MODERNIZING SYSTEM DEVELOPMENT: REQUIREMENTS-BASED, MODEL-DRIVEN DESIGN, IMPLEMENTATION, AND TEST

Bill Chown and Michelle Lange, System Level Engineering Division, Mentor Graphics Corp.

Abstract

With no end in sight, mechatronic system complexity continues to increase, worsening the challenges that have already plagued systems designers for years. Incomplete or volatile requirements, poorly specified and managed interfaces, integration testing finding problems at the very last stages of a program, development and analysis in disconnected domains, multi-domain expertise in short supply, and coordinating work and status across multi-level supply chains – these are all problems commonly discussed among systems developers. These challenges are further compounded in highly regulated domains, such as aerospace, automotive, and medical, where the end system must also be proven safe and conform to pertinent policy.

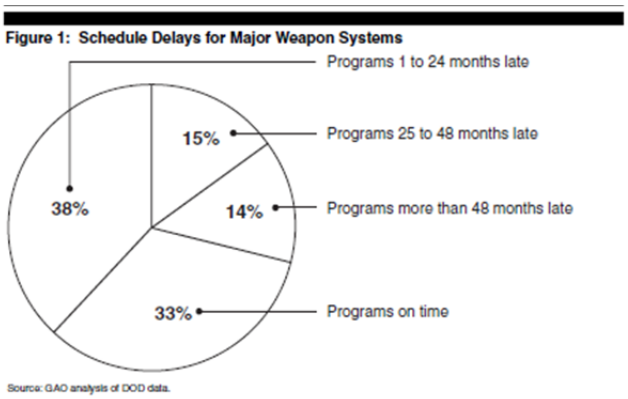
In this paper, we discuss the root causes of system design challenges, and how the industry is responding with more effective design methods, tools, and collaboration mechanisms.

Introduction

Across every industry, demanding customers are expecting more intelligence and functionality at less cost. The result is increasingly complex mechatronic systems, where electronics and software control the mechanical aspects. A 2008 Aberdeen Group study [1] describes this as a "growing trend to blend mechanical, electro-mechanics, digital control systems, and electronic design elements into an integrated system." The study sites a number of common problems in developing systems of such complexity, and concludes that "How effectively companies are able to address these challenges carries with it a significant impact on a company's ability to meet key targets that drive product profitability."

Very similar words were expressed by Dr. Donald T. Ward, Systems Architecture and Virtual Integration (SAVI) Program Manager during a presentation at the 2011 *FAA National Conference for Software and Airborne Electronic Hardware* [2], where he described the "distinctly unpleasant experience in several recent projects where problems were found too late." The SAVI organization, which is a program under the Aerospace Vehicle Systems Institute (AVSI) [3], is focused on establishing a new way of specifying and integrating increasingly complex aerospace systems to reduce cost and schedule, while increasing quality, safety and performance. At the 2011 *Safe & Secure Systems & Software Symposium*, Dr. Ward gave a similar presentation on behalf of SAVI [4], citing schedule delays and rising costs as disturbing trends leading to a situation where complex system designs – if continuing to be developed as they are today – will simply to be too expensive to be practical.

As another data point, a 2008 United States Government Accountability Office (GAO) report on complex weapons systems [5] determined that the DOD is not receiving expected returns on its large investment in weapon systems and that cost and schedule performance of recent programs is getting worse. In terms of delays, the report summarizes the situation as follows:



Whether weapons systems, aircraft, medical devices, or similar, the stakes are very high (and growing) in these expensive programs. A single miscalculation, miscommunication, or misunderstanding in such complex programs can lead to cost overruns, schedule delays, reliability problems, or worse. Every engineer and manager worries about risks like these and takes measures to avoid them.

The problem is that these sorts of problems are nearly inevitable in today's complex systems. Even well-run companies can struggle in their development programs, simply because overwhelming complexity is rendering previously "best practice" processes outdated and ineffective.

The SAVI organization, its member companies, and many others in industries that are creating complex systems understand that change of the development process is needed to ensure safe and cost-effective systems of the future. But in order to enact that change, it is essential to have a thorough understanding of the underlying problems that are driving the complexities and

corresponding challenges in the systems arena. These key challenges are described in the following subsections.

Multi-Domain Development

Expertise in the each of the various development domains is a precious commodity. Multi-domain expertise is even more rare and valuable. Capturing that knowledge and utilizing it not only in each discrete domain but also across domains is essential. A common problem in today's programs is that development teams work as isolated islands, each focusing on their own unique piece or aspect of the system, but few if any people really focused on the whole. The organization, tools, and process are often completely separated across chip, board, software, controls, power systems, mechanical design, etc.

Bringing this domain expertise together as appropriate throughout the development is imperative for project success. Allowing disconnected development and waiting until the latest program stages to perform systems integration and connect all the pieces is a high risk situation, yet one all too common in today's product realization process. This is a key cause of problems being found too late to be economically feasible to fix.

Requirements and Interface Management

Complex systems have several tiers of requirements, derived from stakeholder needs, starting at the System of Systems level and culminating with component specifications. Managing/tracing requirements through the various stages of design evolution, levels of hierarchy, and supply chain boundaries is no easy task. Far too often this aspect of design is document-based.

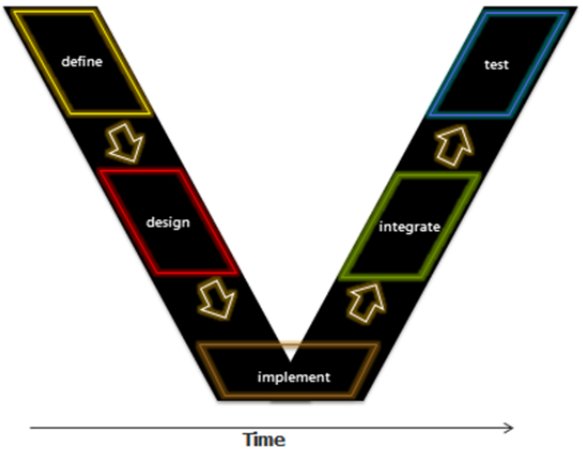
Many companies do have requirements capture systems, but the common problem with these systems is that the process of "requirements management" is limited to working on a database and is thus separate from the actual development activities. A database stores requirements and an expert on the database tool (or requirements management) is usually in charge of keeping this data in order. Also, each of the isolated development domains may use their own tools and processes for managing the requirements for their domains. Even when a tool is used at the top level, or within various groups, communicating down or across the project still often relies too heavily on documents, which far too often are out of date or out of reach.

A more modern approach utilizes strategies and tools that link the requirements databases to the development environment and activities – across all development domains, making requirements-driven design (including management and traceability) the responsibility of development, safety, verification, and test engineers, in other words, the people actually doing the development work.

The problems with interface management are quite similar to the problems of managing requirements. In fact, a more integrated approach to management of requirements can help with interface management as well. For example, one approach is labeling requirements as "interface" requirements and identifying the interface stakeholders (i.e. people, groups, organizations, etc.) who need to be involved with changes. These interface requirements are linked with the systems engineering source database, which with appropriate tools can then be propagated with verifiable digital continuity down the design and verification pathways of hardware, software, and test organizations.

The Development Process Itself

The V-Diagram is probably one of the most common ways to express a system design process. An example is presented here.



The wording of the individual stages may vary somewhat, but ultimately the process begins by some sort of system definition and proceeds into a preliminary design or architecture phase, which feeds into implementation of the items that make up the system (namely, the components, chips, printed circuit boards, software, controls, and mechanical parts). Once the pieces are implemented, they are then integrated and the whole system is tested.

The biggest problem with a process such as this is that it is inherently sequential. While there are certainly iterations within and between the processes, in general, the flow is from left to right, as the "time" axis indicates. This means that in the most traditional flows, proof that the design works is reserved for the very final stage of the process – testing the physical, integrated system. While this model may have served as a guide for the development

of simple systems, with today's complex system developments, following a sequential process and not verifying the system operation until the end of the process is a recipe for disaster. The more complex the system, the less likely this approach will result in a working design, developed within schedule and budget. In fact, the comments and data presented earlier corroborate this.

Another factor concerning the V-Diagram process is the disconnect of the design development itself from the lifecycle process and data. The use of Product Lifecycle Management (PLM) solutions, which provide enterprise level business solutions for requirements management, product data management, configuration management, program and project management, is on the rise. And yet adoption of PLM solutions does not always lead to the seamless, collaborative development environment that is needed to bring multi-disciplinary teams and development data together. Critical gaps still exist.

The process itself must be improved with more concurrency, earlier validation and verification, development integration, and process collaboration. The resulting V-Diagram representing such improved processes must be completely re-evaluated and modernized.

Increased Regulation

System complexity and the associated risks and concerns have been the driving forces behind new or updated regulatory requirements for system design in several domains. In aeronautics, the newly published ARP 4754A mandates systems design companies take a more structured and rigorous approach to system development. Requirements-driven design and validation/verification throughout the process are two key elements of the new ARP 4754A document.

In automotive, the all-encompassing ISO 26262 process governs development of all safety-related electronics in the entire vehicle. Here too, requirements management and verification play a key role, with an added emphasis on the safety analysis aspects of the program.

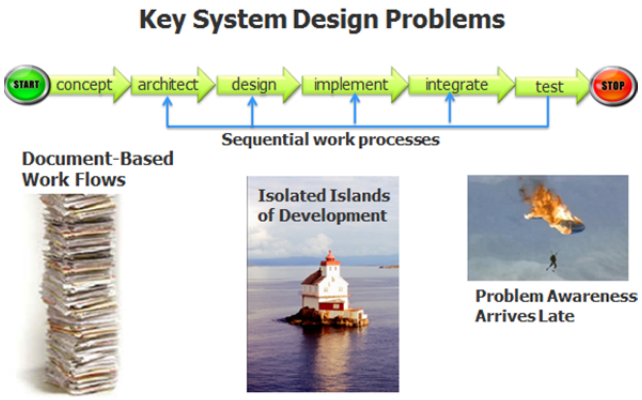
The medical field is very similar, with FDA regulations originating from 21 CFR 820.30 Design Controls and further supported by ISO 13485 and ISO 14971. These regulations govern the complete development process including requirements management, design, verification and validation and safety risk management.

Other industries as well, such as Nuclear, Industrial, Transportation, Space, and Military have similar regulation and/or requirements to ensure the safety or mission-critical aspects of their systems. After all, it is one thing to be late or over budget, but it is an entirely different thing to produce a system with flaws that ends up killing people. The regulators of course only care about the latter. The systems companies producing these devices must care about both.

Summary of the Challenges

In their most basic form, the challenges of system design all really stem from three basic problems. 1) development processes that are sequential as opposed to concurrent, 2) processes that are driven by static documents as opposed to live data, and 3) development work that is done in isolation as opposed to in collaboration.

The symptom of any or all of these problems is that major problems are found much too late in the process to economically solve.



Failing to recognize and address these challenges can have an adverse effect on the schedule and cost of a system development program, and this can have a serious impact on a company's profitability or even ongoing viability. The convers is also true. Addressing one or more of these challenges, by modernizing the design process, results in notable benefits.

The Remedy: A Collaborative Model-Driven Development Process

A temptation is to look for a single solution to what in fact is a complex combination of challenges, and can only be effectively addressed with an integrated combination of capabilities, working together. In other words, no single bullet can eliminate all concerns. However, a modernized approach to system design encompasses nearly all aspects of the flow, and serves to address most of the key concerns expressed here.

The Model Driven Development (MDD) process brings key aspects of these solutions together, utilizing models to describe needed capabilities, and also to drive through the process and connect intent and implementation. This involves languages able to describe needs in linked and in disparate domains, a methodology

that recognizes the separate needs, but the necessary connection points, and focused tools that serve key steps through this flow.

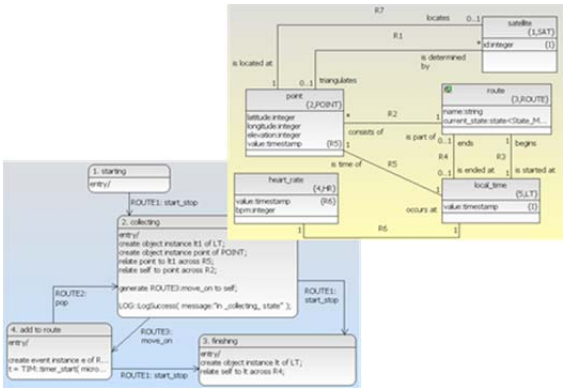
MDD technologies with a greater focus on architecture and corresponding automation yield higher levels of abstraction in the development process. This abstraction promotes simpler models with a greater focus on problem space. Combined with executable semantics this elevates the total level of automation possible.

The Object Management Group (OMG) [6] has developed a set of standards called Model Driven Architecture (MDA) [7], building a foundation for this advanced architecture-focused approach. By applying the wide reaching and integrated automation possible under MDA, engineers using these technologies have found dramatic increases in both total productivity and quality in their development projects.

What is Model-Driven Development?

The models are key to an MDD approach. A model is a simplified or abstracted representation used to explain the workings of a system or element of a system, then applied in a Model Driven Development process to the eventual creation of an implemented result. In this approach, the model becomes the communication vehicle to describe and even demonstrate the product from early stage conceptual design through implementation.

So, why model? A model helps to capture the subject matter, and then enable operating upon or using that model to gain knowledge, and answer questions about the system it describes. Further, a graphical model, such as the UML (Unified Modeling Language) model shown below, adds to the understanding through visualization.



With models in hand, developers can "see" and even "animate" the system requirements, understand and manage interfaces between functionality, assess design decisions, ask questions, try out alternatives, and demonstrate to other stakeholders the intended product.

Starting with requirements, or rather the concepts behind those initial requirements, models help elaborate and validate the real system needs. The model can then be the vehicle to communicate the requirements to the ever-broader team needed for implementation. And it is their interpretation of those requirements that will actually shape the product that eventually emerges from the process. Good models significantly aid in maintaining a consistent view and interpretation.

The model then demonstrates what the sum of requirements really means, and seeks to deliver a presentation of the requirements, in an active form, that can be assessed and adjusted, from the beginning of the overall design process.

The ability to not just document, but *demonstrate*, is a pervasive part of any successful MDD flow, and needs to be in place from the very beginning. It enables the parties to this great design to ask the questions that need to be asked at the beginning of the story, and obtain meaningful answers at the same level of detail available at this design stage.

In a traditional system development process, assignment of functionality to an implementation path is often performed very early in the design process, and is therefore without sufficient information to make a truly informed decision. Furthermore, once selected, the chosen architecture tends to become the only choice; later learning is very hard to incorporate and so the design architecture is prematurely frozen.

Typically, early design stages do not offer enough detail to enable optimum partitioning choices. To improve this step, and enable a more effective and more flexible flow, more information is essential, as is an architecture of the process that will enable change.

Here the use of models, the fundamental premise of MDD, can make the crucial difference. Models can exercise the proposed partitioning and answer the next set of questions.

Rarely is the first architecture the best architecture. Learning during the flow, acknowledging new derived requirements, and dealing with an ever-changing marketplace (i.e., requirements volatility throughout the program) all drive the need for a flexible process. To have the time to execute a significant iteration, the MDD process has to deliver real improvements in the information at hand at each stage, and readily support the consequences of a change. The ability to execute, or simulate, the specification of the expected design behavior becomes crucial to having that flexibility in the process. This idea of being able to both validate and verify design

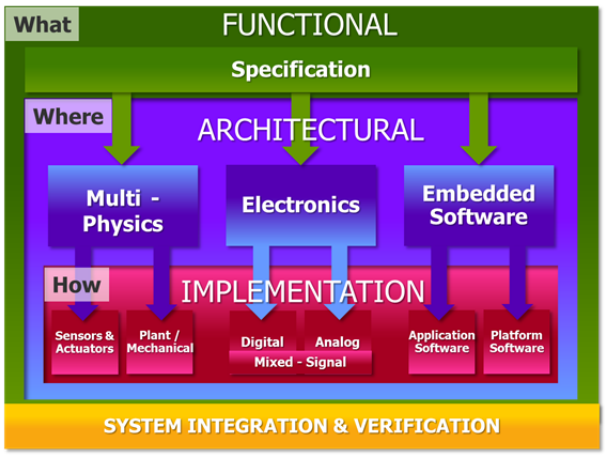
function carries through in an MDD process to every stage of development.

Within the BridgePoint UML tool, two concepts are introduced that really deliver value in the MDD flow: 1) executable models [8] (i.e., the "x" in xtUML), and 2) integral translation (i.e., the "t" in xtUML).

Executable models make possible the Executable Specification, in turn enabling a far more effective process to confirm "what" is being designed. From first concepts, ideas can be refined, potential solutions can be exchanged with other team members, and intentions can be demonstrated through model execution. By contrast with other - more static - drawing or document-driven flows, the dynamic ability to execute the model has been proven to clarify understanding and reduce design iterations significantly. The result is a clearer understanding of the actual system needs, and a specification that can be used to make comparisons throughout the flow. MDD is a significant driver in the move from static documents to a dynamic and synchronized data-driven process.

Another key feature of Model Driven Development, that of driving this data consistently through from step to step in the process, is supported by the Translation aspect of the BridgePoint tool. Taking the model, with design decomposition, interfaces, data and behavior, and passing that forward to both the next stage and tools in the design flow, removes manual conversion errors, and in turn enables a data-driven design process that can be shared across teams and disciplines.

A strong MDD process supports making informed design decisions, asking key questions about design choices, and properly understanding *what* is being developed (i.e., the function), *where* partitioning choices are made (i.e., the architecture), and *how* it will be implemented. In addition, MDD supports verifying the model at each stage of design and thus reducing reliance on late stage physical integration and testing as the primary means to validate the system.



The Idea and Value of a Virtual Prototype

The questions asked (and answered) at the conceptual design stage relate to overall system functionality and behavior, at a high abstraction level, and independent of [most of] the impacts and constraints of the eventual implementation. In practice, the design as conceived and modeled up to this point will be implemented in many teams, most likely focused upon their specific implementation technology domain, such as digital control electronics, analog sensors, system or application software, etc. Model translation facilitates deriving the appropriate set of requirements for those domain needs, and also conversion of the initial model to other forms that enable further questions to be asked, and answered.

To determine if the design has assembled the full set of required elements and if they interact with each other as expected is the role of a Virtual Platform (or Virtual Prototype). Initially, the Virtual Platform does not need great detail (e.g., timing or similar constraints), but does need to enable rapid execution to exercise all the functionality. What the Virtual Platform enables, and at a significant and important stage in the process, is the checkpoint between these partitioned disciplines to ensure continuing consistency. The desired goal of working concurrently, and across design disciplines, is unattainable without mechanisms to bring those domains together at relevant stages through the flow.

But functional behavior is only part of the design goals. Performance, power, real-world interactions, and overall system capability must also be assessed. Often, this can only be done once the first physical prototype is built, but waiting for this to be done is not optimal. Instead, a much more productive process involves using a System Virtual Prototype, whereby the hardware platform, software applications that must run on it and deliver the system behavior, and the external sensors, actuators and real-world interfaces can all be exercised together.

A multi-physics environment, such as that provided by Mentor Graphics SystemVision®, enables a System Virtual Prototype that can include different discuplines, provide simulation and analysis, and be used to address these questions. At this stage, the model contains sufficient detail to start looking at performance, to identify bottlenecks; assess power consumption, in various different conditions; and even test boundaries of constraints that may limit the real system. This is the right time to start asking those sorts of detailed questions.

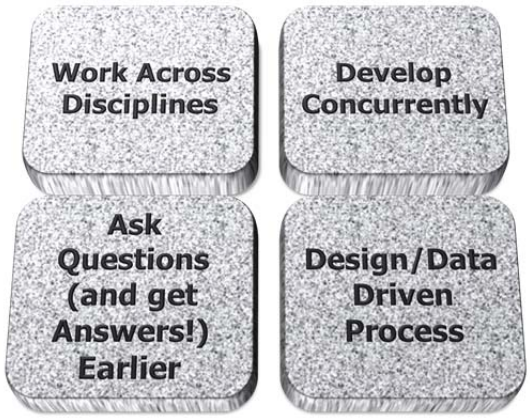
Benefits of a Modern, Model-Driven Development Flow

So MDD is not just a good idea. It actually helps improve productivity in the design process. It makes it possible to use models all the way down the flow, automatically generating parts of the design and thus improving the design's quality by bringing in repeatability and facilitating the process steps necessary in many instances for standards compliance. It enables actions such as eliminating physical prototypes from the process, by instead simulating digital and analog elements working together and with the control software operating the system. It can also eliminate or significantly reduce the paper trail, instead automating the creation of pertinent and necessary documents (for compliance, for example) from live design data.

A productive MDD process incorporates tools and capabilities that facilitate uncovering critical problems early in the development cycle, well before system integration begins.

MDD provides a structure for managing complexity while, at each design stage, making it possible to directly link design functionality back to the program's original requirements and functional specifications. A virtual prototyping infrastructure, in which models from different domains can be integrated at each stage of the design lifecycle, allows system integration issues to be identified and addressed earlier in the process.

Four cornerstones of a modern flow enable these results.



Considering the process gaps discussed earlier, solutions for improvement can be focused on these four areas:

1) Working across disciplines.
2) Developing concurrently as opposed to sequentially.
3) Having design data (as opposed to documents) drive development and communication.
4) Having a process that supports asking questions about the design – and getting answers – earlier, when decisions and changes are economically feasible.

Improvements in all or even any of these areas can yield notable benefits in managing system design complexity. MDD supports improvements in all four areas.

The Next Stage of Process Evolution: Product-Centric Lifecycle Collaboration

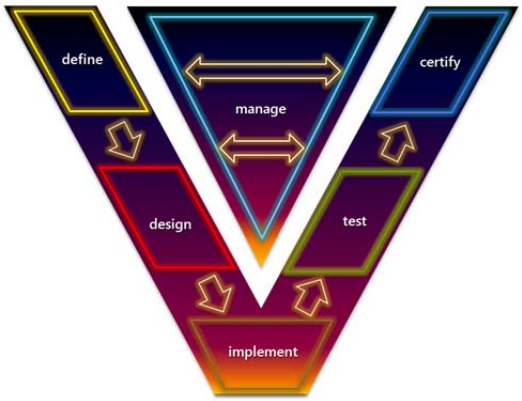
Bringing order to the complex and often chaotic mass of information contained in the data of a product is a perennial and growing challenge. Design data and the design process are intertwined, and connections throughout the flow are hard to follow and maintain.

Companies creating electronic systems are looking for a system development solution that will help them manage the process, the product itself (and future revisions), along with the product development.

Key required attributes include managing product requirements (with functional, performance, interface, safety, certification, and other aspects), communicating and coordinating project status and activities, across disciplines and from concept to implementation, maintaining continuity and synchronization of data, and facilitating needed regulatory compliance. Requirements-driven design and validation/verification throughout the process are two key elements of the processes required to satisfy many regulatory specifications.

An effective solution links lifecycle tools, such as requirements management, defect and change tracking, quality measures, etc. with the design tools and flow, without getting in the way of the normal flow of those activities and tasks.

Communication becomes an integral part of the design flow, and many inter-discipline disconnects are bridged, helping to make possible a concurrent design flow that has a real reason to share data and progress.



Critical questions arise towards the completion of a project. Is the product ready to go out? Is it in actuality what it was supposed to be? Could the development process be repeated? Could it have been developed better? Documenting all that went on in the MDD process is essential and leads to the ability to continuously improve and innovate. Having a mechanism to enable this information sharing within an MDD approach is key.

Examples of Companies Modernizing their Systems Design Approach

Companies either feeling tremendous pain or recognizing tremendous opportunity are motivated to change. Those that have taken steps to modernize their systems development processes – even if it is simply addressing one key issue listed previously-- have seen notable benefits.

For example, a large government contractor developing smart munitions was faced with the challenge of having to demonstrate a design concept in order to win a contract. Instead of building and demonstrating a physical prototype of a precision guided munitions device, as they would have done in the past, they used an MDD approach with VHDL-AMS in SystemVision to develop a model of the device, which involved complex interactions between electronics, electro-mechanics and the power system. Through the use of a virtual prototype, they were able to quickly iterate on the design virtually via the model.

Initially they used the model to prove the concept and win the contract. Later, in the actual development phase, they were able to use the model to very accurately predict the behavior of the system. Prior to using an MDD approach, they would have had to build actual prototype devices and validate them by test firing and measuring the results (an extremely expensive and logistically challenging situation).

The benefits of an MDD approach continued into a subsequent program where they were tasked with developing a similar but variant device. They were able to reuse much of the design, validate the new/different requirements, and develop it very quickly by leveraging their previous work in developing the model. The result was that they produced a working variant design, in short order, and with reduced cost.

Other examples have involved the development of medical devices, which may sound completely contrary to weapons development but at their core are very similar systems involving hardware, software, and mechanical aspects. Two different companies – one doing incubator designs and another developing pacemakers – have seen very similar benefits from adopting MDD flows.

An incubator is a safety-critical device leveraging hardware, software, and mechanical elements to create a life sustaining environment for newborns. One company developing such devices decided to design the whole complex system virtually. They developed models of the analog and digital hardware, including the sensors, actuators, and plant using SystemVision. They used SystemVision conneXion, or SVX, to enable execution of C++ code on the virtual prototype. The early software and hardware/software integration testing gave them the opportunity to review and validate design choices in light of executed behavior, and improve the design to address limitations of the initial concept.

A maker of implantable devices was motivated to reduce their design cycle, and they found a way to shave months off the back-end test process. Instead of waiting to for the device to be built before creating and validating the test set (which was general practice), they decided to develop the test set from the requirements at the early stages of the program. This required developing a model of the system in which to run and validate the test set.

The benefits of this approach were not only a reduction of several months from the back-end of the process, but verification of the device via the model at the very early stages of design.

The system being modeled was quite complex and included a specialized low power SoC device. The process included modeling the hardware specific platform to verify both firmware and application software. This process used BridgePoint to model the system in UML at a relatively high abstraction level, and explore design options by executing the xtUML model directly in the Bridgepoint Verifier. Then the models were used to generate SystemC models of the hardware blocks and C modules for the software. These SystemC models describe hardware functionality at a transaction level, which were then run in the Vista virtual platform environment to go into greater depth, assessing hardware/software interactions and performing initial performance analysis.

Similar engagements with companies modeling secure military communication systems, automotive power systems targeting maximum fuel efficiency, aircraft wiring between systems, and other complex multi-discipline systems all demonstrate the same key benefits despite the variation of end uses. Modernizing processes by using a model-driven development process, and leveraging virtual prototypes, enables early system validation and verification and provides the ability to connect and analyze the disparate domains of development far earlier and with far less expense than a traditional prototype. The next step in this evolution is to integrate this modernized development approach into a transparently collaborative product development environment. This will close the final gap between design development and the lifecycle processes.

Conclusion

Something has got to change. It is becoming too costly and risky to develop complex, multi-discipline systems using processes of the past. Working sequentially and in isolation, waiting for late stage testing to validate/verify the product, and relying on static documents to drive the process are outdated methods that are yielding late, costly, and malfunctioning systems.

This paper explored a more modern approach to system development built on a model-driven development approach. MDD provides a way to use models throughout the flow from executable specification, to concept refinement, to demonstration/validation, through implementation and testing. It can support evaluation of architectural trade-offs and the model can evolve from a functional virtual platform to a system virtual platform, supporting detailed physical mechatronic systems. By employing an MDD approach, iterations are fast and easy, and concurrent design, validation, and verification occur at each stage of design evolution. Requirements traceability and documentation are inherent and synchronous parts of the flow, and not afterthoughts. The model itself becomes both knowledge about the design and a living reference to the intended implementation goal, inherently connecting the disparate groups and disciplines initially by communication/demonstration and later by simulation.

MDD directly addresses the three key challenges of systems design – sequential design, document driven, and isolated development. The concurrent, data-driven, and connected environment of MDD results in higher quality systems developed more quickly and at less cost. It creates an environment where appropriate questions can be asked and the model can be used to demonstrate the answer. This, capability, as part of the overall verification and validation that MDD introduces at each stage of design, reduces reliance on late stage integration testing and minimizes the chances of finding serious problems too late to salvage the program.

References

[1] “System Design: New Product Development for Mechatronics.” Aberdeen Group Report, January 2008.

[2] 2011 National Software and Airborne Electronic Hardware (SW&AEH) Conference, http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/conference/

[3] www.avsi.aero

[4] D. Ward, S. Helton, “Growing the SAVI Paradigm,” presented at the 2011 Safe & Secure Systems & Software Symposium

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

[6] Object Management Group (OMG), www.omg.org

[7] K. Scott, A. Uhl, D. Weise, MDA Distilled. Addison Wesley, 2004

[8] S. Mellor, M. Balcer, Executable UML, a Foundation for Model-Driven Architecture, Addison Wesley, 2002