2010-01-0935

Pragmatic Strategies for Adopting Model-Based Design for Embedded Applications

Eric Dillaber, Larry Kendrick, Wensi Jin, and Vinod Reddy
The MathWorks

ABSTRACT

When transitioning to Model-Based Design for embedded systems development, it is essential to consider an overall plan spanning people, development processes, and tools. A common sense approach when beginning any process improvement activity is to first identify the problem to be solved and then develop a plan to implement the solution. When transitioning to Model-Based Design, performing the transition in an iterative manner – do, learn, adjust, and repeat – has been shown to be most effective. The end goal is a development process where the model is the design, verification is done throughout the development process using simulation, and the implementation of the entire application onto target hardware is highly automated. Faced with design and organizational complexity, time, quality, and cost pressures, the transition is akin to changing a flat tire while moving down the highway. Choosing the right first steps are key to a successful transition. This paper presents a set of practical strategies for determining the first steps when deploying Model-Based Design and code generation in production development processes.

INTRODUCTION

A growing number of embedded systems development organizations are seeking to adopt Model-Based Design to take advantage of benefits, including enhanced ability to deal with complexity, reduced time-to-market, reduced cost, and improved quality. Many organizations have taken the first step by building models for early stage design analysis or rapid prototyping of control algorithms. The next challenge for these organizations is determining how to effectively incorporate Model-Based Design into their production development processes. An effective migration plan requires accounting for the impact that Model-Based Design has on an organization, processes, tools, testing, and design.

This paper presents a set of considerations and strategies for adopting Model-Based Design for embedded software development. A key component of the strategies outlined here is a phased rollout, which spans from the first production deployment to a highly optimized integration of people, processes, and tools that delivers on the promise of Model-Based Design. We present the common challenges associated with such a rollout and the current industry best practices that address them.

The paper is structured into three main sections:

1. Organizational Challenges and Impact
2. Strategy for Change
3. Process and Tool Migration Plan

In the section "Organizational Challenges and Impact," we answer the questions:

- What does Model-Based Design mean to the people in my organization?
- Do I need new skills?
- Are there new kinds of work being performed?
- Should my people be organized differently to take full advantage of Model-Based Design?
- How will the inputs and outputs to my embedded systems development organization change when I adopt Model-Based Design?

In the section "Strategy for Change," we explore how to optimally migrate to Model-Based Design. Our phased approach involves establishing carefully constructed criteria for selecting the first project which address the following questions:

- What problem am I trying to solve?
- What project should I start with?
- What return-on-investment (ROI) is needed to prove the value of Model-Based Design?
- How can risk be mitigated during the migration?
- How can a large legacy code base be effectively migrated?
- After the initial project, what's next?

In the section "Process and Tool Migration Plan," we examine many of the important details that need attention during the transition. Each phase of the development process, including requirements, design, implementation, and verification, is analyzed in some detail. Specific recommendations are made for the initial project and the expanding, optimized rollout of Model Based Design.

ORGANIZATIONAL CHALLENGES AND IMPACT

When adopting Model-Based Design, the requirement to deploy new tools and to support the engineering staff with training is often apparent. However, the need for evolving the development organization to take advantage of Model-Based Design is less obvious but no less critical. The need is driven by three factors:

1. New engineering tasks
2. Emphasis on requirements and design
3. Process and tools integration

New Engineering Tasks: First, the migration to Model-Based Design often brings new tasks that were performed on physical prototypes in the past or not at all. As an example, before the automotive industry adopted model-based calibration methods, prototype engines were subjected to hundreds of hours of dynamometer testing to explore the operational behavior of the engine including thermal, emissions, power, and efficiency characteristics. If problems with the design were discovered, design iteration was performed and the

cycle started over. Since the adoption of Model-Based Design, many engine manufacturers now build and simulate plant models to perform the same behavior exploration plus conduct simulation-based testing that would otherwise not be possible with real hardware due to safety or other constraints. The development organization must be able to carry out new engineering tasks, such as plant modeling, to take full advantage of the benefits of early verification in Model-Based Design. In some cases this may mean new teams must be added to the organization. Using the example above, if a department used traditional CAD methods to design an engine, behavioral models suitable for control design and embedded software verification may not exist. Developing physical modeling expertise within the development organization has proven beneficial to the overall development process by eliminating dependencies on external groups for simulation based controls development. The models developed can focus specifically on the controls development task, which leads to fast simulations and the appropriate level of details in areas that are important to control or diagnostic system development.

The interfaces to other organizations doing mechanical or electric design may also change with the introduction of modeling. Other organizations have chosen to connect the hardware and embedded systems development organizations by having a single model serve both the hardware design community and the controls developers. Care must be taken when multitasking a model like this as the level of detail required differs with the audience. A hardware designer's need may be met with complex models that run slowly, but an embedded systems developer may need real-time performance for exhaustive testing. Many times these two constraints cause the model to be overly complex and of only limited use for embedded system development. Plant models that predict trends rather than provide precise predictions are more useful for both analysis and simulation-based validation. Thus, if the plant model is to be developed by someone other than the control developer, there should be close cooperation between the two disciplines to ensure that the model meets the needs of Model-Based Design.

No matter who builds models, new skills will be required. Close communication between the builders of plant models and the embedded controls/diagnostic designer is a proven effective strategy when making the transition to Model-Based Design. The initial model building capability can be acquired by training existing resources or by utilizing external resources for the initial project, while the internal resources learn through the process.

Emphasis on Requirements and Design: Second, the organization needs to shift focus and resources toward the requirements and design phases of a project in order to take advantage of early verification benefits. It is well documented that much of the time spent by embedded systems design engineers is focused on correcting defects that are found during the later stages of development but were introduced early in the design cycle.

In addition to automatic code generation, early and continuous validation and verification of the design is a cornerstone of Model-Based Design. In fact, the full ROI cannot be realized without leveraging the modeling effort for validation and verification activities. As a result, engineers previously focused on reactive problem resolution will now build robust test harnesses connecting their design and testing infrastructure to the written requirements and executing thorough design validation through simulation. Final-stage verification of the implemented design reusing the same testing infrastructure extracts the last bit of value from the modeling effort, ensures everything is correct before building the first hardware based product, and lets the engineer iterate in a safe environment until all requirements are met.

Process and Tools Integration: Third, the organization must integrate newly acquired tools with a new process to maximize the efficiency gains achievable with Model-Based Design. Automation of key process steps is commonly performed as part of the implementation. The most successful organizations take a step back from their current ways of working and thinking to explore entirely new ways of doing business. As an example, most organizations new to automatic code generation continue the traditional process of code reviews for some

time because of process inertia or as a mechanism to validate the code generation technology itself. Eventually someone challenges the process and the result is shifting the review focus from inspecting code to a reviewing models. Just as it is rare for an organization to inspect the output of a compiler through code review, the manual review of C code or HDL will someday be a thing of the past. The models are the new single source of truth for the design. Continuing down this path, a fully leveraged and integrated process and tool integration institutes model checking mechanisms that automate much of the mundane aspects of manual review of models. This approach avoids wasteful verification effort and frees the engineer to accomplish additional value-added tasks such as formal verification or design optimization.

If a tools and process team is already in place, Model-Based Design environment support can become a natural extension of its responsibility. If such a team does not exist, a cross-functional implementation team should be assembled to represent program management, systems/controls engineering and software needs. The areas of focus for the team should include:

- Defining and carrying out the necessary process and tool changes or integrations, which includes building automated interfaces to other tools in the development process like configuration management systems or text based requirements management, defect tracking, and project management
- Creating the environment and processes utilized by application developers
- Defining and maintaining standards related to Model-Based Design. These standards are critical to conducting parallel development normally associated with large development projects. Parallel development may be conducted by different parts of the same organization, by remotely located elements of the organization or by suppliers. Architectural and modeling style standards, for example, are required for mutual understanding and production feasibility of designs as well as the ability to integrate work products from different sources.
- Configuring and qualifying tools
- Establishing supplier interface procedures and standards

Model-Based Design does not replace engineering expertise such as control design and software architecture. With Model-Based Design, the control engineers' role will expand from requirements to implementation, thanks to production code generation and model verification tools. Software engineers, on the other hand, continue to be of critical importance although their focus will shift toward software architecture, code generation tool configuration, platform software, and system integration.

In summary, the introduction of Model-Based Design requires new skills and creates a stronger distinction between application development and platform development. Managing these changes will require training and organizational changes to fully realize the efficiency and quality advantages of Model-Based Design.

STRATEGY FOR CHANGE

PLAN FOR CHANGE

The introduction of Model-Based Design should align with a required process change, which may be the adoption of standards such as AUTOSAR, ISO 26262, and DO-178C; the development of disruptive technologies such as hybrid electric vehicles and wind power systems; or a continuous-improvement initiative.

Identify the problem you are trying to solve

Before embarking on the transition to Model-Based Design, it is necessary to have a thorough understanding of the inefficiency and waste in the current process and organization. Metrics need to be available or established to set goals, quantify inefficiency, and determine the focus areas, specific strategies, and tools to adopt.

Often, detailed metrics of the current process and organization are limited. In such cases industry metrics serve as a good starting point to identify the key areas to focus process improvement on. A number of references are available that document typical industry metrics [2,3,4]. One study indicates that a major source of delay reported across all industries is getting the specification right and verifying the design. Another study cites that 60% of defects are introduced in the specification while 55% of these defects are not detected until the testing phase, which results in inefficient design and requirements debugging by test and calibration engineers. The relative cost of fixing errors late in a development process is documented in [2]. There are also often overlooked metrics about process inefficiencies such as the amount of time calibration engineers spend debugging controller software.

In summary, focusing process improvements on eliminating defects and waste will achieve the highest return on investment.

Choose a project with the appropriate level of complexity and technology

What projects are the best candidates for initially deploying Model-Based Design? Projects that require minimal changes are not very good candidates because there isn't much opportunity to demonstrate savings. An engine controller being carried over to the next model year with minor calibration changes wouldn't motivate migration to modeling, but a project with significant new feature content such as a new hybrid powertrain controller would. If a project is critical to the success of a company, a back-up design is occasionally developed using traditional processes and methods. This approach serves to mitigate the risk and provide valuable metrics for both process and tooling. Most organizations, however, either chose a project that is slightly less risky or one in which Model-Based Design presents the only way to meet the project schedule.

A typical embedded system spans multiple domains. It is important that components of the selected project can be clearly represented in the modeling domains available in Model-Based Design. For example, control algorithms expressed in a signal flow diagram are best implemented in modeling tools such as Simulink. Algorithms involving complex logic, supervisory control and flow charts for fault management and state machines are best represented using tools such as Stateflow. Components with close ties to hardware, such as the operating system and device drivers, are better suited for low level languages such as C. Where a component doesn't fit, cosimulation can often be performed to allow for a complete system wide simulation for verification and validation.

Mitigate risk with a phased approach

An important initial consideration is how to incorporate tools for Model-Based Design to address objectives while minimizing risk to a program. Production organizations rarely have the luxury of halting development while major process changes are put in place. Often, questions arise on such topics as how development time will be affected, whether the technology can address application needs, and whether the resulting code is efficient enough. These concerns are best addressed through a phased approach to adopting Model-Based Design that involves breaking the deployment of Model-Based Design into phases and optimizing the process with the experience gained from each completed phase. This approach reduces exposure and minimizes risk. Care must be taken to account for the learning curve when planning and drawing conclusions on the potential efficiency gains of the initial project.

The phased approach shown in Figure 1 begins with the development of an initial migration plan that is used for subsequent execution of carefully scaled deployments first of a single component and finally a full application. Full application deployment is followed by a phase of process optimization and expanded rollout, after which maintenance and upgrades will continue as part of ongoing improvement. Each phase should have clearly defined objectives, metrics, and milestones, and include post assessment and migration plan refinements. The activities in each phase encompass both modeling and the application of process improvements. Refinement of the tools and processes for Model-Based Design is an ongoing activity. New capabilities, recognition of additional opportunities for eliminating manual activities, and revision of design standards and techniques will continue to provide opportunities for efficiency and quality gains.

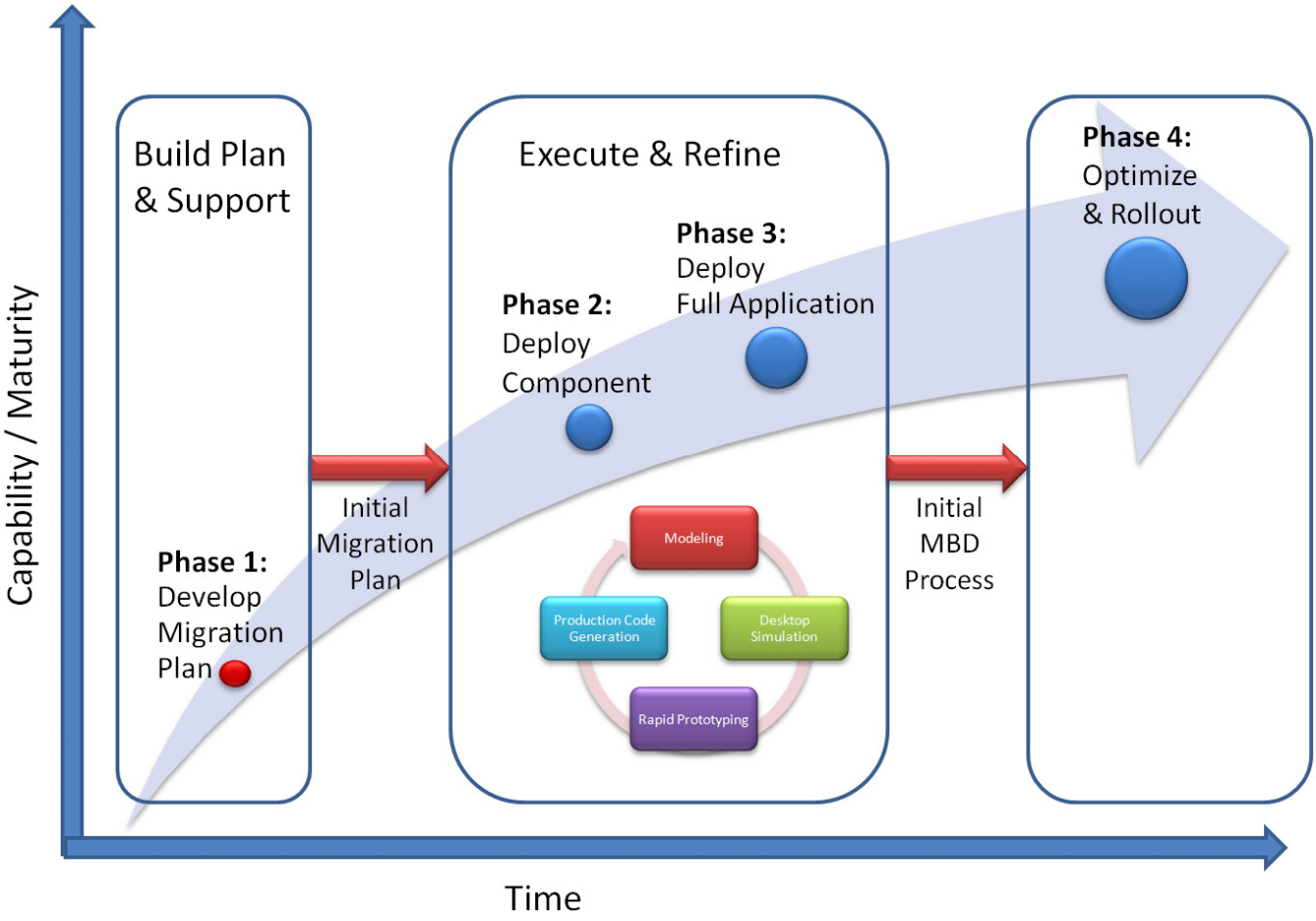

Figure 1: Phased approach to deployment

Phase 1: Develop an initial migration plan

The objective of this phase is to identify the tools and features necessary to address application requirements and process improvement objectives and to understand how the tools integrate with development tool chain. This phase typically starts off with some level of formal training on the tools to understand the capabilities. The next step is to map the application and process requirements onto specific tools and features that will be needed to address them. The final step is to put the knowledge gained into practice by stepping through the development process with a representative model. This phase is commonly accomplished by developing a simple model, generating code from the model, compiling the code, and downloading to the target. Getting support from tool vendors whenever possible will greatly reduce the learning curve. This phase will also demonstrate the feasibility of using the tools and build management support for any subsequent scaled deployment. The end result of this phase is an initial migration plan used for subsequent execution of carefully scaled deployments of first a single component and finally a full application.

Phase 2: Deploy a component

The objective of the component deployment phase is to test and refine the new capabilities put in place by developing a single component with Model-Based Design while limiting the exposure. The generated code is treated as hand code and manually integrated with a standard production application build for the purposes of this phase. Process refinements learned from this phase are fed back into the migration plan as process improvements.

Phase 3: Deploy the full application

The objective of the full application deployment phase is to develop a full application using Model-Based Design, generate code, and automatically build and download the executable onto the target. The integration process with basic software (operating system, device drivers, and communication stacks), as well as the build and download processes are fully automated.

Phase 4: Optimize and expand rollout

The objective of the expanded rollout phase is to optimize and deploy the process to the wider organization. The lessons learned from the previous phases are factored into the process in preparation for deployment. Activities in this phase focus on process improvements to address the wider organizations needs and further improve the process capability and maturity in areas such as requirements traceability, automated document generation, and more extensive use of plant models for continuous verification.

Choose the appropriate legacy components for migration

Many production organizations have a significant amount of legacy code. Should all of the legacy code be converted to models? If so, should this be outsourced or done in house? How much effort would be involved and what would be the benefits?

As discussed in “Best Practices for Establishing a Model-Based Design Culture” [1], migration represents a tremendous learning opportunity. It’s not uncommon for an embedded control strategy developed over the years by different authors to have sections that are not well understood by the current designers. The process of modeling the existing algorithm lets the designer get a deeper understanding of these areas while improving clarity, quality, and maintainability.

While there are clear benefits of migrating legacy code to models, the obvious tradeoffs are the effort required and the risk. Fortunately, it is not necessary to migrate the entire application to take advantage of the benefits of Model-Based Design. Model-Based Design provides facilities to integrate with legacy code that support both simulation and code generation. The system architecture model can thus contain both intrinsically modeled components as well as legacy components. This mixed approach allows for a phased migration of legacy components while supporting system simulation, verification and code generation along the way. To further illustrate, Figure 2 describes the components of a typical embedded software architecture. The shaded components in Figure 3 indicate the strategic migration of software components to the modeling domain over time. Components that are not shaded are retained as legacy code.

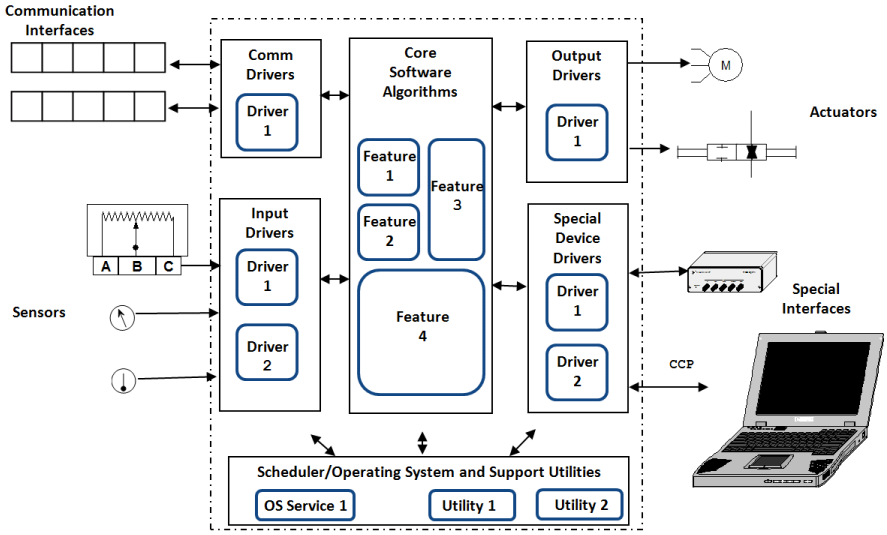

Figure 2: Full application legacy architecture.

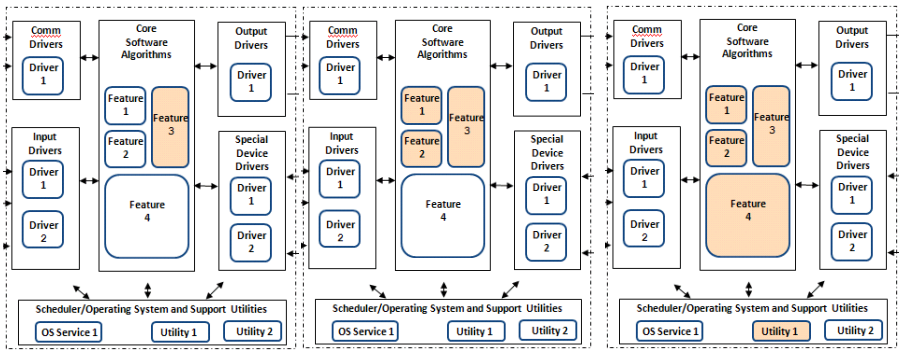

Figure 3: Phases 1 through 3 of legacy migration.

A careful review and analysis of the existing code base should be performed to prioritize the migration of each component. Priority should be given to migrating components with the following attributes:

- Are frequently changed or may need to be in the future
- Have quality issues
- Are complex and difficult to maintain
- Can be clearly represented in the modeling domain

Components that fall outside of these areas provide less benefit to migrating.

PROCESS AND TOOL MIGRATION PLAN

Model-Based Design provides a wide array of tools and techniques to facilitate each phase of an embedded software development process. Developing an initial process and tool migration plan requires an understanding of how the activities of Model-Based Design process differ from traditional processes. To this end we present an overview of activities in Model-Based Design and a set of pragmatic strategies for each stage of the process. The strategies are designed to help an organization discern what approaches to initially adopt and what practices to avoid. These recommendations are based on our cumulative experience in helping organizations make the transition.

Figure 4 illustrates a typical embedded software development process composed of requirements, design, implementation, and integration and test phases. The underlying processes of continuous verification, configuration management, and change management support overall process activities.

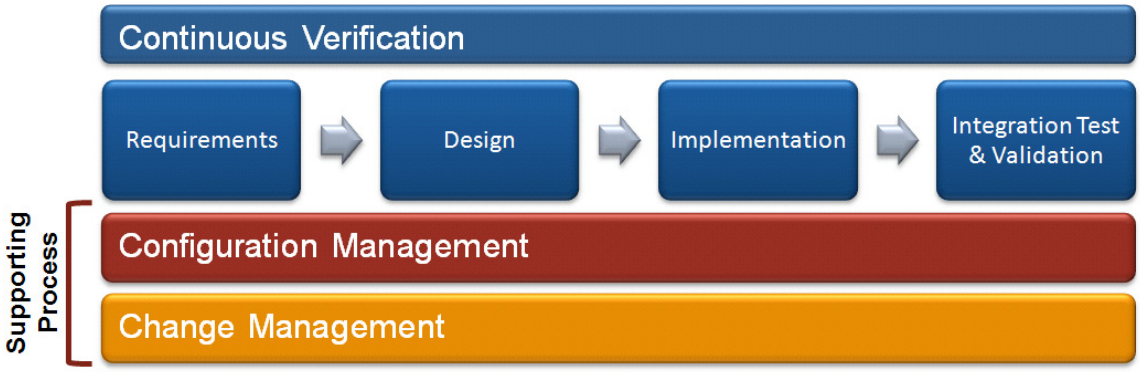

Figure 4: Typical embedded development process.

Requirements phase

Requirements specification is the process of analyzing and documenting the requirements and constraints that the system being developed must satisfy. The ability to use a model to create an executable specification and simulate makes it possible to analyze and validate requirements early and at a level of depth that wasn't previously possible.

Use executable specification development as an opportunity to solidify formal requirements

Industry studies show that requirement errors are the largest source of errors in embedded system development and are the most costly to fix because they typically aren't found until system integration and test at the end of the process. One of the most significant sources of reduced development time and cost savings in Model-Based Design is the early identification and correction of requirements errors. However, if formal requirements do not exist for the project, using executable specification development as an opportunity to create a formal requirement doesn't create immediate savings, but it will yield benefits for subsequent design iterations. In some cases requirements for the design on which the new project is based need to be synthesized from the code implementing it. Once the formal requirement is established, the executable specification should be linked to requirements, and ultimately integrated with the change management process.

Validate requirements before moving to detailed design

Sufficient engineering time should be allocated to requirements gathering, analysis, and verification prior to initiating detailed design. Verification will often involve not only analyzing requirements but, in the case of new or complex applications, it will involve simulation and rapid prototyping to ensure that the requirements are both correct and complete. Failure to do so results in unnecessary design iterations. If verification is not possible without a fully elaborated design, use the model to document the expected output for each of the possible inputs and stub the design with a simplified implementation.

Communicate to internal and external customers and suppliers using a model

Specifications are developed to facilitate communication that occurs between an OEM and its supplier or between two departments within the same company. A model represents an unambiguous executable specification that can serve as an ideal communication tool. Engineers can use the executable specification to help eliminate one of the most common sources of error, misinterpretation and incorrect translation of the specification. When an executable specification is used between an OEM and its supplier, the supplier should review the executable specification with the OEM as a project milestone, demonstrating how each requirement is to be met and confirming behavior of the system as the supplier understands from the requirement.

Make the model a source for documentation

In addition to serving as the core of the executable specification, the model can also be used to produce other documentation such as the test coverage report and, with proper information captured, the calibration user guide. Tools are available to automate the extraction, rearrangement, and presentation of information stored in the model. The automation helps eliminate the need to prepare many design documents separate from the model, which can easily become out of sync with the design.

Design phase

Design is the process of defining the software architecture and interfaces and developing the detailed functions and operations that satisfy the requirements. In Model-Based Design models are used to describe the design rather than traditional design documents. Artifacts developed in the requirements phase such as the executable specification are directly leveraged as a starting point for the initial design, thereby reducing translation errors and accelerating development. Design documents are generated using the model as a source. Desktop simulation and rapid prototyping are used to quickly iterate and verify that the design performs as desired to meet requirements in both simulated and real world conditions.

Choose architecture and component technology early

Model architecture and the technology used to encapsulate components within the design affect a number of areas including simulation performance, code generation speed, code efficiency, configuration and change review process, and maintainability. The merits of component technologies such as model reference, atomic subsystems and libraries should be evaluated early on in the design process to derive a model architecture that meets current needs and is scalable. For example, model reference should be used for components that need to be independently verified outside the model they reside in or when there is a need to generate the same reusable function in software.

Establish and enforce design standards such as MAAB

Modeling environments offer a great deal of flexibility while at the same time provide extensive libraries of design patterns to solve common problems. If not managed carefully, models can quickly become unreadable and inefficient, thus becoming less effective for communication of design information, simulation, and code generation. One solution for preventing ineffective models, is to put design standards in place to ensure that the model is production feasible and easily understood and avoids common design errors before it can move to the next design phase. Start with applying industry standards such the MathWorks Automotive Advisory Board (MAAB) Style Guide, which can be enforced through a set of automated checks. If a safety critical workflow is required, then modeling rules for IEC 61508 or DO-178B should also be considered. Use automated checks to ensure that your design is feasible and implementation will be efficient. The standards and automated checks are based on the cumulative experience of the industry and should be weighed heavily.

Put a process in place to facilitate reusability

Design with reusability in mind, but have a process that facilitates reusability by dedicating resources to both simplify and improve design flexibility. Although this best practice reflects common sense, only a small fraction of companies do it and those that do are the top performers [5]. The top performing companies, identified by their success in hitting a variety of engineering target goals, integrate the preparation and verification of designs for reuse into their processes. Top performers are also more likely to dedicate resources to simplifying designs for reuse. The following strategies can be used to facilitate reusability in Model-Based Design:

- Separate target characteristics, such as data types and device drivers, from the design model. Separation can be done with proper architecture and by using a separate data dictionary to define target specific types
- Use model reference to encapsulate reusable components to facilitate verification and enforce context independent boundaries

Avoid overdesigning

Complex designs can be quickly developed using Model-Based Design. There are often a number of different ways to model the same functionality. Dedicate some effort to reviewing models to identify ways to simplify design patterns that can improve both readability and performance, as in the following examples:

- Use state machines only if a chart output is a function of an internal state not just an input. If an output is a function of only inputs, then a flow chart will be a better representation.
- Review the C-construct guide to understand how to represent common C constructs efficiently in the modeling domain.
- Make commonly used design patterns available by establishing a library to share among other design team members

Develop a plant model with "trend-correct" behavior

Plant models are critical for conducting early and continuous verification from initial concept to production implementation. It is possible to build plant models of various levels of fidelity. Typically, increased fidelity implies additional development effort. A model that reproduces the approximate behavior of the real system in simulation is called a "trend-correct" model. A plant model with "trend-correct" behavior can be used for confirming that a controller model behaves as intended or for regression testing when the controller is changed. Such a model utilizes empirical data mixed with physical laws. Limited in complexity, the plant model provides

approximate behavior between controller output and input (feedback), and can generally be developed with modest resources. Often, "trend-correct" models are developed by the engineers developing control strategies.

Set objectives upfront for utilizing a plant model

Organizations that have successfully deployed Model-Based Design have generally followed the path of starting with "trend-correct" behavioral models and over time evolving models to become more predictive. This approach provides the immediate benefit of having the capability to verify the control strategies' correct behavior and continue refinement of the plant model in support for controller development.

More predictive capability is required when the plant model is to be used to validate a control strategy. A high fidelity predictive model has the potential to reduce or eliminate expensive test cycles using prototype vehicles. However, the development time and technical expertise required and the resulting simulation speed should be carefully planned. As the plant model becomes more predictive, the model and its validation procedure become more complex. It is often necessary to involve an expert with more detailed knowledge of the plant being modeled. Often this expertise is found in the portion of the organization responsible for design of the physical mechanism. The expert would work closely with the intended user of the model to ensure that the model contains the appropriate level of detail.

Implementation phase

Implementation is the process of translating the design to embedded software that can be executed on production target hardware. In Model-Based Design the translation from design to embedded software is automated through code generation, which significantly reduces translation errors [12] and effectively eliminates coding time. Development focus can shift to the creation of intellectual property, such as design models, which are core to a company's competitive advantage and profitability.

Edit the model, not the code

Treat the model as the single source or truth of the design and resist the temptation to modify the generated code directly. Instead, make the change to the model and regenerate code to avoid waste and synchronization issues. In traditional processes this approach is no different than treating the C code as the source and refraining from making changes at the assembly level. Verification efforts resulting from a change can be reduced by assuring your design is partitioned into reasonable size model reference components.

Automate the code integration and build

A production code generator such as Real-Time Workshop Embedded Coder generates target-independent ANSI C/C++ code. If you have an existing build process, the first step to adopting code generation is to treat the code as hand code and integrate it into the existing process. Leverage tool features to automate the process of exporting code to your build environment.

If you don't have an existing build process or would like to further automate the build process, take advantage of an IDE link product and Target Support Package to develop a fully automated application build and download to the target from within the modeling environment. This approach lets developers focus on their designs and quickly iterate and verify on the actual target.

Minimize customizations to reduce maintenance

A production code generator such as Real-Time Workshop Embedded Coder provides many built-in configuration capabilities to achieve common embedded software requirements in the areas of code style, interface definition, data definition, and file formatting. If the built-in capabilities are not sufficient, extensive customizations can be developed to address less common embedded software requirements.

Extensive customizations beyond the normal built-in configuration capabilities require much more effort to develop and maintain and can potentially increase the effort required to upgrade to new releases. Use built-in configuration methods whenever possible to simplify the modeling and code generation environment. In addition, instead of integrating existing legacy functions where equivalent modeling blocks are available use an intrinsic block which often provides more functionality and similar efficiency. The built-in lookup table block for example, provides support for a variety of lookup algorithms and data types, and has had a number of improvements over the years to generate highly efficient code. In summary, spend time developing models instead of supporting infrastructure.

Verification and Validation Phase

With Model-Based Design, models and simulation are used to verify and validate designs early and continuously throughout the development process. The software and system integration test activity of the verification and validation (V&V) phase takes place at the end of development cycle.

In general, the primary objectives of V&V are to ensure that design meets requirements (design verification) and implementation meets requirements (code verification). Using Model-Based Design, the design is represented by a model, and these activities are accomplished by demonstrating that the model meets requirements and the code generated (implementation) from the model meets requirements, as shown in Figure 5. *The software integration activity typically consists of integrating and validating the software generated from the models and legacy software. Processor-in-the-loop (PIL) testing is one of the methods used to perform software integration testing. The system integration activity generally involves the use of hardware-in-the-loop (HIL) testing and testing on the production hardware.*

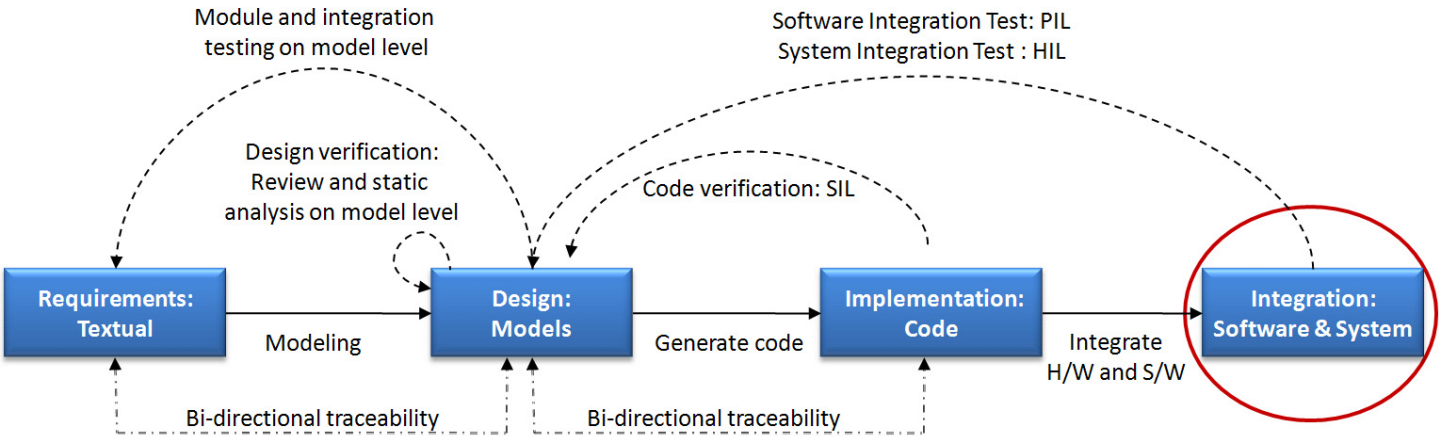

Figure 5: Verification and validation workflow in Model-Based Design.

Verify what you need, not what you want

The first step to deploying V&V on a project using Model-Based Design is to assess what V&V activities are needed to both satisfy requirements and to realize the time savings from Model-Based Design. This determination should be based on the time, quality, and cost goals of the project. An obvious strategy would be to do only as much as necessary. Second, to reduce project costs and ramp-up time, the extent of automation must be carefully chosen and should only be fully automated after the steps in the V&V process are identified and proven. Industry and regulatory standards and guidelines provide objectives for development process activities, which impose additional constraints and require the creation of additional artifacts to show conformance. Model-Based Design supports these activities through automation and reuse of models to facilitate early and continuous verification. In summary, complete the following steps to assess your V&V needs:

1. Determine standards that apply and objectives that must be met (industry and internal objectives). For example, Level A of DO-178B requires traceability to object code, whereas other levels do not.
2. Analyze past software projects to determine the primary source of major defects and waste.
3. Identify additional areas for improvement, such as cost and schedule. For example, if unit testing fell behind schedule due to the time required to create test vectors for full coverage, you can address this issue by adding in an objective in the development process to automate test vector generation. Conversely, if manual vector generation has never been an obstacle, then automation of test vector generation should not be considered as an initial focus area.
4. Determine activities and level of automation required to support objectives
5. Choose tools to facilitate activities and automation required.

Shift the focus to model review

Code reviews are an important part of the traditional design process. In Model-Based Design, models become the true source, translation errors when generating code are minimal [12], and automated technologies such as software-in-the-loop (SIL) testing provide additional verification that the code directly implements the design. More focus is placed on model reviews and formal verification/validation. As a result, engineering time is spent more productively.

MIGRATE KEY SUPPORTING PROCESSES

In a production environment, teams of engineers often work on multiple related projects and update multiple components in parallel. Lack of careful management of component versions and configurations will lead to build and integration errors, adding inefficiency to the project. In many companies, configuration management (CM) is already a formal part of the development process. Whereas in traditional design processes, code is at the center of CM because it is often the most accurate reflection of the design, Model-Based Design shifts the focus to the model, and the CM strategy needs to be adjusted to facilitate engineers working with models. Because the model and other design artifacts are in text format, they can be managed just like source code files, thus requiring no major upgrade of the existing CM infrastructure. The following recommendations can help determine which artifacts should be placed under CM:

1. A design consists of the model, data dictionary, and model libraries, all of which should be in CM, so the design can be reconstructed completely. Tools supporting Model-Based Design also enable the design to be linked with requirements so that bidirectional traceability between the two can be established. Requirements documents should also be placed under CM to maintain their integrity. A design should include its test environment, including test harness models, test vectors, and expected output. A test environment that is

managed with the model ensures that model-based verification including regression testing can be carried out quickly and efficiently.
2. It is important that tool automation scripts and settings for code generation along with source files for legacy application code and files describing I/O drivers and communication stacks be configuration managed with the model so that the application can be consistently recreated. Whether the generated code is placed in CM upon each release along with the model should depend on the CM requirement of the company and the project. For instance, if the requirement is that all files including object code and flash image of the application will be configuration managed, then the generated code should be configuration managed too.

A tool chain supporting Model-Based Design typically includes tools from multiple vendors, so openness is a requirement for any tool to be deployed effectively within the tool chain. CM is no different. For example, to support the basic capabilities required for CM, MATLAB provides an open API which enables it to interface to many common revision control systems. CM strategies vary from company to company. Reference [14] recommends a set of practical strategies, including how to define sets of associated model files to form configurations.

Beyond configuration management, change management is a process in which engineering changes are formally requested and reviewed to determine whether the change is to be made, and then results from the change are documented. Model-Based Design has no impact on this process. However, automation capabilities provided by tools supporting Model-Based Design can simplify or accelerate tasks such as design review and regression testing.

CONCLUSION

In the last decade, many organizations, ranging from small to very large, have successfully made a transition to Model-Based Design. The tool suppliers, such as MathWorks, play a key role in assisting organizations to successfully overcome the challenges of this transition. The strategies presented in this paper represent the collective knowledge from various industries and situations. The intent is to share pragmatic strategies that have been successfully employed by a variety of organizations to facilitate the development of a migration plan for your organization based on your particular situation.

A wealth of material is available to build a detailed knowledge and understanding of the tools and techniques required for Model-Based Design. Beyond knowledge, leveraging the experience of others who have helped organizations successfully navigate the transition and first deployment will help reduce the learning curve, avoid common pitfalls, and accelerate the realization of ROI. In addition, utilizing the best practices that tool suppliers have gleaned from other experiences in your implementation reduces the effort required to sustain Model-Based Design.

REFERENCES

1. Paul F. Smith, Sameer M. Prabhu, Jonathan H. Friedman, The MathWorks, Inc. Best Practices for Establishing a Model-Based Design Culture," SAE Paper 2007-01-0777.
2. J.B. Dabney, "Return on Investment of Independent Verification and Validation Study Preliminary Phase 2B Report." Fairmont, W.V.: NASA IV&V Facility, 2003.

3. Venture Development Corporation. "Embedded Software Strategic Market Intelligence Report, Volume 4, December 2007, VDC."
4. Paul Yanik, Migration from Simulation to verification with ModelSim." EDA Tech Forum, 2004
5. Aberdeen Group. "The Design Reuse Benchmark Report Seizing the Opportunity to Shorten Product Development," February 2007.
6. Kerry Grand, Vinod Reddy, Gen Sasaki, and Eric Dillaber, The MathWorks, Inc. "Large Scale Modeling for Embedded Applications," SAE Paper 2010-01-0938.
7. Tom Erkkinen and Bill Potter, MathWorks Inc. "Model-Based Design for DO-178B with Qualified Tools," AIAA Modeling and Simulation Technologies Conference and Exhibit 2009, AIAA Paper 2009-6233.
8. Tom Erkkinen, The MathWorks, Inc. Scott Breiner, John Deere. "Automatic Code Generation - Technology Adoption Lessons Learned from Commercial Vehicle Case Studies," SAE Paper 2007-01-4249.
9. Jeffrey M. Thate and Larry E. Kendrick, Caterpillar, Inc. Siva Nadarajah, The MathWorks, Inc. "Caterpillar Automatic Code Generation," SAE Paper 2004-01-0894.
10. Edward Kit, Addison-Wesley, "Software Testing in the Real World."
11. Brett Murphy, Amory Wakefield, Jon Friedman, The MathWorks, Inc. "Best Practices for Verification, Validation, and Test in Model-Based Design," SAE Paper 2008-01-1469.
12. Bill Potter, "Achieving Six Sigma Software Quality Through the Use of Automatic Code Generation," 2005 MathWorks International Aerospace and Defense Conference: http://www.mathworks.com/aerospace-defense/miadc05/presentations/potter.pdf.
13. Mirko Conrad, The MathWorks, Inc. Guido Sandmann, The MathWorks GmbH. "A Verification and Validation Workflow for IEC 61508 Applications," SAE Paper 2009-01-0271.
14. Gavin Walker, Jon Friedman, and Rob Aberg, "Configuration Management Within Model-Based Design," SAE Paper 2007-01-1775.
15. Peter J. Schubert, Packer Engineering, Inc. Lev Vitkin and Frank Winters, Delphi Electronics & Safety. "Executable Specs: What Makes One, and How are They Used?" SAE Paper 2006-01-1357.
16. Arvind Hosagrahara, Paul Smith, The MathWorks, Inc. "Measuring Productivity and Quality in Model-Based Design," SAE Paper 2005-01-1357.
17. Jinming Yang, Jason Bauman, Al Beydoun, Lear Corporation. "An Effective Model-Based Development Process Using Simulink/Stateflow for Automotive Body Control Electronics," SAE Paper 2006-01-3501.
18. The MathWorks, "BAE Systems Achieves 80% Reduction in Software-Defined Radio Development Time with Model-Based Design," http://www.mathworks.com, May 2006.
19. The MathWorks, "Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow Version 2.1," http://www.mathworks.com/automotive/standards/maab.html, July, 2007.
20. The MathWorks, "Simulink Report Generator 3.7," http://www.mathworks.com, September 2009.
21. The MathWorks, "Real-Time Workshop Embedded Coder 5 - Developing Embedded Targets," http://www.mathworks.com, September 2009.

CONTACT INFORMATION

Eric Dillaber, Eric.Dillaber@mathworks.com

Wensi Jin, Wensi.Jin@mathworks.com

Larry Kendrick, Larry.Kendrick@mathworks.com

Vinod Reddy, Vinod.Reddy@mathworks.com