2005-01-1357

Measuring Productivity and Quality in Model-Based Design

Arvind Hosagrahara
Technical Consultant (The MathWorks, Inc.)

Paul Smith
Managing Consultant (The MathWorks, Inc.)

ABSTRACT

Accurate measurements of productivity and quality are essential for balancing workload, creating predictable schedules and budgets, and controlling quality. Traditional software development processes include well-established methods for measuring productivity and quality. These include Lines of Code (LOC). With the introduction of Model-Based Design, organizations require a different measure of the software development process.

INTRODUCTION

A measure of the size of a software application, LOC is the foundation for productivity measurements (LOC/unit work) and quality measurements, such as defect densities (defects/LOC).

With the introduction of Model-Based Design, organizations require a different measure of the software development process. For example, Model-based Design enables automatic code generation from graphical models. This means that the average engineer can produce remarkably more LOC per unit time than is possible with hand coding, with virtually no software coding defects. While these productivity gains are grounded in real process improvements, new metrics are required to properly instrument and measure those improvements.

The automatic capture of process metrics in a modern development process increases data accuracy and overall productivity. Practical experience has shown that an organization quickly learns to "manage" metrics captured manually to produce mandated improvements, often without improving the underlying process. Additionally, requiring developers to manually capture a comprehensive set of process metrics can burden and distract them from their primary work. Model-Based Design offers the capability to automatically extract metrics, minimizing cost, time to market, and avoidance of quality-related issues.

This article describes an automatic, noninvasive measurement technique for gathering accurate metrics. We describe specific measurements that should be captured when using Model-Based Design and introduce a free tool that can capture these process metrics in the Simulink® and Stateflow® environment.

OVERVIEW OF MODEL-BASED DESIGN

At the heart of Model-Based Design are Simulink models, graphical, hierarchical, executable block diagram representations of the physical system, the environment, and algorithm behavior (a control or signal processing and communications application).

The models provide:

- A behavioral description of the embedded software—the physical objects and environment upon which it acts
- An executable specification that can be tested using simulation to ensure that it meets all functional requirements
- A tool that enables the design team to communicate information about the design
- A specification from which real-time software code can be automatically generated for testing, prototyping and embedded implementation
- Automatically generated code that can be used by itself or integrated with other code to form the complete embedded application

Model-Based Design enables design engineers to quickly evaluate multiple design options by testing and optimizing their algorithms in the modeling environment before they deploy them as an embedded system, reducing design time and development and implementation costs.

Because the design engineer is working in a simulation environment, design iterations are often much faster and more flexible. Design options are validated and tested by tracking the behavior of the design via simulated tests. Once the system is optimized and performance meets expectations, the model is used to build a hardware-based real-time prototyping system for testing.

When the design meets specifications, the Simulink model can be used to generate production-quality code using Real-time Workshop® and Real-Time Workshop Embedded Coder. This stage often involves a rigorous test process before final system deployment.

The engineering process followed by many modern control system development organizations can be described through the "V" design and development diagram shown in Figure 1.

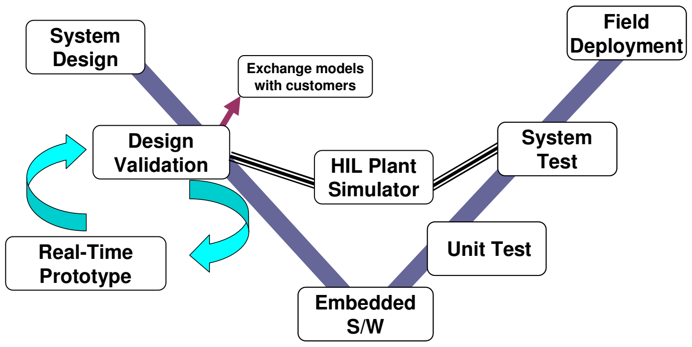

Figure 1. The design "V" process.

The "V" is a two-dimensional view of a development process in which the x-axis represents time and the y-axis, abstraction. As you move down the left-hand side of the V you add more detail to the design, with implementation at the vertex. As you move up the right-hand side of the V, you reach increasing levels of integration and test, culminating in a working product. Formal process measurements are required at each step of the "V" design cycle.

With Model-Based Design, a project that could take months to complete using a traditional development process iterates through the various phases of the "V" process in hours or days. This increase in efficiency enables the rapid evolution of designs and promotes a spiral development method or one where rapid iterations occur, zeroing in on the final design solution.

The diagram in Figure 2 shows the major process elements used in Model-Based Design. These elements can be mapped onto the "V," a waterfall process model [2, 3], or any other development process.

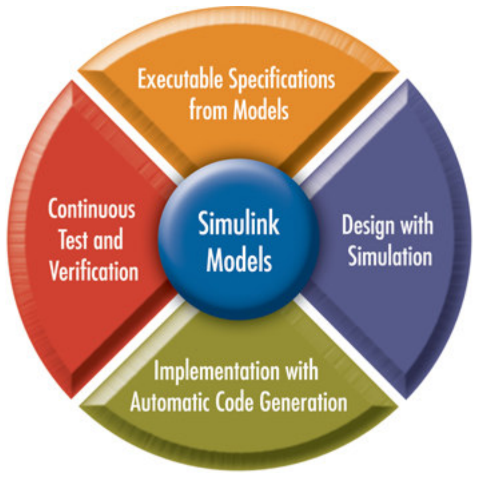
Executable Specifications from Models
Continuous Test and Verification
Simulink Models
Design with Simulation
Implementation with Automatic Code Generation

Figure 2. Major elements of Model-Based Design.

TRADITIONAL EMBEDDED SOFTWARE METRICS

Most organizations that work in a C or C++ embedded software development environment use a variety of tools to measure effort and productivity. These include LOC, used with the popular Constructive Cost Model(s) (COCOMO) [4]. Other common metrics include the change in LOC from one design iteration to the next, computational complexity, and time spent per design task. These metrics are usually collected manually or semi-automatically.

An organization that uses a traditional design approach will typically measure the following:

SYSTEM DESIGN

- Time required to produce each design iteration
- Total number of design iterations required
- Number of defects introduced
- Time required to repair design defects
- Number of functional requirements
- Number of functional test scenarios
- Objective measures of design size and complexity

DESIGN VALIDATION

- Time required to produce each design iteration
- Number of defects introduced
- Time required to repair design defects
- Number of test scenarios simulations run
- Time required to run each test scenario simulation
- Total number of design iterations required

EMBEDDED SOFTWARE

- Number of System Lines of Code (SLOC or LOC)
- Time required to generate code or implement functionality (Time/LOC)
- Defects introduced per LOC
- Time required to repair each software defect

UNIT TEST

- Number of test cases
- Time required to prepare each test case
- Number of defects detected in each test case
- Time required to run each test case
- Time required to repair each defect
- Defect detection and removal efficiency

SYSTEM TEST

- Time required to execute each test case
- Number of defects detected in each test case
- Time required to repair each defect
- Defect detection and removal efficiency

FIELD DEPLOYMENT

- Number of defects detected
- Time or cost to fix each defect
- Measurements in Model-Based Design

With Model-Based Design, traditional metrics can result in misleading conclusions. For example, code is generated automatically, producing thousands of LOC in minutes. As a result, using LOC to measure design effort and time invested can produce a misleading result. To measure process efficiencies accurately, metrics systems must emphasize upstream design activities in the modeling environment.

For example, in Model-Based Design, a task such as switching the structure of a controller from a PID loop to a feed-forward network typically takes a few mouse clicks in a model. (In a traditional process, the software designer would need to rework large sections of code, a time-consuming and error-prone operation.) This small change in the modeling environment could result in the modification of several thousand lines of generated code, skewing productivity and quality metrics while failing to capture the actual time or effort invested in making the modification.

For certain processes, such as the Design for Six Sigma (DFSS), performance measurement depends on the ability to quantitatively track and manage projects (both individually and collectively) [5]. When Model-Based Design is employed in a DFSS project to meet productivity and quality objectives, it is essential to accurately measure the actual effort required [6].

Model-Based Design uses the following measurements of process efficiency and productivity:

- Development time schedules and performance against those schedules
- Modeling development resource requirements and associated costs
- Model size, complexity, and functionality
- Model quality
- Size of the automatically generated code
- Simulation speed and overall time spent simulating models
- Collecting structural coverage metrics directly from the model

<u>Development Time Schedules</u>

Model-Based Design requires specific measurements of each item in a development plan. Project managers set development schedules at the outset of a project and track them using one of several commercially available tools. Once the project is initiated, the developers must link schedule requirements to actual progress. In Model-Based Design, they can do this by linking specific model components to line items in a schedule and maintaining and reporting status in the form of percent complete or some other objective measure. Simulink provides documented and open APIs that enable users to develop interfaces and extensions for tracking these types of items

<u>Modeling Development Resource Requirements</u>

To predict modeling resource requirements, the project manager must estimate both the size of the model required and modeling productivity. Such estimates depend on a historical account of modeling efforts and rates of development. Project complexity also affects resource requirements and overall productivity. In Model-Based Design, two metrics are used to obtain these measurements:

Objective measure of model content—can include model block counts and the numbers of signals, layers in the hierarchy, block parameters, states, masked blocks and mask parameters, state transition diagrams, events, and specific types of blocks. A weighted sum of all these measures is useful in measuring overall content.

Time required to produce content—must be estimated or measured by the engineers doing the work.

<u>Model Complexity</u>

The key to making a practical measurement process work is to design a mechanism for computing and measuring complexity and then keep that metric stable throughout the project. It is tempting to continually refine a complexity measurement to get the "right" or "best" metric, but evolution of this measure will result in different measurements for the same model, leading to schedule and resource allocation errors. At the end of each project, it is appropriate to review and refine the measurement system, as long as one can go back into the historical record and make new measurements on previously archived blocks of functionality.

<u>Model Quality</u>

Model quality is measured by defect densities, which must be calculated by the engineers who track design performance and functionality. This can take the form of number of defects detected per unit time or defects per unit size of the model. Automation of this measure enables an organization to implement advanced automated testing procedures.

Code Size

Traditional tools can be used to measure the size of automatically generated code. This metric is important only during the transition to Model-Based Design, to demonstrate the effectiveness of Model-Based Design to product development managers. It is also relevant when operating in a memory-space-constrained target. Modeling effort can be linked to LOC to highlight the productivity gains achieved over traditional hand coding. Code size and modeling effort measurements should be presented together to show productivity metrics.

Simulation Speed and Structural Coverage

To enable accurate objective measurements, simulation speed should be measured and a ratio between model size or complexity (or both) and absolute speed should be reported, together with platform-specific parameters such as CPU type and speed and RAM space.

Additionally, an objective measure of the total time spent simulating models should be captured to ensure thorough model execution. Project managers will soon question why models are being created, if they are not being used to simulate alternative designs or validation and verification scenarios, or for advanced tuning and optimization of controls. If batch simulations are used for validation and verification, this must also be noted, as the engineers participate only in the set-up and post analysis of the results.

The Simulink Accelerator, a Simulink companion product for accelerating and optimizing model performance, can collect performance data while simulating the model. The resulting simulation profile report shows how much time Simulink takes to execute each simulation method, highlighting efforts to optimize model simulation speed.

Structural coverage metrics collected at the model level are an important measure of the quality and completeness of the test cases being developed, and provide ideas for algorithm optimization. Simulink Verification and Validation, another Simulink companion product, provides a measure of cyclomatic complexity, a measure of the structural complexity of the model. It approximates the McCabe complexity measure [2, 7] for code generated from the model.

The final section of this article describes a custom tool that was designed to measure the time and effort spent modeling with Simulink and Stateflow. The tool was built based on an aggregation of customer process measurement requirements, using MathWorks products.

THE MODELING METRIC TOOL

Several process measurement techniques are standard in the embedded systems development community. These include Practical Software and System Measurement (PSM) [8] and Software Engineering Measurement and Analysis (SEMA) [9]. They are used to feed a variety of process management techniques, such as Six Sigma initiatives and the Capability Maturity Model for Software (SW- CMM or CMMI).

In Model-Based Design, gathering accurate metrics is essential to process improvement. A high-level graphical design environment such as Simulink or Stateflow lets you automatically gather accurate measurements that accurately reflect the effort invested in algorithm design. The automated measurement of these metrics minimizes human error and ensures meaningful, usable, and stable results.

The Modeling Metric Tool is a graphical user interface (GUI) that enables you to quantitatively measure the content of a Simulink and Stateflow model and incorporate these metrics into a development process measurement system. A detailed description of the tool is provided in the user manual, part of the download package available on MATLAB Central.

The Modeling Metric Tool supports:

- Model-Based Design in Simulink and Stateflow
- Structural modeling features, such as libraries and charts
- Automated capture of input activity and time spent on various tasks
- Automated analysis and documentation of captured metrics

It provides the following functionality:

- Enables customization of the tool and metrics for different applications and projects
- Supports Simulink and Stateflow with no other product dependencies
- Works with semi-complete models (that do not simulate)
- Works with models with incomplete library linkage information (broken links or partial components of large models)
- Supports custom masked blocks
- Extracts in-depth measurement of the Stateflow content
- Reports details of hand-written custom code included with Stateflow
- Enables quantitative capture of all input activity for the automatic measurement of person-effort in developing the model
- Generates reports and analysis plots

One of the most important measurements that the tool provides is the measurement of the model functional content. This set of metrics, when combined with metrics from the task time tracker and the journal, gives a detailed picture of the productivity and effort involved in developing a model.

CONCLUSION

Measuring content, effort, and time in Model-Based Design poses new challenges. The techniques and metrics described in this article and demonstrated in the Modeling Metric Tool can help capture these measurements and can be used as an integral part of any process improvement cycle.

REFERENCES

1. The MathWorks Inc. (2004), *A Model-Based Design Approach* . Retrieved June 22, 2004 from www.mathworks.com/applications/controldesign/description/index.html

2. Stephen R. Schach, "The Waterfall Model," *Software Engineering* , Richard D. Irwin Inc. and Aksen Associates Inc (1990).

3. W. W. Royce, "Managing the development of large software systems: Concepts and Techniques," *Proceedings of WestCon* (August 1970).

4. B. Boehm, *Software Engineering Economics* , Prentice Hall (1981).

5. Michael Harry Ph.D. and Richard Schroeder, *Six Sigma* , DoubleDay Publications (2000).

6. Forest W. Breyfogle III , *Implementing Six Sigma, Smarter Solutions using Statistical Methods - 2nd Ed* , © John Wiley and Sons (2003).

7. T. J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering* SE-2, (December 1976).

8. Practical Software and System Measurement, *A Foundation for Objective Project Management* , Retrieved June 22, 2004 from www.psmsc.com/Default.asp.

9. Carnegie Mellon Software Engineering Institute, *Software Engineering Measurement and Analysis (SEMA)*. Retrieved June 22, 2004 from www.sei.cmu.edu/sema/welcome.html.

CONTACT

Arvind Hosagrahara and Paul Smith work in the Consulting Services Group at The MathWorks, Inc. They can be reached at:

arvind.hosagrahara@mathworks.com
paul.smith@mathworks.com