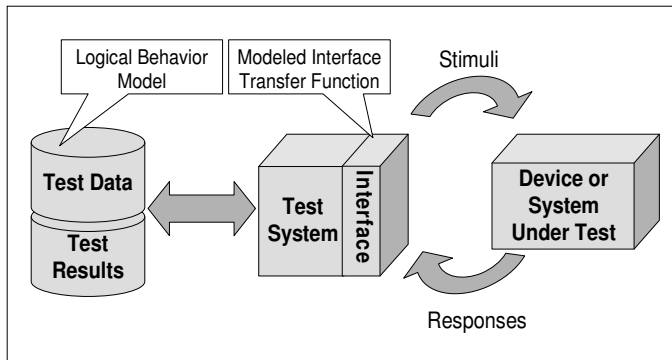
What is desirable is an approach that is focused on maximizing continuous integration and leveraging the V&V artifacts across the complete development cycle. The intent is to hide platform-specific behavior and provide the information that the organization wants to see. This could result in a reduction of labor costs ranging from 25% to 50%.

Formal Notation promotes communication and reuse

A simple definition for testing is:

- Predict what the system should do
- Stimulate the system and observe the response
- Measure and compare the response to the prediction

In defining what these tests look like, the engineer creates a model of the test in terms of its logical inputs, outputs, and timing, commonly called a test vector. Traditionally, when the engineer wants to execute the test, hardware and tool dependencies are intertwined with the test vector, and that distorts the model. This mix of domain-specific information and essential test behavior makes it more difficult to communicate what is happening to other engineers, managers, and groups, and greatly diminishes the potential for portability and



reuse.

Figure 3 – Approach to Create the Integrated Test Framework

It is desirable for tests to be formally represented with a logical behavior model; that is, stimuli and responses are characterized in standard engineering units. This allows the most compact form for specifying tests and pass/fail behavior. The interface from test system to the modeling tool, software, or hardware test environment is characterized by a transfer function to keep the physical mapping characteristics independent from the actual tests themselves. This compact form allows hardware and software independence of tests from the actual implementation (simulation models, software algorithms, and HIL test equipment).

This would benefit the total organization by providing a commonly understood notation, thus reducing the overhead costs for communicating and interpreting information costs by 75%-90%.

THE INTEGRATED TEST FRAMEWORK

Reviewing Figure 2, It should be clear that all the commercial tools have strengths for their primary purpose (execute a model, stimulate with high time resolution, measure with high precision, handle loads, or many i/o). Many seem to handle data flows of input and output test data. If these flows intercepted and an appropriate technology is chosen for the source and sink for these test data, the basis for an Integrated Test Framework is formed.

The benefits of using the Integrated Test Framework, rather than engineering labor, to orchestrate modeling and testing tools become obvious.

Benefits of an Integrated Test Framework

The idealized Integrated Test Framework (see Figure 4) allows the following optimizations and improvements to existing test frameworks:

- The ability to choose 'best of breed' tools across the development process, from modeling tools to HIL testers.
- The reduction in effort derived from being able to move test data and artifacts freely across the life cycle. The organization can plan on executing the exact same test cases on a simulation or virtual subsystem as on the actual electronic subsystem.
- The efficiencies of having a whole organization create tests, determine pass or fail disposition, and provide reports in exactly the same manner. The consistency of communication shortens meetings and review cycles.
- The security of having test data in an implementation-independent, translatable format (XML), so that faster, cheaper products are taken advantage of as they become available. The test data are not hidden and locked with the structure of some scripting language.

- The re-use of test data from program to program. Since test data are abstracted from the execution environment, they are now closely coupled to requirements. Test data only change with requirements, not with test equipment.
- Enhancing the requirements documents with logical behavior models (test data) to represent use case diagram. These are automatically executed by the Integrated Test Framework on the system under development in order to verify compliance.
- Not needing to purchase the ‘test front ends’ that are offered with many other test/development environments (e.g., Test Director, Signal Builder, Automation Desk, LabVIEW, CANoe, , etc.).
- The time saving realized on international projects where communication of test data, pass/fail criteria, and results is an order of magnitude more efficient due to their graphical presentation.
- The ability to send development and testing to low-cost centers where modeling, coding and testing is performed by a supplier or other outsource. The resulting software and test data, scenarios, results are sent back to the consumer organization where the artifacts are fully leveraged.
- An easy method for adding virtual prototypes for customer validation early in the design cycle. The virtual prototypes are supported through standard HMI² (e.g., GL Studio, Altia, Active-X panels, etc.).
- User-defined features (such as custom reports, custom test editing capabilities, specialized pass/fail algorithms, etc.) that benefit the engineers throughout the whole testing lifecycle.

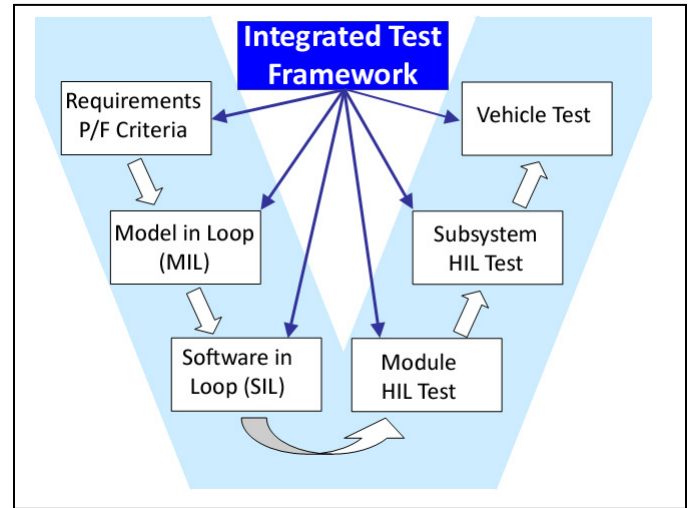


Figure 4 - The Integrated Test Framework

CONCLUSION

The first step to reducing mechatronics life cycle costs via the test environment is realized by separating test data (stimuli, pass/fail criteria) from the interface to the device under test. Storing test data and test scenarios in a universally understood format allows that information to be leveraged across the whole development cycle. The Integrated Test Framework will allow complete automation and reporting of test results. It will also allow developers to focus on feature implementation, rather than manually retesting features that “should just work”. Lastly, the Integrated Test Framework promotes concurrent testing during development, regardless of the tools used to model, develop, or unit test the implementation, and provides the most benefit to the organization.

CONTACT

Mr. Chris Domin
 MicroMax, Inc.
 5840 N Canton Center Rd, Ste 270
 Canton, MI 48187 USA
chrisd@mrmx.com
<http://www.mrmx.com>

² Human Machine Interface