2013 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
SYSTEMS ENGINEERING (SE) MINI-SYMPOSIUM
AUGUST 21-22, 2013 – TROY, MICHIGAN

REDUCE PROJECT SCHEDULES AND INCREASE QUALITY USING MODEL DRIVEN DEVELOPMENT FOR DESIGN, VERIFICATION AND TEST

John Vargas
Systems Architect
Mentor Graphics
Philadelphia, PA

ABSTRACT

As contracts move from cost plus to fixed deliverables, total project cost and reducing schedules become more important. This paper will show how Model Driven Development can address common challenges in the system design, verification & testing of complex systems and systems of systems. Project success requires that hardware, software, and test teams fluently integrate application software, controlling firmware, analog and digital hardware, and mechanical components, which often proves to be costly in terms of time, money, and engineering resources. Model Driven Development and virtual prototyping using a tools flow emphasizing requirements tracing, UML / SysML system modeling, and linking to functional FPGA, IC, PCB and cabling domains supports system engineering teams along with software, digital hardware, analog hardware, system interconnect algorithm development, hardware / software co-simulation, and virtual system integration. This paper covers such solutions that reduce project schedule while improving product release quality.

INTRODUCTION

Sixty-seven percent of government weapons contracts were found to be behind schedule, with 30% of them more than two years behind schedule in a Government Accounting Office Report [1], and an Aberdeen Group study found that even well-run companies struggle in their development programs because overwhelming complexity is rendering previous best practice process outdated and ineffective [2].

Compounding the challenges are the recent economic conditions affecting total spend and new and existing funded programs with a focus on cost controls and a trend towards increasing competition in DOD and DHS programs with a move away from award-fee and fixed-fee contracts with a 11.9% 3-year CAGR growth in fixed-price contracts and 13.4% growth in competed contracts with multiple offers over the same period [3,4].

At the same time, product and program complexity has continued to increase exponentially over the last decades [5]; therefore the stakes are very high (and growing). A single miscalculation, miscommunication, or misunderstanding in program development can lead to cost overruns, schedule delays, reliability problems, or even field failures. As the DARPA research indicates, industries such as Integrated Circuit design have managed to keep the cost in check even as the complexity increased while other industries such as Aerospace and Defense have had cost growth increases of 8%-12% per year over the last 40 years. This paper will detail some of the ways that methods from the IC design industry, such as Model Driven Development, can address common challenges in the system design, verification & testing of complex systems and systems of systems, reducing project schedules and improving quality.

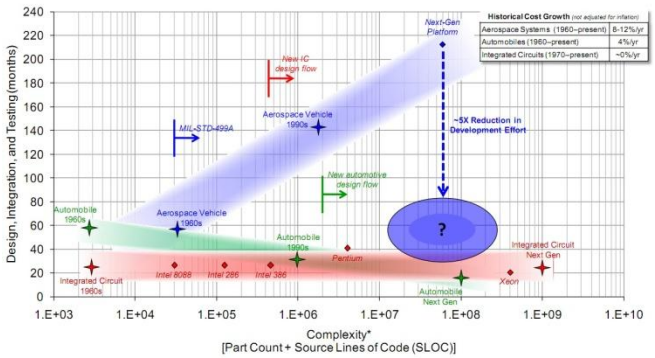

Figure 1: Historical Complexity and Cost Trends [5]

TRADITIONAL METHODS

The traditional system design process begins with a requirements-driven system definition and proceeds into a preliminary design or architecture phase, which feeds into designing and implementing the components that make up

the system (chips, printed circuit boards, firmware, algorithms, application software, and mechanical elements).

Once these are implemented, they are verified, proceeding in succession from the component to each subsystem, grouped subsystems, and finally the "serial #0" candidate production system, which is verified against the system requirements and validated against the top-level customer and marketing requirements. This is typically visualized as the "V" diagram.

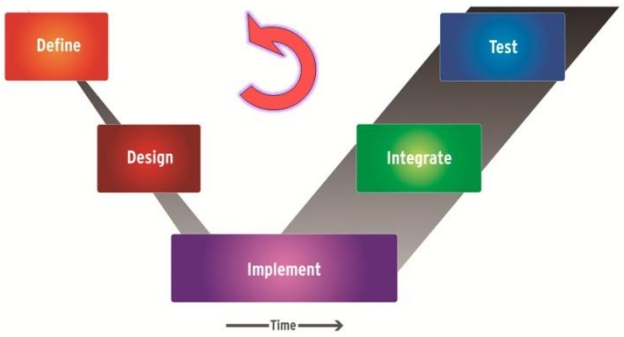

Figure 2 : Traditional V diagram flow

This is an inherently sequential process where verification and validation is not complete until the end, as the "time" axis indicates (see figure 2). Since for DOD programs and other large deliverables the customer almost always requires "serial #0" physical system verification and validation, most companies have tried to implement the V process with the goal of minimizing the time from design to physical system and eliminating any steps not essential to the direct manufacture of the physical system. Writing requirements down in a document based on previous experience and expert knowledge is viewed as faster than creating a behavioral model of the requirements, executing them, and performing simulations and dynamic analysis.

The result is a "paper-driven" front end process. The requirements and system architectures are done on "paper" elements such as documents, static diagrams, and requirement lists which, of course, are electronically created, stored, indexed, and linked, but are not qualitatively different from what was done many decades ago with a pencil, ruler, typewriter, and paper. It is likely that simulation and analysis is incorporated at some point, but usually within a specific domain and focused on aspects essential for implementation or regulatory compliance.

A complete analysis and virtual simulation of the entire complex system is not undertaken as it is viewed as too time consuming or impractical; instead efforts are focused on getting to an early physical prototype quickly. With this approach, the first real visibility of complex system integration problems is reserved for the very final stage of the process – testing the physical, integrated system.

This traditional "paper–based" approach to implementing the V process may have worked in the past, but as sources listed in the introduction indicate, it is no longer working well. With today's complex system developments of integrated, inter-related mechatronic elements interacting in non-obvious ways, the time to verify the physical integrated system can quickly destroy the original schedule if any re-design is required and repeated candidate serial #0 units are produced and run through the verification cycle. The more complex the system, the less likely this approach will result in a design developed within schedule and budget.

MODEL DRIVEN DEVELOPMENT

Imagine instead a process where concepts and requirements from all disciplines could be easily tested from the very start using software modeling techniques and, once proven, would pass from step to step in the process. Teams of experts would work concurrently throughout the technical, system, and process levels, with individual designers concentrating on their specific tasks using the best available tools for their individual task while able to seamlessly fit into a complete virtual system environment. Verification would proceed in parallel, with design occurring at first completely virtually; these same virtual tests then would be re-used when the system is physically verified. Monitoring of compliance with regulatory requirements and customer specifications would be integral to every step. Ultimately all the pieces would be efficiently integrated into a system that works the first time and does not need physical re-designs. This is the vision that gave rise to the innovative Model Driven Development (MDD) tools and processes that are fast gaining favor among system developers. MDD lays the groundwork for an integrated design flow that addresses the complexity challenge once and for all.

Model Driven Development Approach

So how does the MDD approach help move a development program from sequential design and verification to a concurrent process? It does it by replacing "paper-based" static documents with a dynamic data-centered approach. In an MDD environment, the information about a program is the model data. The whole process is driven by the data, which is always synchronized with the current stage of development. This gives developers and management a much clearer and realistic view of the program's status and is an easy way to assess the impact of changes, which has both business and regulatory benefits.

Requirements are a critical aspect of program data. Many companies understand the importance of requirements, but, too often, requirements are kept in documents or databases that exist outside the everyday world of the designers. When changes occur in a requirement, this may or may not be

Reduce project schedules and increase quality using Model Driven Development for design, verification & test.

communicated effectively to the design or verification teams. This is a common struggle and a cause of many problems often not found until final testing.

In an MDD approach, requirements are connected into the models and their verification suites. Changes to requirements necessitate changes to the models, and this is highlighted immediately in an automated way. The same can be said for interface controls. Instead of existing outside the development environment in some sort of document, in a modeling environment these become active properties of the design itself. Artifacts of the process are another aspect of a program's documentation. Instead of reports being created by hand (and therefore immediately out of date), the pertinent data from an MDD environment can at any time be viewed, audited, or even automatically generated into a report.

The shift with an MDD approach is to a focus on virtual validation and verification (V&V) at the front of the 'V' instead of physical V&V at the end of the 'V.' Physical V&V does not disappear but is much reduced and streamlined due to the extensive virtual V&V performed earlier and the testbenches and artifacts from the virtual V&V leveraged for the physical V&V.

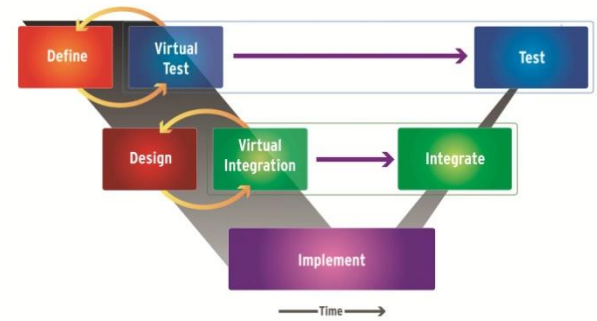

Figure 3: Model Driven Development Approach

Define Phase → Concept Validation

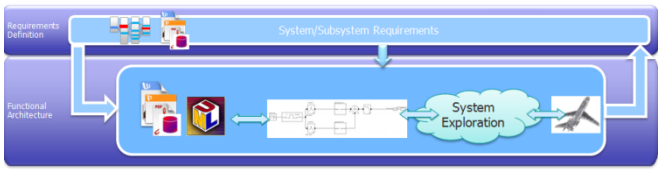

Figure 4: Shift from Sequential Define → Design to Concept Validation

At the very earliest stages of design, the initial idea and its requirements can be captured in a high-level conceptual model, using graphical languages such as UML or SysML. Customer and Marketing requirements remain in a requirements database, but these models link directly to them and start to implement the engineering system-level requirements as models instead of another large list of derived requirements. This blends the initial idea, requirements capture, and conceptual design stages into one new "concept validation" stage. This initial conceptual model can be dynamically executed and tested against a comprehensive suite of verification tests and ultimately help clarify and validate the idea, requirements, and concept. This conceptual model, which initially defines only the function's required behavior, can then be broken down further into a closer representation of the real design at the next level.

At this phase, the key difference with the traditional approach is that instead of the paper documents containing the whole design specification, a federated set of data contains the specification and it is not duplicated so where there are models--"The model is the requirement." It does not mean traditional documentation disappears or is no longer needed, instead, possible executable models are used rather than English language specification as the source of truth for the system design. Where is it possible to do this? It used to be that it was practical mostly in the software and digital logic domains, which left a whole pile of paper documents for the rest of the system. But, with modern MDD approaches, this is no longer the case, and, at the functional level, the behaviors can be validated and verified while the implementation decision on hardware vs. software is postponed.

At the functional level of the model hierarchy, system-level engineers create executable functions with measurable behaviors that correspond to functional specifications for the design as derived from the system requirements. Interactions and tradeoffs between specifications are explored virtually using functional-level models. Theoretical behaviors are modeled at this level without concern about whether a function will be implemented in hardware or software or which specific components will be used in the design. At this level the models constitute a mix of further broken down UML or SysML combined with algorithmic or physics-based continuous time models in high level behavioral VHDL-AMS or similar languages, along with rough 3D models of the mechanical aspects of the design.

It is important to note that the reality of very complex system designs means that no single language or integrated toolset can capture the entire system design from 3D physical aspects to algorithmic models; it requires an interconnected and cooperating set of domain-appropriate tools.

In order to deliver on the MDD promise, each of these tools needs to be able to support three core functions: (a) operate on executable models that can be fully verified and tested, (b) cooperate in a federated system of appropriate information exchange to enable requirements tracing,

workflow and project management linkages, and (c) integrate with a co-simulation environment.

With this MDD functional architecture complete and having gone through several rounds of validation against use cases, concept of operations scenarios, marketing requirements, and customer reviews, then and only then are these new executable requirements really ready to pass down to the next design phases.

It is highly likely that through this process the requirements have changed significantly and both the customer (whether internal or external) and the engineering team have a much richer understanding of the desired operation of the system. Waiting for a physical prototype to have these revisions and reviews would have certainly cost a lot of time and money and would have been wasteful as effort would have been expended designing and fabricating something that was not fit for the intended purpose.

Design Phase → Virtual Design Verification

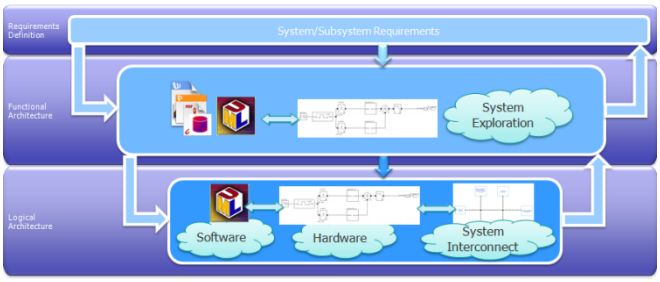

Figure 5: Shift from sequential Define → Design to Iterative Early Validation & Verification

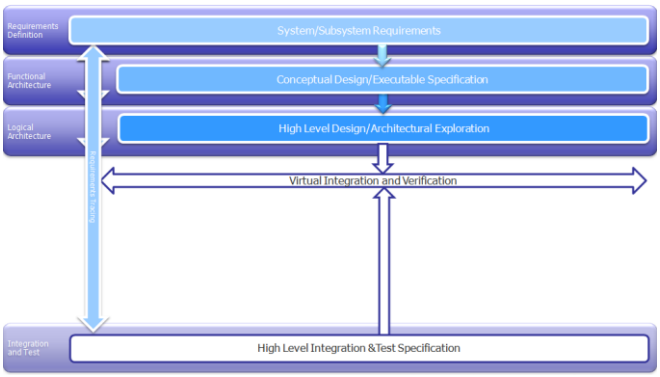

Figure 6: Shift from Sequential Define → Design to Virtual Design Verification

At the architectural or logical level, teams of system architects along with domain experts use model simulations to explore options for implementing the system architecture. Each team can operate in parallel, exploring different aspects of the architecture while feeding into and testing against a cohesive complete system model. Different teams start to create more detailed models that are more domain specific (mechanical, continuous time, discrete, processor scheduling, algorithmic, etc.) as appropriate but remain able to verify them against a cohesive virtual view of the system that is revision controlled and can be traced to the earlier functional level requirements implemented as functional models.

Decisions are then made about which functions will be implemented in embedded software, which in electronic hardware, which functions will communicate virtually via network layers, which will be implemented with a discrete interconnect, and which using other physical disciplines.

System engineers test the interfaces between different parts of the design virtually before the design is fully implemented. These tested interfaces are passed down to the implementation level as requirements that each domain must adhere to or request a review by the system engineers. This reduces errors and allows integration issues to be identified and addressed early in the design process.

Implementation Phase and Virtual Design Verification

During the first phase of the implementation or physical level, each domain-specific engineering team drills down into critical areas of functionality, while dealing with the less critical areas more abstractly until the design is closer to completion.

In some cases, parts of the implementation such as software, digital hardware, and cabling designs can be automatically synthesized from the architectural models into a lower-level implementation (e.g., C, VHDL, wires, etc.) of the actual design, reducing the implementation work to a simple verification task for these cases. For other parts of the implementation, such as analog hardware and mechanical design, there are currently no practical synthesis methods and more manual effort is required.

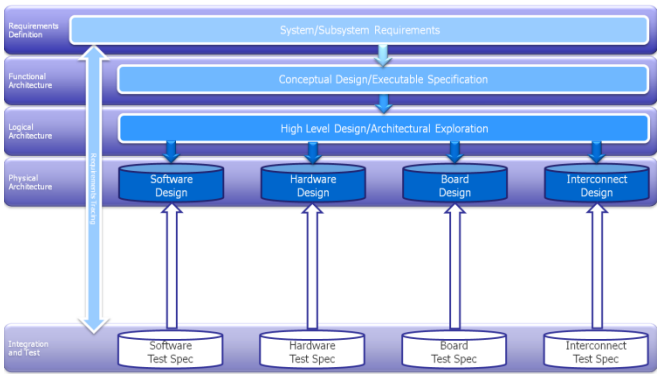

Figure 7: Implementation First Phase – Independent-Domain Verification

Reduce project schedules and increase quality using Model Driven Development for design, verification & test.

In this first phase, design engineers perform detailed simulation and verification of their own components and verify their subsystems are operating correctly before going onto the next phase. Still there is no need for physical prototypes; all these tasks can be performed with virtual representations of the system in MDD Cad-type tools.

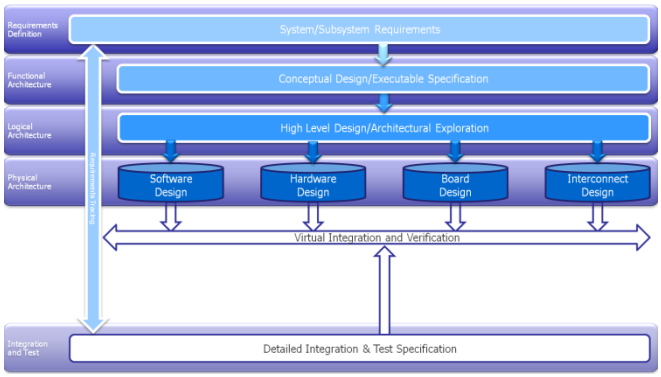

Figure 8: Implementation Second Phase – Virtual Integration

In a second phase of the implementation, a virtual integration is performed and the domain engineers then feed their implementation designs into another more detailed view of the complete virtual system, which allows verification engineers to test the interfaces between different parts of the design virtually and ensure that the design engineers are adhering to the required interfaces specified by the system engineers.

It is not usually feasible to simulate the complete, final implementation-level design of the entire system in a reasonable time; this causes many groups to give up completely and head straight for the physical prototype. Others use this fact as an excuse to not perform very much early system-level modeling ahead of time with a sense of despair that in the end it is intractable, too complex, and just better to get the "real" prototype and not "waste" any more time with the virtual verification since it won't be "accurate enough" or the inability to "model reality close enough." However, it is really not necessary to simulate every last transistor, mechanical spring, and line of code of the entire system *at the same time.*

The goals of virtual system integration and verification can be achieved by following a process from the beginning and creating executable requirements that bound the behavior of the sub-systems. The naysayers miss the point of proceeding through a methodical and sequential system validation, verification, and virtual integration at each step of the process. After having performed complete virtual system verifications at each prior level, all that remains to verify at the physical level is that each of the component's detailed implementation continues to operate within the bounds of the previous higher level model requirements correctly while it is connected to the rest of the system. If no prior architectural-level system models and verification steps were performed, then complete system virtual verification at the physical level is not tractable. However, if an MDD approach was followed from the start, it is very possible and tractable.

Sometimes called a checkerboard approach, the architectural-level models of the complete system are used, replacing one or a few components at a time at the detailed level.

$$DetailedVirtualSystemVerification \approx \sum_{1}^{n}\left[\begin{pmatrix} VirtualSystemArchitectureLevel \\ -architecture_{component_n} \\ +detailed\_component_n \end{pmatrix}\right]$$

Figure 9: Tractable Virtual System Verification & Integration

This process is in use every day with great success at IC design groups that design the world's largest and most complex semiconductors, such as CPUs, GPUs and massive systems on a chip. Even with the most advanced simulators and emulators, it is intractable to verify the entire IC design at the most detailed semiconductor level while running the application-level software such as Windows or Linux.

But do they then just give up and wait for first silicon to test the system out? Due to market pressures they cannot afford the cost of a mistake or the time to wait so they use the architectural-level models to verify the system operation and then rigorously compare the behavior of each architectural model to the behavior of the isolated block-level, detailed implementation simulation. After success, then they plug in one or a few critical detailed components into the system verification at a time for a final system integration check.

Physical Construction, Integration & Test

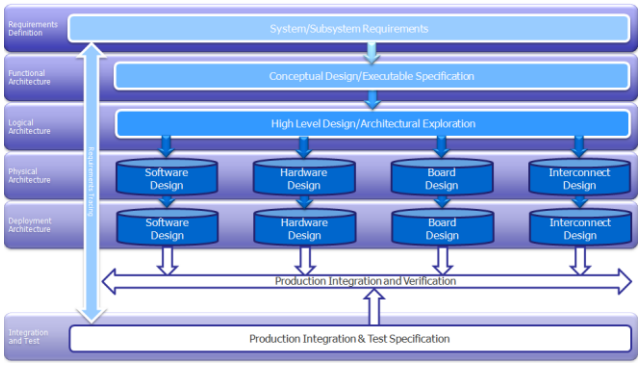

Figure 10: Physical V&V

Reduce project schedules and increase quality using Model Driven Development for design, verification & test.

The traditional physical construction, integration and test processes remain part of the MDD flow; they are just reduced in time and complexity. By the time the physical system is built, the system and every sub-system have been tested separately and together in a virtual environment. This means that the verification & validation group has been involved from the beginning and has already created the tests for the virtual phase; so not only are they created, but appropriate test access has been built into the system because of this. Therefore, when the subsystems start to arrive to be tested, most of the infrastructure is in place and physical V&V proceeds at an accelerated pace with very few surprises.

This happens because, at each step of this process, the original system requirements are traced to measurable attributes of the design and verified. This is supported by the virtual system integration platform, which combines multiple models and levels of abstraction and provides a way to exercise the behavior of a design at a functional, architectural, or fully-implemented level of abstraction or a combination of these levels. While following the process of virtual verification, the verification group is also designing and developing their final physical verification tests against the virtual platform. This test set can run on the system model at any time during the development. Then, when run at the final stages of physical system integration, this "final" test stage (which so often in traditional flows is the beginning of a very long process of debug) becomes merely a sanity check that the system was built correctly.

ORGANIZATIONAL EFFECT & IMPACT

Collaborative MDD

It is important to note that the MDD flow does not require all the participants to use the same tool or the same modeling language. The MDD flow allows the experts to work independently in their own domains, using their own languages (e.g., UML, SysML, Verilog, VHDL, VHDL-AMS, C, SystemC, C++, Java, m-script, etc.) and tools, as they would prefer to do. But, the models they produce can be integrated into a broader system architecture model and executed in any simulator that supports all the chosen standards concurrently or by use of a simulation backplane that connects multiple domain-specific simulators together into a live, concurrently executing "meta simulator."

In a similar way, appropriate information exchange between the tools for requirements tracing and workflow via standards (such as Open Standards for Life Cycle collaboration or OSLC) enable these different tools to communicate requirements, test cases, tasks, and product tree views with other software designed for project managers to track status and regulatory and safety officers to verify traceability and compliance. This creates a virtual collaboration environment where the team is collaborating across divisions and disciplines without everyone being forced to use the same exact interface or software tool.

This same sort of "virtual" collaboration can extend from integrators to suppliers to contractors. Models and OSLC exchange become the mechanism to collaborate and verify both function and progress at any stage of development. Collaborative MDD will not literally force systems, mechanical, electrical, electronic, and software engineers to sit down in the same room, talk, and jointly work on a program. Instead, it creates a virtual environment that automates this collaboration, transparently.

Schedule Re-Alignment

With all these benefits, what is the catch? A significant barrier for adoption of MDD can be the organization's inertia and prior history with projects and schedules. Past project timelines and schedules feed the project management office or management expectations for new schedules. Schedules that are back-end loaded have been created for so long and with a feverish determination to get to Critical Design Review (CDR) and first prototype as soon as possible that there is typically no allocation for system modeling and virtual verification in the early part of the schedule.

Adopting MDD requires that project managers and executive management be willing to embrace new methods and commit to a change in schedule alignment. While it may seem counter-intuitive, adding time to the schedule up front and delaying the creation of the first physical prototypes of a system and implementing an MDD process on top of the traditional V actually reduces the total real-world time from start to customer-accepted product and increases release quality. Months are shaved off the backend test development and test processes, and, due to the significantly higher probability of the first-pass success, customer acceptance is achieved earlier.

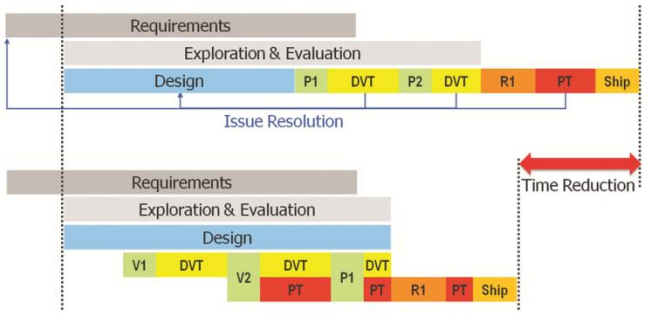

Figure 11: Cost and Schedule Reduction with MDD

Reduce project schedules and increase quality using Model Driven Development for design, verification & test.

CONCLUSION

Federated Model Driven Development is the enabler of virtual system verification, validation and integration with tools that (a) operate on executable models that can be fully verified and tested, (b) cooperate in a federated system of appropriate information exchange (OSLC) to enable requirements tracing, workflow and project management linkages, and (c) integrate with a co-simulation environment.

Mentor Graphics has a suite of tools for this MDD approach, covering software, digital and analog hardware, platform architecture, wire and harness, and supports links into existing solutions such as The Mathworks Matlab, National Instrument's LabVIEW, and the IBM Doors, Rational, and Jazz platforms. To read more about the Mentor tools that enable an MDD flow, please visit mentor.com/sm.

MDD in this manner integrates models from different domains into a data-centered, collaborative environment. The model at each stage of refinement is truly a virtual prototype of the end system. Verifying this model early and often throughout the process catches issues early (when they are easy to fix), before they jeopardize project schedules, budgets, or worst case, lives.

Using new design methodologies, such as model-driven development with virtual prototyping and federated information exchange, can help ensure successful product development on schedule, under budget and with increased quality. The world is truly connected, and adopting collaborative processes that leverage this as opposed to isolated ones that ignore this fact is essential for complex systems programs. MDD technology gives the system integrator an effective platform to communicate the overall system requirements and individual component specifications. It can also tie project management into development and automate mundane and time-consuming tasks, so designers can spend their time doing what they do best -- designing.

REFERENCES

[1] GAO Report to Congressional Committees (GAO-08-467SP), "Defense Acquisitions: Assessments of Selected Weapon Programs", March 2008.

[2] System Design: New Product Development for Mechachtronics, Aberdeen Group, January 2008.

[3] Defense Industrial Initiatives Current Issues: Cost-Plus Contracts, Center for Strategic & International Studies, csis.org, October, 2008

[4] DHS contract Spending and Supporting Industrial Base, 2004-2011. Center for Strategic & International Studies, csis.org, Dec 2012

[5] Adaptive Vehicle Make, DARPA, http://www.darpa.mil/WorkArea/DownloadAsset.aspx?id=2147484350