Validation and Verification for Embedded System Design – An Integrated Testing Process Approach

*Adnan Shaout, Dennis Breton
The University of Michigan – Dearborn
The Electrical and Computer Engineering Department
Dearborn, Michigan

Abstract-- *This paper identifies the need for an integrated software solution to manage configuration requirements of currently popular software applications and tool chains used for the validation and verification tasks associated with embedded system design. The intended results of such an approach include: increased design development speed, decreased time to market and additional improvements in quality assurance. A theoretical unifying software automation process is proposed to link specifications and requirements documents not only through the design and test specification phases but directly into the testing platform itself including results and reporting. The result of this approach can be shown to streamline the design, verification and validation testing and reporting processes for an embedded software system. The observations and conclusions presented are based on an elementary user understanding of several software development tools and tool chains and their potential vs. intended extensibilities.*

Keywords: Software Tools, Validation, Verification, Testing, Embedded Systems, Software Automation Process, Integrated Testing Process Approach

I. INTRODUCTION

Human existence is defined in part by the need for mobility. In modern times such need is luxuriated and partially fulfilled by commercial interests of automotive and aerospace/avionic companies. In the terrestrial and aeronautic forms of personal and mass transportation, safety is a critical issue. Where human error or negligence in a real-time setting can result in human fatality on a growing scale, the reliance on machines to perform basic, repetitive and critical tasks grows in correlation to the consumer confidence in that technology. The more a technology's reliability is 'proven', the more acceptable and trusted that technology becomes. In systems delivered by the transportation industry: buses, trains, planes, trucks, automobiles as well as in remote systems in which humans play little or no directly controlling role such as satellites and space stations, computerized systems with active real-time response capabilities become the control mechanism of choice. The more reliably real time embedded systems perform the more they are trusted and their use proliferates to the control of critical systems. The use of microcontroller units in the automotive industry has grown in recent years so considerably that there may now be found from 50 to 80 electronic control units in an individual vehicle. In efforts to implement safety and redundancy features in larger commercial transportation vehicle such as airplanes, the 'x-by-wire' (brake by wire, steer by wire, drive-by wire, etc.) concept is now being explored and implemented in aerospace companies that make or supply avionic systems into a 'fly-by-wire' paradigm, that is, the proliferation of electronic control 'by-wire' units over the mechanical aspects of the system.

In the critical industries, automobile or aircraft development processes endure a time to market that is rarely measured in months but instead in years to tens of years, yet speed of development and time to market are every bit as critical as for small scale electronics items. The common thread among these commercial products is that they include embedded controllers and are therefore considered embedded systems. Product and process verification and validation, including end-of-line testing, contribute to longer time to market at the cost of providing quality assurances that are necessary to product development. In industry of small-scale or personal electronics where time-to market can literally be the life or death of a product, validation, verification and testing processes are less likely to receive the level of attention that they do in mission-critical or life-critical industries. An integrated software solution for validation, verification and testing procedures help address several issues including reduced complexity and decreased development time for embedded systems design of any size or scale.

An embedded system is a control mechanism that includes hardware and software components. Hardware

may include computers, microcomputers, and / or microcontrollers. Software components provide functional and logical control of the hardware. A major factor in the development time, cost and complexity of an embedded system is the software development. While the hardware provides the ability for a system to do useful work, the software controls an embedded system at all levels. The basic software is responsible for the low level functional control of the hardware components to ensure they are properly configured and integrated into the physical system in which they reside. The application software is responsible for control of the data input that is generally provided by external sensors and how that input is processed by the system, that is the application provides the hardware system behavior. While hardware presents its own set of complex challenges the concentration of this paper will be the software development in which a broad range of the complexities involved in the development of real-time critical large scale systems is addressed. The products being designed are highly complex. The processes of design and quality assurance are deeply complex. The tools that are used and the process of using those tools are broadly complex. These compartmentalized considerations underscore the need for a theoretical model for large scale system integration. By reviewing the subjects of design, development process, testing, verification and validation, it will be shown that there is ample need for software application tools that provide links through all the stages of software development intended to reduce interface and integration complexity and provide traceability. This paper will entertain and propose several possible considerations of both the problems and the solutions for potential commercial software applications intended to fill several presently void aspects for the realization of a potentially unified all-in-one process for the verification, validation and testing phases of a design project with focus on basic and application software and configuration and change management. Although there may be many possible solutions, this paper will present ideas that may contribute to increased speed of design, verification and validation testing and reporting processes for an embedded software system.

The paper is organized as follows: section II will present an overview of a software development process, section III will present development process: requirements and resource methodology, section IV will present validation and verification methods, section V will present integration design process with validation/verification process: argument for an integrated solution, section VI will present testing at every level, section VII will present scaling upward toward fully integrated solutions, section VIII will present comparison of commercially available verification and validation tools, section IX will present the problem statement, section X will present the proposal for integrated solution, and section XI will present concluding remarks.

II. OVERVIEW OF A SOFTWARE DEVELOPMENT PROCESS

There is no argument to the usefulness and benefit of a software development process when considering a commercial product that employs an embedded system. A development process provides a developer (individual, group or enterprise) with some degree of control, predictability, traceability and repeatability. There are several useful process paradigms commonly used including the waterfall model of which the V design paradigm is a derivative process. The V development process depicts a modifiable paradigm in which the design verification phases of requirements management, system design, architecture design and module design flows down one branch of the V from high level (abstracted) to low level (detailed). The design validation phases of unit testing, integration testing and system testing flow up the opposing branch of the V from low level to high level. Each successive level or phase relies on the accumulated framework of a previous phase for its developmental foundation.

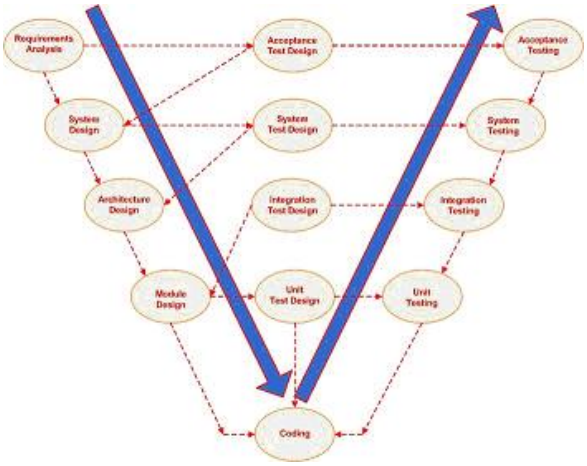

Figure 1: Modified "V" Development Process [1]

Figure 1 depicts a V-model software development process that has been modified to include concurrent test design phases that are derived from the fundamental stages of the design process. Extensions to the V model such as the Dual-V provide for further concurrency by allowing for phases to be stacked (layered) in the time domain (adding dimensionality) even to the degree that a stage may itself spawn an orthogonal design tangent to the original V, further multiplying the concurrent

possibilities that lead to a linear decrease in development time. A key feature of figure 1 is that test designs are linked to phases in both the design arm (the leftmost process from high level to low level) of the V and the Verification and Validation arm (the rightmost branch of the V from low level back up to high level). Although this process suggests that a testing suite for complete verification and validation is easily linked throughout the design process, in practice this is far from the case. An internet survey for verification validation and testing process software application tools quickly reveals the absence of unifying tool support for conjoining the terminal phase to supporting pre-design and post-design documentation as is vastly utilized in the design branch of the process. There is a variety of commercial software tools available that provide process stages some which singularly or in tandem incrementally approach a complete solution, however nothing currently fills 100% of the void.

Figure 2: "V" Process Model modified to indicate the potential for a completely unified process approach [1]

Figure 2 depicts a Flow diagram of product–process development based on the V model. The intention of this diagram is the graphical representation of series and parallel activities at several levels of detail over time during the development of a product. The subset of phases along the bottom of the diagram effectively represent required activities that are not necessarily associated with particular phases connected to the V path, however constitute their own set of interdependent phases that may occur along the timeline as the process is implemented from the leftmost to the rightmost phase of the V represented phases. This diagram is color coded such that commonly colored stages represent a known and/or proven process trace among like-colored phases. For the phases and chains of phases colored in red, Software tools are not available. The phases color coded in yellow represent emerging software tool developments. For the green colored components there may be one or more well known or proven software tools however these may not necessarily be interoperable or are used inefficiently [2]. Figure 2 plainly indicates the lack of conjunctive applications to facilitate a unified process flow conjoining requirements and design to validation and verification phases at all levels of development. Some design/development platforms however such as MATLAB/Simulink offer solutions that partially bridge the gap. Some source integrity and requirements management tools also provide solutions that may however involve complex configuration management or require users to learn additional program languages or syntaxes.

III. DEVELOPMENT PROCESS: REQUIREMENTS AND RESOURCE METHODOLGIES

In reference again to figure 2 it is plainly evident that a high degree of parallel development is desirable and should be achievable. By deployment of concurrent development the cycle is shortened along its linear timeline thereby potentially reducing time-to-market of a product. While having obvious practical implications, the implementation of such an enhanced development paradigm however is farther from a plug-and-play solution than might initially be expected. The problem lies mainly in the lack of an all-in-one utility that particularly unites the requirements and resource development stages with the validation, verification, change management and test reporting stages. As a contextual foundation it is natural to expect source integrity or requirements management tools would provide the required extensibility to allow for a completely unified development environment. A brief overview of such tools as follows indicates the contrary. However with further investiture the contextual foundation concept may be eventually grown to necessary proportions to accommodate a truly integrated process and development environment.

A. Source Integrity and Repository Bases

Source integrity tools such as MKS, a data management tool that is intended to provide secure repository services for project resources, provide a database foundation for process elements. The MKS tool is primarily adept at database management for elements in the design, validation and reporting phases of development. A project can be safely versioned, stored and securely managed in MKS however does not provide checks and balances against individual user sandbox development. While this does allow for nearly unlimited development concurrency, it can also blind

each developer from the work of the other which may in turn result in concurrent development proceeding in many different directions. Because source integrity tools allow resources to be continually added to a project database or namespace at any time during development, these applications provide a thorough resource and configuration management service, but do not necessarily facilitate cross-platform utility. Depending on the level of user sophistication such resources can support workaround applications for example using a MATLAB/Simulink model file (.mdl) to read data (parametric I/O, test scripts, etc.) from a Microsoft Excel spreadsheet, process the data and write the result back to a file, however this is done on an individual user sandbox platform and does also require some managerial arbitration of model configurations that are permissible to be checked-in to the repository. The project member versioning update paradigm used by MKS does provide for protection of baselined resources but can not protect against the propagation of faulty or erroneous member versions contributable to human or machine error. Traceability is supported through member versioning however without highly disciplined administration of the tool use and privileges, erroneous work can be extensively 'fanned out' especially across a highly concurrent development environment.

B. Requirements Management Platforms

The website for Reqtify, A Graphical Requirements Traceability Environment accordingly reports: "Requirements Management is now recognized as a Best Practice for project management. At a minimum, a sound requirements management process is mandatory for CMMI level 3" [3]. The preceding statement underscores the importance of oversight and leadership at the highest project level.

IBM/Telelogic DOORs is a requirements management application that provides "a variety of features, such as views, links and traceability analyses" [4]. Features of newer releases (currently 9.5 dating back to version 8.1) takes advantage of a Uniform Resource Locator (URL) identifier protocol handler that permits resource locating for boundary crossing between file namespaces. This is a major step in the direction of allowing access to and from database files that are not necessary part of a DOORs project. This is an essential mechanism for providing "cross-database linking" access between independent software platforms by assignment of unique URL identifiers to each internal resource (object, database, module, folder, etc) and extension of these URLs to any URL aware [5] browser application. Although this extensibility is provided as part of the software package, it is expected to be implemented by the users who "are responsible for writing a protocol handler and setting up the communication channel between the protocol handler and the application that will open the URL" [5]. Versions since 8.3 of this software are extensible for test tracking system creation that allows creation or importing of Test Definition resources. Through the DOORs adaptation of the URL protocol, traceability is achieved within and without the repository. One of the inherent deficiencies of DOORs is that to run active test utilities the testing platform may be required to sustain a real-time connection to the repository server which can severely limit accessibility to physical test environments and may inhibit the achievement of necessary processing speeds for desired test data capturing rates in real-time environments.

Some graphical requirements management tools like TNI-Software/ChiasTek Inc.'s Reqtify offer solutions for "Traceability through Entire Project from High-level Requirements to Models, Code, Test Scripts, and Test Results" and a "solution to link development and verification processes with requirements" [3]. The implication of these claims is supporting traceability from requirements to test scripts and test results however there is apparently no facility provided for generation/auto-generation or template type creation of validating test scripts or verifying test environments. Therefore such tools stop short of offering any new linking capabilities beyond those that already exist in other development design paradigms, thereby providing only an alternative choice and not necessarily a further extensible one.

IV. VALIDATION AND VERIFICATION METHODS

Model Based Development/Design has become a standard engineering industry operating procedure in CAD/CAE. There are several proven capable and trusted tools for graphical modeling and simulation of commonly engineered systems such as manufacturing, electrical, medical, computational, mechanical, and communications. Commercial software simulation tools are presently in a highly advanced state of development having long since proven their usefulness and reliability in many engineering fields in the global marketplace. The concept of graphical modeling is simple representation of any physical system by its inputs, a black box containing functional logic and outputs. The approach can be a top-down or bottom-up hierarchical structure within which each a black box may contain multiple subsystems with the lowest level containing the basic logic and arithmetic operations, even down to bit level control if so required. The architecture of a given physical system is graphically arranged or designed to best simplify conceptualization and understanding of underlying logic. This has

tremendous advantages over interpretation of a system by analysis of potentially hundreds of thousands of lines of code. Extensive efforts are made by simulation suppliers to continually upgrade and extend the application potential for their products however less progress has been made towards easing integration complexity.

Validation and verification can be viewed as an integral part of what control theory considers the "formulation of an optimal control problem" [6]. Description and statement of physical constraints are the first two stages and the performance measure is considered as the final of three major stages of problem formulation. Included tasks in problem formulation are: mathematical modeling such as creation of state diagrams, Mealy-Moore finite state machines or through the use the software tools that represent states and transitional conditions graphically such as Telelogic Statemate or MATLAB/Simulink/State flow. The statement of physical constraints equates to a requirements specification of a system. Together these phases provide the leftmost branch of the V-design paradigm relating to design which must undergo validation. Each of the hardware and software components of the system may have their own set of requirements documents extending down to the lowest modular level of the basic system component. The mathematical model may be replaced with a virtual or conceptual representation through block diagrams or graphical user interface software that can interactively or reactively describe the behavior of a physical system by using black box components to represent the functionality of a system down to its lowest modular level. The usefulness of this approach is that in a well modeled system using the proper software tools, the generated software that commands the model can be optimized as source control software for micro processors and micro controllers in the real physical system. This approach also lends itself to an iterative validation or verification approach in that the custom hand written or software computer generated code can be tested at multiple levels and with a number of interfaces that progressively approach integration into the final physical system. This iterative approach is commonly recognized in modern design engineering as the successive in-the-loop processes of MIL/SIL/PIL/HIL testing.

Model in the loop (MIL) occurs with model components interfaced with logical models for model level correctness testing.

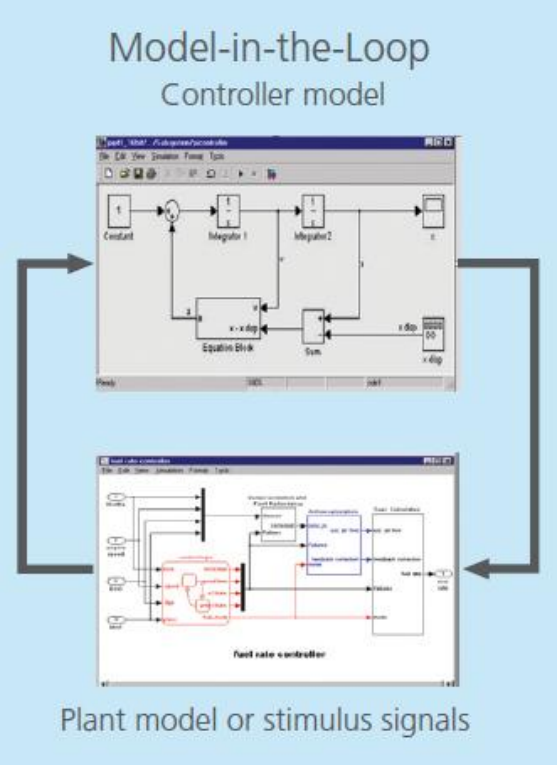

Figure 3: MIL process from dSPACE Catalog © dSPACE 2014[7]

As illustrated in figure 3 culled from the current dSPACE 2014 product catalog, a controller model is introduced into a closed-loop system with the logical model. The control model is not limited to a model based design. It can be any variety of testing extension, tool box, or third party application that can be interfaced with a graphical programming tool such as MATLAB/Simulink or dSPACE/Targetlink.

MIL testing of a software system design is the initial phase in which a system that is considered ready for testing is subject to simulated environmental control. The base model considered to be the plant model which includes the stimulus signals is the constant factor in the loop-testing sequence. The plant model is driven or stimulated by a specially developed controller model. In the same context that the software itself represents a physical reality, it is reasonable to expect that a software representation of inputs to the system can achieve the desired validation results. As the system has been designed and is ready for testing, so can test software be designed to represent real world input to the system. A reasonable validation procedure can be undertaken by replacing inputs with data sources that are expected, calculable or otherwise predefined and monitoring the output for expected results is an ordinary means of simulating real system behavior.

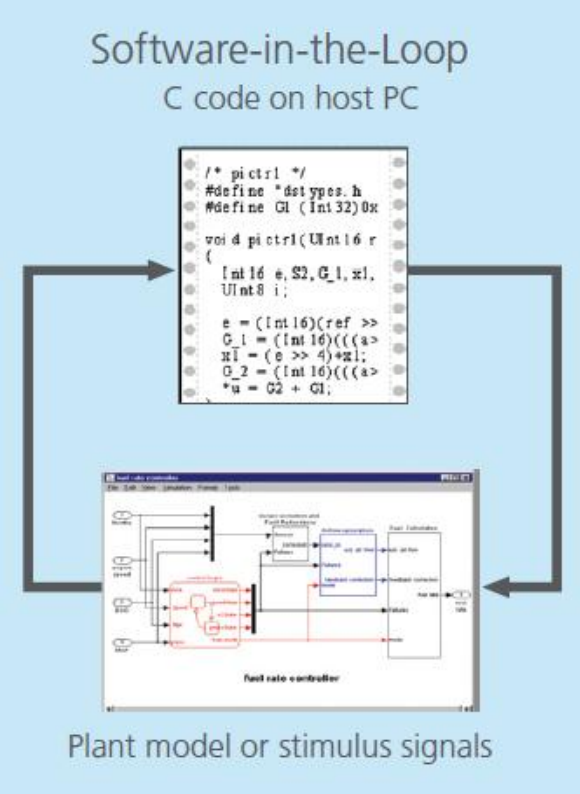

Figure 4: SIL process from dSPACE Catalog © dSPACE 214 [7]

Figure 4 represents the closed-loop system consisting of design model software interfaced with c-code running on a PC hosted application. The plant model remains consistent inasmuch as its performance proved satisfactory at the previous MIL stage. Software in the loop (SIL) occurs after code has been generated from a model and run as an executable file that is configured to interact with the plant model software. This midway point in the V design methodology is perhaps the most important stage of testing as the progression will begin at this point to lead into hardware testing. This is the optimal stage at which code optimization for hardware should be considered, especially before the configuration grows in complexity. Code optimization is dependent on the constraints of the design under test (DUT). For example it may be necessary to minimize line of code count in order to not exceed ROM limitations based on particular micro-processor architecture. Other code optimizations may be aimed at RAM availability and can include pipelining and loop unraveling.

From this point Processor in the loop (PIL) testing is undertaken for proof that the generated code can run on a hardware platform such as micro controller, E/EE/PROM or FPGA. Figure 5 represents the PIL stage by illustrating the inclusion of an 'evaluation board' however this may be any target processor platform sufficiently design to show proof of design at the project level. Once this stage is adequately secure Hardware in the loop (HIL), if applicable to a system, is performed. At this point a test procedure such as JTAG/boundary scan may be considered for hardware testing prior to implementing a system under test (SUT) scheme. Joint Test Action Group (JTAG) Boundary scan specification outlines a method to test input and output connections, memory hardware and other logical subcomponents that reside within the controller module or printed circuit board. The JTAG specification makes it possible to transparently access structural areas of the board under test using a software controlled approach. By isolating physical substructures on a circuit board these areas can be considered as independent circuits. Path tracing on individual areas can then be implemented to reveal unreachable real estate in the circuitry, signifying a defect in the hardware. This technique also allows further software control and test procedures can be implemented across the circuit subsystems using a 'scan chain'. When used in conjunction with built-in-test (BIT) testing modules coded into the control software, the JTAG/boundary scan procedure can be extended to act as a debugger and diagnostic tool for embedded systems.

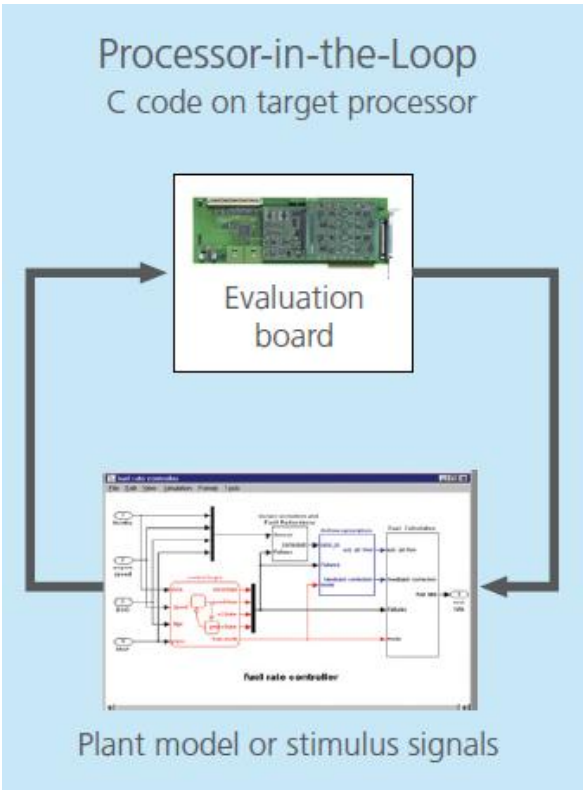

Figure 5: PIL process from dSPACE Catalog © dSPACE 2014 [7]

Once there is certainty that the software performs as intended and that there are no defects in the hardware, the final 'In the Loop' stage Hardware (HIL) is undertaken to prove that the control mechanism can perform its intended functionality of operating on a hardware system.

According to joint open source initiative United Simulation Environment (UNISIM @: www.unisim.org) "Simulation is a solution to the test needs of both microprocessors and software running on microprocessors" [i.e. embedded systems]. "A silicon implementation of these microprocessors is usually not available before the end of the architecture design flow, essentially for cost reasons. The sooner these simulation models are available, the sooner the compilers, the operating system and the applications can be designed while meeting a good integration with the architecture."

Design Verification Process Validation (DVPV) Testing has become overwhelmingly reliant on graphical design with simulation tools such MATLAB/Simulink, dSPACE/Targetlink, Simplorer, and IBM/Telelogic/Statemate. This is in part due to their sophistication and versatility and largely due to their added functionality of automated code generation. Simulation tools make for excellent testing tools because they are capable of providing instantaneous feedback to system or subsystem design. Test input may be provided internally, modularly, and/or from external scripts/application resources. Simulation allows developers to quickly test designs for completeness and correctness and many tools also offer auto-generated test reports such as for code coverage and reach-ability of generated code.

As mathematics and physics are used as interpretive similes of real life systems, so computer aided simulation offers a means of interpretive modeling of those systems. Sophisticated software applications allow increasingly numerous phases of design and development to remain in a unified development environment. Such an environment may include single or multiple-tool custom tool chains where the software applications required are correlated to choice of target hardware (micro-controllers, communication networks, etc.) or the need for a sophisticated compiler/debugger scenario in the case of some more sophisticated projects. For example a configuration of software tools to support the Motorola MPC555 can be implemented with a particular MATLAB configuration while support to develop a system using an NEC V800 series micro controller could include MATLAB but would additionally require dSPACE Targetlink and a debugging tool such as included in the Green Hills Multi Integration Development Environment tool. While there may be redundancy amongst some software applications in the supported hardware, there is presently no single tool that easily configures to a broad base hardware support. The ongoing development of many of these tools is largely relegated to the logical and functional domains while the needle has barely been moved in the domain of external interface configuration, integrated hardware testing, and change management. While simulation remains the most common verification method for embedded systems design, there is room for vast improvement that a move toward a unified integrated development environment would be duly suited to.

V. INTEGRATING DESIGN PROCESS WITH VALADATION/VERIFICATION PROCESS: ARGUMENT FOR AN INTEGRATED SOLUTION

Testing, debugging, verification and validation are inarguably essential tasks of embedded system development regardless of the process adopted. By extending the abilities of existing tools to enable the addition of an integrated test procedure suite and making it applicable at the earliest development stages, the ability to move from left to right across the "V" gap can become greatly enhanced.

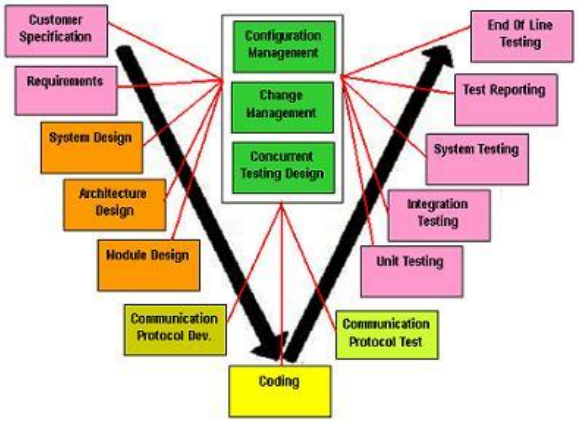

Figure 6: Theoretical V-design paradigm depicting one-to-one, one-to-many, many-to-one, and many-to-many associations between design, testing and management phases. The implication of these associations is that they should take place in a single unified development environment.

Figure 6 depicts a theoretical V-design paradigm in which design flow can be considered to spiral through any given combination of the design, testing and management phases even as the overall V-progression propagates from left to right, and through top-bottom-top progression. The associations amongst phases can and should be not only one-to-one but may concurrently

be one-to-many, many-to-one, or many-to-many. A further consideration is that the entire design paradigm should be contained within a single software/hardware development platform. This conceptual framework should combine the current abilities of resource management tools such as DOORs, source integrity database and versioning tools such as MKS, model based design tools such as MATLAB/Simulink and dSPACE Targetlink, in-the-loop-testing paradigms such as provide by dSPACE, sophisticated compiler/debugger and IDE utilities such as Green Hills' Multi environment, communication protocol software such as CANoe/CANape, measurement testing and data logging tools such as National Instruments' LabVIEW. A change-management, versioning process and report generation facility should all be included, eliminating the need to leave a single unified development environment. These applications can be plugged in to such a work flow block diagram as follows below in figure 7.

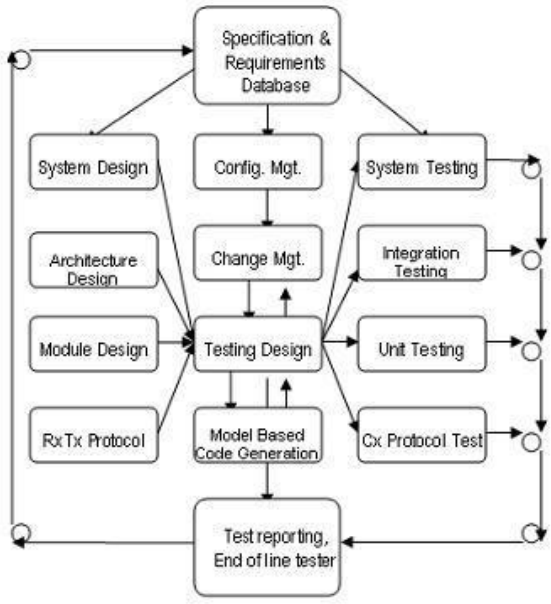

Figure 7: Work flow block diagram where blocks represent development stages that can have applications substituted for each process phase.

It is easy to recognize that a developmental paradigm such as the one depicted in figure 7 poses a highly complicated configuration process to represent as a single tool chain. It is therefore suggested that a use of single tool approach can drastically reduce the complexity and time spent on such a configuration. What can be gained from this approach is a very high degree of concurrent development bolstering early fault detection, design enhancement and the potential shortening of overall development time. The existence of such a tool could easily eliminate hundreds of hours spent just in configuration of such a tool chain of independent applications. Although the diagram in figure 2 is somewhat outdated, it clearly depicts deficiencies in current software tools and tool availability to meet well defined tasks and requirements. While there are Software companies that are making inroads to these territories, it is also evident that gross discontinuities exist between the conceptual framework of a development process and a real-world ability to economically implement such a process. Furthermore, figure 2 makes clear the need for unification of the sub processes that can lead to the unification of an entire system process that allows real-world implementation of the theoretical model. The color coding in Figure 2 represents a bridging process applicable to components of an overall system development process considered analogous with concurrent engineering design, and design for manufacturing and assembly which are also accepted development processes. It is evident that an evolution towards integration of these sub-processes can increase oversight and concurrency at all levels of development.

Typical engineering projects of systems with even low to moderate complexity can become overly convoluted when multiple tools are required to complete the various aspects of the overall system tasks. It is typical for a modern software engineering project to have multiple resource databases for specifications, requirements, project files, design and testing tools, change management and reporting formats. A fully integrated and unified process that bridges the V gap would solve the problem of configuring multiple tools and databases to meet the needs of a single project. Furthermore such an approach can simplify a process that follows recent developmental trends of increased utilization of a 'model based design' paradigm.

Potential benefits of an integrated development process include

- high degree of traceability resulting in ease of project navigation at all levels of engineering/management
- high degree of concurrent development resulting in reduction of overall project development time/time to market
- testing at early/all stages enabled resulting in potential for improved product and reduced debugging costs.

These benefits alone address several of the largest issues faced by developers to improve quality, reduce costs and therefore remain competitive in the global marketplace. Benefits also apply to other developmental

practices adapted to enhance recent-trend design and quality processes such as the model based design paradigm in which testing can be done iteratively throughout the entire development process. An integrated process can also add utility to reiterative process structures such as CMMI and Six Sigma/DFSS which have become instrumental practices for quality assurance.

In model based design testing an integrated development process will enhance and simplify procedures on multiple levels. Because model based design lends itself to modularization of components using serial systems, parallel systems and subsystems that may be embedded ad-infinitum, testing can begin near the beginning of the design process as opposed to a post-integration phase as in legacy testing paradigms. Component level, system level, software and hardware testing can be increased and testing can begin at earlier stages in the design. Testing can additionally occur with more concurrency due to the modular nature of the newer design paradigms, decreasing the time to market via a parallel/pipeline type of approach.

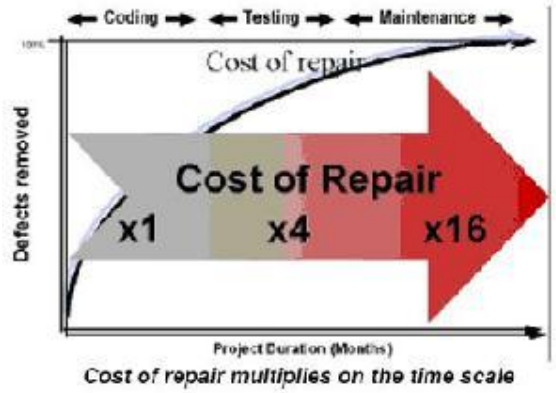

Figure 8: Cost of repair increase estimated against development time scale [8]

As indicated in figure 8 above, changes and improvements made later in the design process are far more costly than those made in the earlier stages.

Validation and Verification procedures are a certain means of improving product quality and customer satisfaction. By applying such procedures at every step of the development, an enormous cost savings can be realized by iterating improvements at the earliest possible stage of the development when integration is considerably less complex.

VI. TESTING AT EVERY LEVEL

Like the product design itself, it is important that a clear and concise testing specification is provided so that appropriate tests can be developed. Test specifications should have a built in mechanism of flexibility so that additional tests can be developed that may not have been considered in the original concept of the design. The test specification development stage can be greatly enhanced when test requirements documents reside in the same source repository as hardware and software requirements documents and are likewise derived from the original customer specification documents and traced through object linking. Creation of test plans can therefore have flexibility and also be created or changed any time a requirement object changes, is added or deleted.

With available test requirements and a source repository/requirements management system in place, testing plans can be devised concurrent in time to design development along the initial arm of the V model. The proposed scheme of a unified testing, validation, verification and development software tool or utility can greatly increase the level of control maintainable over the lifecycle management aspects of a project and take concurrent test development to new heights. By implementing and maintaining a configuration that reduces the number of software platforms deployed in a standard embedded system design project the development process can also gain the effects of enhanced traceability and enable testing to occur at every level.

The traceability inherent in a unified integrated solution is an important aspect which simplifies the process of relating auto-generated code back to its source. Furthermore by forward and backward linking ability, greater traceability can be achieved across the entire environment e.g. from a test report to executable code, to a model, to requirements documents, to the customer specification. Linking across test requirements and specifications can also provide for useful data management strategies such as the separation of operating system and application software to separate storage drives than test application software and test log files. A linkable platform allows an engineer to have immediate access to these separate directories or servers from within a single configurable environment. Enabling of linkable functional test stations can be made to occur in intermediate stages throughout the manufacture process and ultimately for End of Line (EOL) testing where the final product is tested at the end of the assembly process in a simulated use environment. The test bases can remain as part of the development environment with strong basis, perhaps automated derivation, from the available test requirements documents. Since the most important aspect of complete and adequate project DVPV could be considered to be testing requirements documents, it

follows that a direct linking from test specification to a design plan and further to a design report would add both economical and temporal efficiency across the process.

According to "Professional quality management and assessment when developing software for embedded systems", a white paper from IBM/Telelogic: "Wrong estimates that are frequently made in the planning of complex software development projects are often due to the erroneous assumption that less time is needed to test software than to actually specify, develop and encode it. And since the implementation phase takes longer than was originally assumed in the vast majority of development projects, the test phase that is planned to follow on after it gets shifted back too" [8]. The implication of this statement is that test planning and implementation that is delayed until later stages in the development process may be marginalized or omitted based on an improperly budgeted project. With a unified approach testing can be projected, planned for and initiated at the earliest design stage. The fact that systems and system components may undergo multiple revisions during or preceding any of the stages of the development process reinforces the need for early test development that is also linkable / traceable to its requirement source for increasing change management control. Also, well documented test cases, test scripts and test results will have the added benefit that in later maintenance stages of the process when the original architects, engineers, designers or suppliers may no longer be retained or available, in which case there should be clarity in documentation that guarantees any qualified entity or individual will have documentation available to enable quickly coming up to speed on the details and needs of the project. This can be affected by having the individual test environments embedded into a sole development process.

Admittedly even the major software system application developers are aware of the inability for the marketplace to keep pace with the academic and theoretical progress: "testing and debugging tools have not kept pace with the increases in embedded software size and complexity. As a result, the cost of testing an embedded system today can be up to 50% of total development costs" [9]. Early code verification can help to reduce the number of latent faults, that is, run time errors that are not evident in normal operational testing. The types of code verification testing usually employed are: manual code review, static analysis (evaluation of static expressions, constants, etc.), dynamic (script based, batch based) testing.

Currently there is emphasis from software testing tool providers for automation of testing. Automation offers advantages such as reduction of erroneous data logging or transcription, and improved repeatability. Accordingly for systems with "complex operations with many inputs and outputs" test automation provides the advantage of "increased data throughput" [10].

Other potential befits of automated testing is summarized as follows:

- elimination of redundant tasks such as redefinition of the test specifications to be used in a test plan
- re-usability – in which a test configuration can be reused on the same or different projects by modification of only parameters or parameter names
- allow for the use of tool sets or software applications to port across platforms by placement or automated recognition of files extensions or types which in turn eliminates some human error in terms of physical portability [9].

The most important effect of testing at every level is that requirements documents from a test specification can be developed, tested, reported, linked and updated iteratively and concurrently.

VII. SCALING UPDWARD TOWARD FULLY INTEGRATED SOLUTIONS

By way of acknowledging the absence and need of an all-in-one based development approach, some companies are moving in the direction of offering extended applications to their existing tools sets either through internal research and development or acquisition and assimilation of third party software applications. Software Tool providers must continually provide improvement and extensibility of their products in order to maintain a relevant position in the marketplace. The growing complexity of these tools, the need to keep pace with technological advances and theoretical design ideologies places considerable burden on tool providers. The inability of software process tool development to provide a unified process is evident by the variety of applications available to patch between design and testing environments and the general lack of all-in-one or multi-tools to meet design validation and verification demands. The resulting disparity between acceptable theoretical development paradigm and the availability of a tool to fill those requirements is plainly revealed. Due to colossal time and financial resources that impede the development process itself, the gap between need and fulfillment is further perpetuated. Even though a given software tool provider may be

presently engaged in research and development of a unified solution, many providers are nonetheless forced into the position of stop-gap (quick fix) solutions in order not only to remain competitive, but to avoid obsolescence. One approach has been incremental steps, micro-solutions, modulating and linking smaller processes and software solutions. One problem with this approach is that it moves software engineering processes further away from unification because the overall integration solution can become further complicated by difficult configuration strategies. In extreme though not uncommon cases engineers are expected to learn entirely new programming languages to support new tools and tool functionalities. For example IBM's Telelogic DOORs offers extensibility through development of external interfaces or applications executable though a proprietary language "DXL". Another drawback of such a stop-gap approach is that by requiring a user defined tool set extension focus from the project is detracted and forced instead to the underlying process which can severely inhibit productivity and introduce further engineering complications and error.

Some patchwork offerings are available that make initial inroads toward a process that may be more or less easily propagated further through a tool chain. However these tend to offer features defined by the provider and may resolve only small partial issues relative to a universal environmental approach. One example is an offering from Telelogic; the tool TTCN for Testing and Test Control Notation [11]. This tool is intended for use in conjunction with Telelogic's source repository tool DOORs, although it may also be used as a third party standalone or add-on to other development environments. TTCN offers the ability for testing modules and application interfaces (APIs), however with the expectation that the user has a desire to learn an entirely new programming language. Contrarily a system with a more user friendly graphical interface not unlike navigating a modern webpage would naturally reduce development time over one that would require new programming knowledge. To utilize the TTCN conceptual framework, the Telelogic company commercially markets an "integrated environment" that provides utility to the new language and framework. The Telelogic Tester provides several utilities whose need has been extolled here such as a high degree of automation however again; this company expects that users of this tool chain will afford the time and financial commitment to learning a new programming environment.

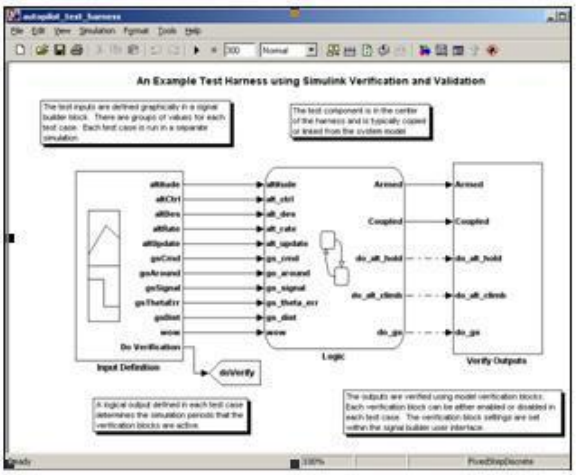

Figure 9: MATHWORKS® Simulink Verification and Validation Tool box example. A platform for logic testing application extensions. An example of a model based development platform [12]

Figure 9 above depicts another example of a utility add-on. MATHWORK's Simulink Verification and Validation add-on "enables you to develop requirements-based designs and test cases in Simulink and Stateflow and measure test coverage. By linking requirements to your designs and test cases and performing coverage analysis at the model level, you can trace requirements, validate your design, identify inadequate requirements, and expose unnecessary constructs and design flaws" [11]. However this tool does not extend across platforms toward lower level code generation testing, validation and optimization for production code outside of the MATLAB MIL environment and is thus not applicable to HIL testing. Mathwork's Simulink Verification and Validation Tool box provides one small incremental step towards testing integration into the overall development however does not match the capabilities found in other testing suites such as National Instruments (NI) LabVIEW. Like Simulink, LabVIEW (a graphical programming paradigm with its own G programming language) is further utilizable as a control and data acquisition package supporting HIL level testing external to the design environment.

VIII. COMPARISON OF COMMERCIALLY AVAILABLE VERIFICATION AND VALIDATION TOOLS

Many code-generation, partial process automation, or test-bench generation tools require the use of additional software tools for patching through to another root software platform to complete an uninterrupted tool chain. Here follows a brief overview of commercially available tools that are integral pieces, providing essential large-stage or small-step additions to a movement towards a universal all-in-one tool paradigm.

A. MATHWORKS HDL CODER*:*

MATHWORKS provides a widely used and highly sophisticated tool set for model based design and a variety of code generation utilities for application software development. One example of an extension product that does not fulfill the implications of its utility is the Simulink HDL Coder [13]. Hardware Description Language (HDL) Coder is an extension of the model based development package whose intended use is the auto-creation of HDL code for use in a third party synthesis tool. Although the HDL coder offers automation of a test bench and saves the user from learning additional software programming languages (VHDL or Verilog-HDL), it still lacks a complete solution because the tool requires the acquisition of other tool sets: synthesizers such as Synplify® and simulation tools Mentor Graphics® ModelSim® simulator or Cadence Incisive® to affect code instrumentation (device programming with production ready micro controller code). The end result is that this tool is really just a conversion instrument from one type of simulation (model based) to another (hardware description) prompting third party providers such as Impulse C to develop their own code optimization extension tool, again with the impetus landing on the engineer to learn to navigate an additional development platform complete with new flavors of proprietary c-code syntax.

B. dSPACE Targetlink Production Code Generator

dSPACE is an internationally reputable provider of complete system development software (Control Desk and Automation Desk IDEs, Targetlink Model Simulator), hardware (HIL testers, load-boxes) and staff solutions for Automotive and Aerospace industries. Basing observations on page 186 of the 2014 dSPACE Product Catalog [7] design and code implementation, dSPACE offers a complete MIL/SIL/PIL integrated environment. In the dSPACE design workflow Modeling Simulation and Code Specification encompasses several tasks including behavioral validity checking that can be tested in Model in the Loop Paradigm. This stage is related to reference checking. The next stage of the design process testing is software in the loop in which production code is hosted in the simulation environment. This stage is related to precision checking. The next stage of the design process facilitates "Production Code Target Simulation" that encompasses several tasks including target code verification testing with processor in the loop evaluation. This stage is related to optimization of production code. What the architecture of this environment lacks is clarity as to the configuration and interface needs and complexity required to link Simulink modeling, third party calibration tools and ECU programmers. Again the onus is left to engineering for configuration and interface.

C: Micro-Max Technology MxVDEV– Unit/System Test Tool Solution

Micro-Max Technology offers a foundation development environment, a Virtual Development Bench of programs used for requirements capture, design, development and test of real-time control systems. Some components of this suite include:

- Mx-VDev™ Unit/System Test Tool
- Mx-Sim™ System Simulator
- Mx-Automate™ Continuous Integration Tool
- Mx-Xchange™ Test Data Interchange Tool

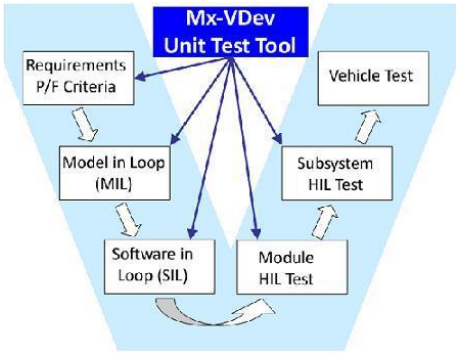

Figure 10: Micro Max Technology's Mx-VDev™ Unit/ System Test Tool [14]

Again the referring back to figure 2 the configuration given implies a complete development environment yet remains vague in the areas of model development, integration and especially verification, validation and test reporting as the project moves towards completion along the right arm of the V. Alternatively figure 10 indicates the extensibility requirements of tool chains utilizing the V design in which the Mx-VDev unit test tool can provide an integral component for integration of test development. Other issues involved with this and other tools include server/resource repository storage, mapping and configuration.

D: Hewlett-Packard Quality Center Solution

Adding yet further quality management concerns into the mix, Hewlett-Packard provides their Quality Center Solution tool boasting an enterprise ready integration approach to quality management that extends visibility and control up to a generalized/project management level including "out-of-the box capabilities for SAP, SOP and Oracle quality management" [15]

environments. This high level control environment empowers the strictest management up to and including purchasing, billing and individual time management control over all aspects of a project. This tool enables top-level project management with respect to business strategies but provides little or no facility for low level engineering. For this reason the Quality Center Solution does not particularly lend itself to a software development process.

Table 1 Part A and Part B indicate a comparison of the features and available utilities in commercial off the shelf software applications (COTS). Note that a process can be traced across the development phases by connecting a variety of tools however no single tool provides a direct trace through an entire process.

Table 1-A

COTS Software Application Comparison Table 1 part A.

	Source Integrity / Database/ Repository	Version Support / Linking Features	Model Based Design	IDE
DOORs	X	X		
Telelogic / Tau			X	X
Telelogic Tester	X	X		
MKS	X	X		
Matlab			X	X
dSPACE			X	X
Green Hills Multi				X
NI Labview			X	X

Table1-B

COTS Software Application Comparison Table 1 part B.

	Requirements & Design Validation	Compiler/ Debugger	Product Verification	Testing Extensibility
DOORs	X		X	X
Telelogic / Tau	X	X	X	
Telelogic Tester		X		X
MKS				
Matlab	X	X	X	
dSPACE	X	X	X	X
Green Hills Multi	X	X	X	
NI Labview	X		X	X

IX. PROBLEM STATEMENT

Without a standardized design process, especially in a large modular design paradigm that may require multiple individuals or multiple groups to work concurrently on different aspects of a project to be integrated at a later stage into the final project, tracking changes, collecting and evaluating data and especially phase and software integrating can be extremely difficult at best.

A company may hire a new employee who is required to use specific software tools and tool chains requiring very specific configurations on multiple levels. At the first level installation and configuration may be difficult due to some software tool combinations and depending on the clarity of installation/configuration instructions of the tools. There may be additional complexity based on the license management such as product key and directory installation requirements of each link in the software chain. Once this installation is complete at the top level, there may be further convolution with additional complexity of maintaining projects with links to multiple directories which are necessary for user defined scripts that direct software applications to share scripts with other applications. If not well documented in a large project, this task in itself may require an extensive 'debugging' process rivaling that of the software project itself. While it is not a difficult matter to develop and track such configurations on a single user computer, it can be particularly burdensome when a new developer is added to an existing project. Likewise when additional developers are added to a project, it will be imperative that their configurations appropriately mirror a master configuration in order to have the same utility in matters of design debug, simulation, and compilation as well as further forward moving development. With a process in place to ensure that all users begin with a common configuration, project management is simplified and may even be relegated to a software application for source integrity such as MKS or a communized resource repository tool such as Telelogic's DOORs requirements management application. Furthermore, in a modular design paradigm for instance with a large and complex project, pieces, modules, components can tend to get scattered over many resource repositories such as storage drives, individual computers, and software specific directories. Without strict process and project management this can result in dramatically increased complexity for integrating, especially in the absence of a top level topology.

X. PROPOSAL FOR INTEGRATED SOLUTION

The proposal for an integrated solution for a complete and universal software system application process must take an approach adapted to circumvent the necessity of third party component tools to enable a truly integrated development environment from start to finish. The present benefit of leaving the development process environment to utilize third party tools is that each component can be a highly specialized component designed specifically by a supplier that specializes explicitly in a given aspect of the overall development. For example Matlab is a specialty graphical programming language and IDE for model based design and code generation. The drawback to the present approach is that maintaining a replicable configuration of integrated third party utilities and extensions can be and generally is complex, non-intuitive, may not be portable to other platforms and may not be easily duplicated.

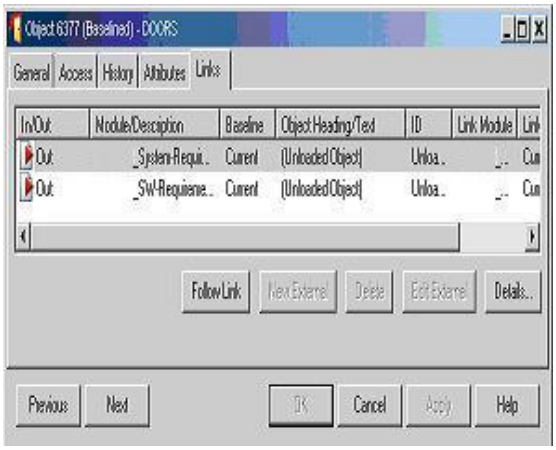

Figure 11: Screen shot of Link tab in the DOORs Object GUI editor.

An ideal integrated development environment will include functional elements such as DOORs ability to link multiple requirements documents to other documents on a line item by line item / function by function / parameter by parameter basis as indicated in figure 11 above. Using this same utility/functionality of the DOORs tools, individual functions/process/parameter/stages may be externally linked to a specific test requirements document that may be linked to other software applications or may be even be developed as a custom template/application as extension to the existing DOORs environment. DOORs do provide this facility for development of user defined extensions to the tool.

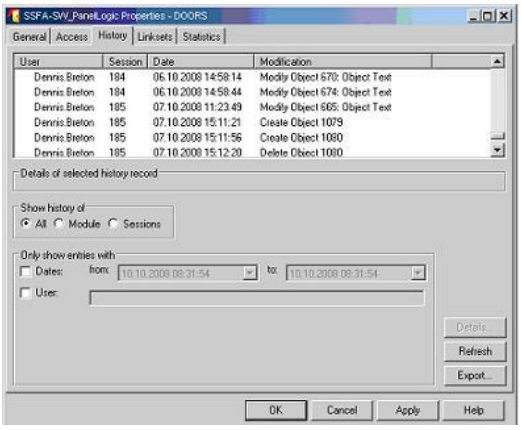

Figure 12: Screen shot of DOORs Properties GUI editor.

The History tab in the Property Editor GUI in Figure 12 above reports versioning and modification data on every given object in the repository. This DOORs feature may also receive future modifications that allow provision of test applications, configuration, and change management tracking that is routinely done in third party applications. In the same manner that a numerical hierarchy outline is kept which also allows inclusion of tables, spreadsheets or other graphic objects into the editor, DOORs may be modifiable to represent complicated test chain applications and environments. The property editor can and should be configured with drop down menus and URL hot links; however these links could also be sourced within the tool itself. Any GUI type application for importing/exporting test templates that can take advantage of the same DOORs-like functionalities for up-linking and down-linking to other documents would be suitable but should provide measures further for linking to other applications, or sub-environments including entire test environments that may include tool-chains such as Campbell Scientific PC9000 data logging / with CAN data communication combination evaluated through an NI Labview testing environment.

The ideal solution for an integrated solution is that the integrated environment features the best of all worlds, that is to say that the environment uses the optimal resource, design, test and capture/reporting tools without ever leaving the integrated development and with a minimal of integration effort. The benefit of this approach is that configurations can be embedded (contained) within the resource environment itself as opposed to keeping external records (or human memory) about configurations required between dev stages. The system can be portable from the aspect that a designer should be able to expect the ability to log in to the source repository and have at a single button press the entire development and its history loaded and built

to a unique environment. Real-time tracking and traceability can be implemented if such a feature is necessary or desired. It is a perfectly sound notion to base such a contextual framework on the already accepted theoretical design paradigm models such as the V-model.

A traditional or standard process such as the aforementioned V model should minimally consist of the following stages:

- Definition
- Design
- Development
- Integration
- Testing
- Prototype

"In classical design paradigms a performance measure may have been resolvable to a single (quantitative) parameter or set of outputs. A more complex system may require a multi-faceted approach to gather multiple performance measures including: Minimum-time problems, Terminal Control Problems, Minimum control-effort Problems, Tracking Problems, and Regulator Problems [16]".

While simulating real I/O is an effective testing approach for verifying and validating a process control system, there remains many missing components required to apply simulation techniques across the entire system development while remaining in a single integrated environment. This deficiency extends to the complete application lifecycle management in systems engineering and software development

In order to facilitate progress from traditional and classic scenarios to implement the developmental complexity of modern control systems the following tasks and/or resultant design stages require additional consideration.

- Composability - support for modular development and integration without side-effects
- Test automation
- Linkable test software
- Auto-publishing to requirements document repository/source integrity tool.
- Integration
- System Configuration management

Important test factors to ensure software verification and validation quality and reliability include accuracy and repeatability. Testing shall be done against customer specifications. Testing shall be done against failure. Additionally, a test requirements document may call for Statistical Process Control test and reporting in which an SPC value dictates a requisite number of times that a specific test is requested run on the same device thereby providing statistical analysis figures.

With these requisite developmental stages in mind, the remainder of this section outlines a template for an application that may have foundation in existing COTS or may take the best features from any of those environments to create a new software application that supports a complete beginning-to-end IDE with features that additionally include/address concurrent development, link-ability, integrated testing/reporting, source integrity, configuration management. The idea behind such a SW application is to replace the necessity for having to learn new proprietary coding languages when it is necessary to add enhanced utility or extend the application tools beyond its foreseeable use.

Fundamental rules:

- The application shall be maintainable in a SINGLE document repository or source integrity software tool such as Telelogic DOORS or MKS.
- The application shall be associated (linkable) to relative resource documents either through object linking (for traceability) in the case of DOORs or through logistical file management as in the case of MKS.
- The application shall have an appropriate menu option for linking utilities.
- The application shall have an appropriate menu option for directory/path maintenance for use in tools like MKS and MATHWORKS/dSPACE related tools.
- The application shall utilize a windows/OLE type of object embedding utility so that entire documents can be associated with an iconic link.
- The application shall have a project initialization utility upon creation that allows user input of known parameters and other data intended to be v&v by the user.
- The application shall maintain traces to software tool configuration/installation management documentation.
- The application shall maintain traces to a pre-configured or configurable directory or data path.
- The application shall allow installation, configuration, launch of software application and/or tool chains from within, and further it

should allow this feature by a right-click of a given test parameter to initialize either/and/or data acquisition functionality or already acquired result data.

- The application shall allow navigation through either/or: reference frame, callout, data tree, hierarchical structure.
- The application may allow the user to enable/create functionality through drop down menus and other graphical programming techniques.
- The application/project should maintain configuration data/repository of records through links and directory paths to locations on a computer or server, thereby also allowing the IDE to be portable across computers or workstations.

Telelogic/Quality Management tools seem the most likely developer to provide an overall integrated solution for verification and validation. In its present incarnation it can support most of the repository/linking/interfacing/reporting needs for a software development project. Telelogic DOORs repository/integrity functionality and powerful GUI editing interface provides an excellent example of what can and should be made available in an all-in-one tool. Note the tabbed editing capabilities in the GUI editor in figure 13 below.

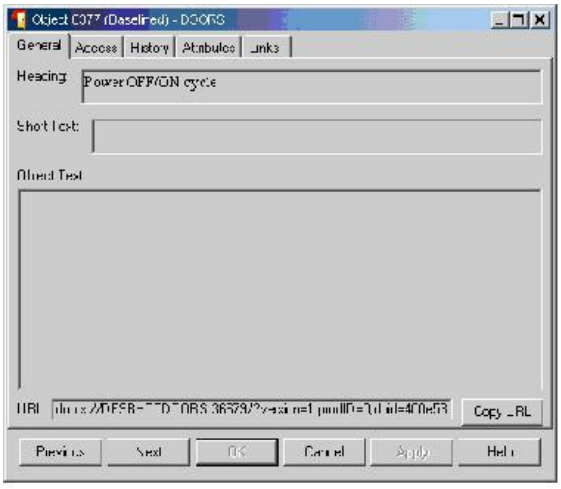

Figure 13: Screen shot of DOORs GUI editor.

Added features can be implemented in this same paradigm with the addition of new tabbed properties per each object or feature.

New windows can be added for the following types of features:

- input of I/O named parameters intended for testing, measuring, verification and validation.
- Integration/choice of data logging tools known in advance by engineer such as Campbell Sci/CAN input (such as P(eak)-CAN).
- choice of methods of data representation known in advance such as spreadsheet/tables/charts/graphs
- a properties-type of data association that allows modification/editing of parameters

In addition to the rules and features the project development process stages should be considered to extend/implement the additional following features and support the following tasks and functionalities:

- Identification of test parameters in the hardware or software requirement documents.
- Create external links from source documents in the basic outline or template of the test requirement document.
- Outline the test requirement document capabilities
  - Ability to choose testing software (perhaps even with a drop-down menu of pre-configured pre-installed software suites – additionally must have the ability to "Add" from the drop-down menu.)
  - Ability to identify and/or create tool chain of multiple software applications
  - Ability to link timeline/scheduling information/Gannt chart type of representation for testing environment.
- Multiple item choice window application that allows selection of parameters/functions, etc to be simultaneously chosen and directed to a master test-overview.
- From master overview, ability to group and direct individual parameter groupings to the software app to which they are best suited for testing.
- Test plan development complete with notes area for the lowest level of instruction.
- Collection of test data/integration of test tools
- Evaluation of test data, explanation of procedures and observation notes.
- Comparison of test data against original requirements.
- Population of an indicator in original test specification such as columns with drop down menus that allow quick linking and quick checking of such parameters as: test performed (y/n; date; type; SW), evaluation complete (y/n; date; by whom), presented for review to higher level engineers or managers; sign off (date, by whom).

Note the top level requirements document should link down level by level to test specification, test data, evaluation of data, which can in turn link to a hardware or software modular requirement. Benefits are illustrated by another prime example set by Telelogic's Statemate which is noted for its ability to use simulation software to "capture" and "Include" information in a test specification such that an "engineer can explore what-if scenarios to determine if the behavior and the interactions between system elements are correct. These scenarios can be captured and included in Test Plans which are later run on the embedded system to ensure that what gets built meets what was specified. This executable specification is also used to communicate with the customer or end user to confirm that the specification meets their requirements" [17], Statemate offers back-end high-to-mid level approach for validation via customer/user feedback mechanism, while MATHWORKS Simulink sets the leading example as an extensible environment by interfacing to add-on tools used for verification via multi-domain modeling. Either of these tools can provide a solid foundation for development of a complete integrated software development process tool.

XI. Conclusion

A successful product is expected to meet the customer's needs and expectations within a budget that has been agreed upon and is delivered in a timely manner. To meet marketable requirements and time and budget constraints, the developer of a product may cut corners on testing, verification and validation of a final product. Such omissions may result in reduced customer satisfaction and unanticipated product failures that are exponentially more costly to rectify once a product has evolved beyond the design stage. Rigorous testing, validation and verification are important final steps to making a delivery that meets the client needs. Not only does this impress and reinforce customer satisfaction but a thorough testing, verification and validation process can also save on the long term maintenance phase of the project. A major step in enhancing quality assurance will be to provide the ability to automate and link an entire testing, validation and verification process with extended design and reporting domains to provide linked processes at all levels for example bridging the gap of a V development process flow. A linking, unifying process helps to facilitate not only concurrent test specification development but also early stage testing in which discovered areas of improvement can be exponentially less costly than if required at later stages of development. Testing at every level becomes possible including Model in the Loop (MIL), Software in the Loop (SIL), Processor in the Loop (PIL) and Hardware in the Loop (HIL) stages. By linking at initial and terminal stages, a testing paradigm can be implemented that insures testing against customer specifications as well as against product failure and provides a traceable link to any stage on the product development and lifecycle. By encapsulating an entire development process within a test-friendly context, an adopted process may be enhanced for parallel or concurrent implementation that allows for wider access (multiple developers), greater management capabilities (a project with an aggressive deadline may even be projected into a real-time environment such that a server based repository can facilitate real-time updating), control (checks and balances against specifications and requirements at every level), and finally correctness and repeatability of the entire development process.

References

[1] V-Model (Software Development) http://en.wikipedia.org/wiki/V-Model_(software_development)

[2] "Flow Diagram of Product-Process Development" Retooling Manufacturing: Bridging Design, Materials, and Production Committee on Bridging Design and Manufacturing, National Research Council ISBN: 0-309-09266-3, 123 pages, 8 1/2 x 11, (2004). Page 15. http://www.nap.edu/catalog/11049.html

[3] Reqtify – A Graphical Requirements Traceability Environment. http://www.chiastek.com/products/reqtify.html

[4] "Telelogic Lifecycle Solutions: Getting Started with DOORS" doors_getting_started.pdf@ http://support.telelogic.com

[5] "External links and DOORS URLs" external_links_and_DOORS_URLs.pdf @ http://support.telelogic.com

[6] Donald E. Kirk, Optimal Control Theory – An Introduction, First Dover Addition, 2004, Dover Publications, Inc.

[7] SPACE Catalog 2014, page 185. Catalog 2014, dSPACE, Technologiepark 25, 33100 Paderborn, Germany: http://www.dSPACE.de

[8] "Professional quality management and assessment when developing software for embedded systems" page 1. Quality Management and Assessment: Renate Stuecka, Telelogic Deutschland GmbH, Otto-Brenner-Strasse 247, 33604 Bielefeld, Germany, Tel ++49 (0) 521-14 503-254, http://download.telelogic.com/download/paper/qualitymanagement.pdf

[9] "Code Verification and Run-Time Error Detection Through Abstract Interpretation, A Solution to Today's Embedded Software Testing Challenges", Polyspace_white_paper_abstract_interpretation.pdf. @: www.mathworks.com/polyspace

[10] Gary W. Johnson, Richard Jennings, LabVIEW Graphical Programming Fourth Addition, 2006, McGraw Hill

[11] "Automated Testing In Software Development: Standards Drive Improved Quality" Version 2, August 2008, Irv Badr, Renate Stuecka, Telelogic, Tester_WP_Automated_Testing_In_Software_Development_FINAL_080804.pdf @: www.telelogic.com

[12] "Early Verification and Validation in Model-Based Design" presented by Amory Wakefield at Mathworks Automotive [Interactive Web] Conference. Tuesday, October 28, 2008.

[13] Simulink® HDL Coder™ 1, User's Guide (slhdlcoder_ug)@: www.mathworks.com

[14] Micro Max Technology @: http://www.mrmx.com/products

[15] "HP Quality Center: Combine requirement, test and defect management into a single quality platform." HP Quality Center.pdf @ www.hp.com/go/software

[16] Vahid/Givargis, Embedded System Design: A Unified Hardware/Software Introduction: Chapter 11: Design Technology