

A Model Driven Approach to Software Development for Systems

“If your code doesn’t run because someone changed the hardware and the guy who wrote the code left the company 10 years ago –

You just might be an Embedded Systems/Software Developer ... who doesn’t use MDD”

Jerry Krasner, Ph.D., MBA

November 2008

Embedded Market Forecasters

American Technology International, Inc.

Embedded Market Forecasters
Research and Consulting
for Embedded Products,
Markets and Channels



About EMF: www.embeddedforecast.com 508-881-1850

EMF is the premier market intelligence and advisory firm in the embedded technology industry. Embedded technology refers to the ubiquitous class of products which use some type of processor as a controller. These products include guided missiles, radars, and avionics as well as robots, automobiles, telecom gear, and medical electronics.

Embedded Market Forecasters (EMF) is the market research division of American Technology International, Inc. EMF clients range from startups to Global 100 companies worldwide. Founded by Dr. Jerry Krasner, a recognized authority on electronics markets, product development and channel distribution, EMF is headquartered in Framingham, Mass.

About the author:

Jerry Krasner, Ph.D., MBA is Vice President of Embedded Market Forecasters and its parent company, American Technology International. A recognized authority with over 30 years of embedded industry experience, Dr. Krasner was formerly Chairman of Biomedical Engineering at Boston University, and Chairman of Electrical and Computer Engineering at Wentworth Institute of Technology and Bunker Hill Community College. In addition to his academic appointments, Dr. Krasner served as President of Biocybernetics, Inc. and CLINCO, Inc., Executive Vice President of Plasmedics, Inc. and Clinical Development Corporation, and Director of Medical Sciences for the Carnegie-Mellon Institute of Research. Earlier, he was Senior Engineer at the MIT Instrumentation Laboratory. Dr. Krasner earned BSEE and MSEE degrees from Washington University, a Ph.D. in Medical Physiology / Biophysics from Boston University and an MBA from Nichols College. He is a visiting professor at the Universidad de Las Palmas (Spain) where he is recognized for his work in neurosciences and computer technology.

Copyright 2008 by Embedded Market Forecasters, a division of American Technology International, Inc, 1257 Worcester Road #500, Framingham, MA 01701. All rights reserved. No part of this document covered by copyright hereon may be reproduced or copied without expressed permission. Every effort has been made to provide accurate data. To the best of the editor's knowledge, data is reliable and complete, but no warranty is made for this.

2008
**A Model Driven Approach to Software
Development for Systems**

Jerry Krasner, Ph.D.

November 2008

Table of Contents

I.	Introduction: Embedded Systems – the most important market opportunity of the Millennium	4
II.	The Embedded Marketplace	7
III.	Making the Case for MDD	9
IV.	Why Can't we all Work Together – Creating a Common Platform for C and OO Systems and Software Developers to Co-create	13
V.	Using MDD for designs and systems for which frequent hardware changes are expected – how MDA creates solutions available to developers	17
VI.	Gaining Better and Cost Effective System Design Outcomes Using Model Based Testing (MBT)	20
VII.	Summary	24

I. Introduction: Embedded Systems – the most important market opportunity of the Millennium

Considering Systems

In terms of dollars spent, the systems marketplace is orders of magnitude greater than the entire embedded marketplace – yet the solutions to the systems' design and deployment problems experienced by large vendors and government and military procurement are embedded solutions. In the not too distant future, the embedded marketplace will grow to address these systems engineering needs.

We are surrounded by systems – one's telephone is an obvious example. So is the car that you're driving, as well as the airplane that you travel on, the train you occasionally take, the entertainment center in your home and the system that connects your wireless phone to the SS7 system so that you can contact friends and colleagues at home or in their offices.

The military is replete with systems within systems. The communications antenna in an aircraft or ship is a small system component within larger systems components. It's not easy to simply replace this antenna with a new one. One has to take into consideration all interfaces to the antenna subsystem and the impact the replacement will have on the total systems operation.

When an auto manufacturer wants to replace a component of, say, the drive train assembly controller, it would be desirable to have tools that could accommodate legacy software with the new changes – without having to rewrite the entire system (code reuse).

What happens when a system's underlying hardware needs replacement as busses and components are brought to end-of-life by suppliers? How can systems developers and architects redeploy tested and operational software in such cases and, perhaps, preserve intellectual property, and maintain backward compatibility? What if the original developers left the company decades ago and no one can locate – or understand – the documentation?

NASA, for example, is planning to put a man on Mars in 20 years; naval warships have a lifetime measured in decades. A lot of changes in technology will occur in the span of 20-30 years. How can systems developers and architects create an assurance of long term maintenance, and code compatibility?

These systems development processes are yielding to a process known as Model Driven Development (MDD) that effectively addresses each of these issues in a cost effective manner.

In this report we will use data from the extensive 2008 survey of embedded developers to examine systems development conducted using Model Driven Development (MDD) tools as compared with a similar development that doesn't use MDD tools.

Herein we are referring to full capability UML-based MDD rather than inexpensive UML graphics tools that are sometimes sold under the MDD label. EMF data shows a

significantly higher ROI when developing embedded software products using MDD tools, in comparison to both model-based drawing tools and development approaches that rely on manual development.

There are a number of systems development issues (that also pertain to embedded product developments) that are uniquely addressed by MDD.

Data will be presented comparing systems design outcomes for MDD and non-MDD developments.

Sizing the Military Systems Market

Software acquisition at the highest levels of the government represents the largest segment of the systems market. However government acquisition processes are fraught with waste. This represents a potentially significant opportunity for vendors who can deliver quality results on time. These concerns can be addressed by MDD today.

Witness the testimony of Statement of Michael J. Sullivan, Director Acquisition and Sourcing Management for the GAO to Congress (Committee on Oversight and Government Reform – House of Representatives) April 29, 2008.

Investment in weapon acquisition programs is now at its highest level in two decades. The department expects to invest about \$900 billion (fiscal year 2008 dollars) over the next 5 years on development and procurement with more than \$335 billion invested specifically in major defense acquisition programs. Every dollar spent inefficiently in acquiring weapon systems is less money available for other budget priorities—such as the global war on terror and growing entitlement programs...

Our analysis shows that current programs are experiencing, on average, a 21-month delay in delivering initial capabilities to the warfighter, a 5-month increase over fiscal year 2000 programs...

At the program level, none of the weapon programs we assessed had proceeded through system development meeting the best practices standards for mature technologies, stable design, and mature production processes—all prerequisites for achieving planned cost, schedule, and performance outcomes. In addition, only a small percentage of programs used two key systems engineering tools—preliminary design reviews and prototypes to demonstrate the maturity of the product's design by critical junctures...

In addition, we found four factors that have the potential to impact acquisition outcomes on individual programs: (1) unsettled requirements in acquisition programs can create significant turbulence including increased cost growth; (2) frequent program manager turnover during system development challenges continuity and accountability; (3) extensive reliance on contractors to perform roles that have in the past been performed by government employees raises questions about whether DOD has the appropriate mix of staff and capabilities within its workforce to effectively manage programs; and (4) difficulty managing software, as evidenced by changes to the amount of software that needs to be developed, indicates the potential for cost and schedule problems.”

It is interesting to note that systems engineering, requirements management and MDD tools provide an immediate solution to these four factors cited by Mr. Sullivan in the last paragraph.

- 1) Requirements management processes can effectively address this issue
- 2) MDD effectively addresses the lack of systems architects and provides traceability
- 3) MDD permits multiple participants (across the globe) to work on the same complex system while maintaining full management and responsibility assessments by senior managers
- 4) MDD is ideal for managing software development and deployment – and is essential when systems upgrades become necessary even decades after deployment

Sizing the Cost of Poor Systems Development Practices

Consider the analysis of Steven Roerman, CEO of Lone Star Aerospace a leading government and industry consulting group.

"The Department of Defense is probably spending something over ten billion dollars a year to support software practices that are far from best in class. Clearly, some costs are influenced by unstable requirements and other factors that have little to do with software alone. But EMF data shows significant performance differences between leaders and laggards in the defense community. Laggards cost the taxpayers money, and are at a competitive disadvantage."

EMF data, as published by Roerman, shows that Leaders (developers that have more than 50% of their designs completed ahead of schedule) are much more likely to use MDD than Laggards (developers that have more than 50% of their development completed behind schedule).

Roerman has used EMF's extensive year-over-year survey data to show that programs that use Model Driven Development (MDD) have far better design outcomes, including better performance and systems functionality, and much less waste. Currently the Under Secretary of Defense is working with Lone Star Aerospace (and hence EMF) to address this serious issue.

Lone Star Aerospace surveys of very senior military and government personnel show that MDD is recognized as the 2nd most important "best practice". In this report, we will show why MDD is the preferred solution for systems design as well as for software development.

II. The Embedded Marketplace

The analyses of evolutionary forces following Darwin shows that change, when it occurs, is rapid and decisive thereby changing irrevocably the landscape and the survivors. We have witnessed this in technological terms every decade since the 1970's, and we are currently witnessing such an irrevocable upheaval in the marketplace for software and systems design and development.

Today, military and government markets are moving cautiously but decisively towards a software centric purchasing model. Interestingly, as telecom and other communication markets for merchant computer boards decline precipitously, the value of software as a product offering has increased significantly.

Certified operating systems, certified code and security certified operating systems (defined under MILS, DO 178B and Common Criteria) stand ready to be adopted by medical, transportation and mil/aero systems integrators and developers. What they lack is a coordinated development platform from which to develop, to deploy, and when necessary, to modify and redeploy.

The embedded marketplace for developers has undertaken a radical upheaval, driven by economic and technical forces that are unyielding and destined to change the embedded landscape. We are beginning to see profound changes in COTS availability, new bus architectures, high speed interconnects and design complexities. These in return are causing a "ripple-down" effect on software design complexities and the requirement of new and better tool sets to accommodate multicore and multithreaded technologies. Looking ahead we expect that the embedded industry will experience a series of hardware and architectural changes (every 18 – 36 months) necessitating a redeployment of developed software to new hardware interfaces. This will be of particular concern to military and avionics programs that are measured in decades, not in years.

We are at a unique time in the history of embedded developments, newer applications and utilization of technology. Imagine the challenges:

- Developers are faced with increased design complexities, diminishing time-to-market requirements, familiar components being brought to end-of-life, and changing development processes and tool availability.
- There is a need to deliver quality software on-time and within pre-design expectations - windows of opportunity are short and getting shorter. There is a major cost of being late to market.
- COTS users (e.g., military, avionics, telecom, etc.) are learning that COTS availability is driven by a diminishing number of embedded vertical markets that continue to supply high performance components and boards as well as interconnect technologies. Chip and board manufacturers are moving to the high volume and lower quality processors used in consumer electronic devices that have very short life spans as compared to the long term needs of major systems deployment.
- We are in a period of frequent hardware changes – components, processors and architectures – that necessitate either rewriting drivers or moving to a Model Driven Architecture (MDA) abstraction. With hardware changing every 18-36

- months, the best solution is to either begin with MDD or use MDD to import legacy code that is abstracted for MDA.
- EMF survey results show that the best design outcomes are those that have provided for code reuse – automotive, avionic and medical designs need to be upgradeable without the need for major re-coding.
 - With systems complexity increasing, the percent of project time and the percent of project costs consumed by testing are increasing. EMF has tracked and analyzed non-MDD developments with code based testing (Traditional), MDD developments with code based testing (Transitional), and MDD developments with model based testing (Enhanced). For Enhanced developments, EMF data shows a reduction in time-to-market, superior design outcomes, fewer behind schedule deliveries and reduced project time consumed by testing and reduced project cost consumed by testing.
 - The largest segments of the embedded/enterprise marketplace are recognizing the importance of systems development and the required complexities of systems-within-systems that go far beyond current design and development processes. The latest advances in Eclipse-based OS agnostic IDEs, impressive as they are, cannot address the special considerations that systems and systems-within-systems programs demand.
 - There is a growing need for systems engineers and software engineers to be able to work together on common designs within their most comfortable design environments. Hence C developers need to be able to work with OO developers without having to learn how to design within an OO environment, while continuing to maintain a cooperative design process.

What is most important is that these challenges be met with a flexible solution that addresses each and all of these concerns.

Fortunately that solution is available today – and is the focus of this paper.

III. Making the Case for MDD

MDD has emerged as a preferred method for software design, deployment and maintenance. MDD enables the following advantages:

- Code reuse
- Interoperability
- Easy code redeployment under frequent hardware changes necessitated by processor end-of-life events
- Integrated documentation
- Enhanced design capabilities – better design outcomes
- The ability to integrate legacy code into a new format
- The ability to do systems level design – and systems within systems design and analysis

In addition, MDD uniquely provides the following abilities that non-MDD tools do not:

- Clear, traceable and testable Requirements
- Good Architectural Design
- Automatic code generation
- Rapid prototyping
- Effective Communications
- Automatic Documentation
- Graphical Design Reviews
- Evolving/Iterative Prototypes
- Executable Designs
- Effective Debugging Tools

The salient question remains as to whether MDD users have better design outcomes than non-MDD users. To answer this question we turn to the following data derived from an extensive survey of 455 embedded developers.

Table III-1 presents a comparison between MDD developers and non-modelers. Table III-1 also breaks out its findings according to code size.

	Behind Schedule	Months Behind	Months start to shipme nt	Total (ave) Lines of code x1000
MDD developers	35.1%	3.7	15.8	713.6
Non-modeling developers	44.3%	4.2	13.6	649.4
Lines of code <100k	43.3%	3.5	12.4	
Lines of code > 100k	44.1%	4.8	16.5	
Lines of code > 500k	46.2%	5.5	17.5	

Table III-1

In determining comparative design outcomes, EMF uses the following criteria:

- Time-to-shipment of project (months)

- Percent of designs completed behind schedule – and average number of months behind
- Percent of designs cancelled – and average number of months before cancellation
- Closeness of the final design outcome to the pre-design expectation (for Performance, Systems Functionality and Features & Schedule).
- EMF considers final design outcomes within 30% of expectation to be satisfactory

Table III-2 presents a comparison for final design outcomes within 30% of pre-design expectations.

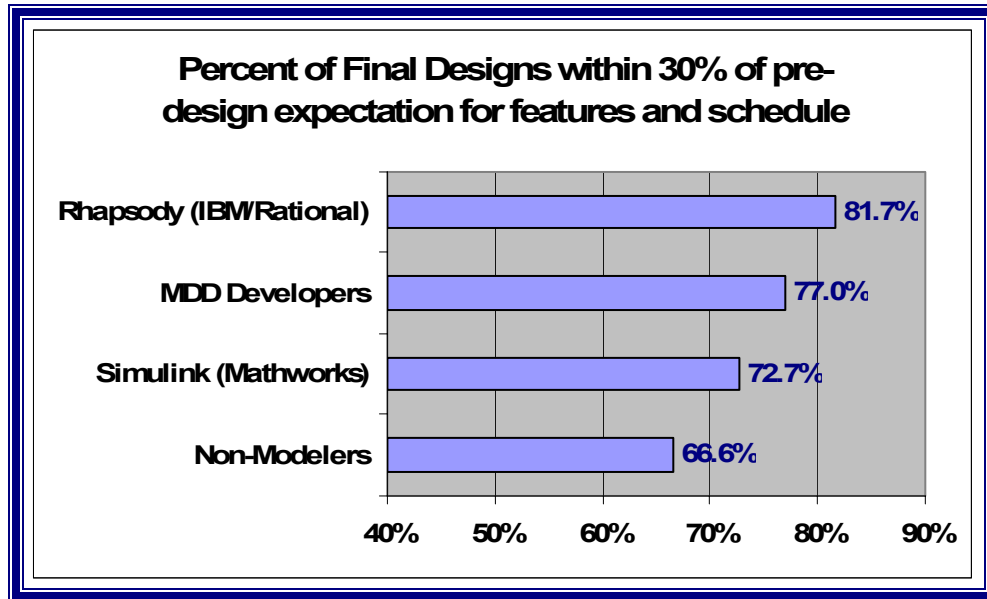


Table III-2

As a comparative example of MDD versus non-MDD designs, let's first consider an analysis between design outcomes for developers of telecom systems, and second for datacom developers.

Table III-3 presents a baseline comparison between MDD and non-MDD designs for telecom.

	Telecom without MDD	Telecom with MDD
Total Project Lines of Code x1000	458.9	464.4
Ave number of SW Developers	13.7	12.4
Ave Project Support Staff	16.3	16.5

Table III-3

As chance would have it, developers using MDD match up very closely for total lines of code, number of developers and support staff. Given the common baseline between these groups of MDD users and non-users, we can objectively calculate the respective associated costs.

	Telecom without Not MDD	Telecom with MDD	Improvement with MDD
Total lines of Code - Project	458.9	464.4	
Months - start to shipment	11.6	9.4	23.4%
Designs Cancelled	14.0%	7.2%	94.4%
Months until Cancellation	3.6	3.7	-2.7%
Designs behind schedule	36.4%	19.7%	84.8%
Months behind schedule	2.3	1.8	27.8%

Table III-4

Looking at telecom designs, Table III-4 shows that MDD designs ship two months faster (for the same code size), have half the cancellation rate than does non-MDD designs, and have nearly half the incidence of designs being behind schedule. For those MDD designs that do fall behind schedule, the number of months behind schedule is less than for the non-MDD designs.

Assuming that the costs associated with software developers (including overhead) is \$10,000/month and the costs of support staff is \$8500/month we can calculate the respective direct costs of development using MDD versus not. These results are presented in Table III-5.

Average Telecom Project Cost	Telecom and Not MDD	Telecom and MDD	Percent Improvement with MDD
Cost of Application Software	\$1,589,200	\$1,165,600	36.3%
Cost of Support Staff	\$1,607,180	\$1,318,350	21.9%
Ave Total Direct Cost of Development	\$3,196,380	\$2,483,950	28.7%
Ave Cancellation costs	\$138,877	\$70,396	97.3%
Ave Late completion costs	\$230,690	\$93,703	146.2%
Total Costs of Software Development	\$3,565,947	\$2,648,049	31.8%

Table III-5

Table III-6 presents the relationship between pre-design expectations and final design outcomes. Design Table IV-6 presents the percentage of final designs that are within 30% of the pre-design expectation.

	Telecom without MDD	Telecom with MDD	Improvement with MDD
Performance Systems	60.0%	73.3%	22.2%
Functionality	55.0%	66.7%	21.3%
Features & Schedule	65.0%	66.7%	2.6%

Table III-6

The savings goes far beyond what is presented in Table III-5. From Table III-6 there is a dollar benefit for better design outcomes – but that number must be predicated on individual company forecasts as to the cost of late delivery and a product that doesn't meet the full feature set anticipated at the start of the project.

Let's now look at datacom developments.

Similar to our telecom example, Table III-7 presents the relationship between Datacom developers using MDD and not using MDD.

	Datacom and Not MDD	Datacom and MDD
Total Project Lines of Code x1000	700.8	695.6
Ave number of SW Developers	8.0	9.2
Team size SW, HW, Support	16.6	21.4

Table III-7

As in the Telecom example, developers using MDD match up very closely for total lines of code, number of developers and support staff. Given the common baseline between these groups of MDD users and non-users, we can objectively calculate the respective associated costs. Note that in this example the support staff for MDD is somewhat larger.

Table III-8 presents a baseline comparison between MDD and non-MDD designs.

	Datacom without MDD	Datacom and MDD	Improvement with MDD
Total lines of Code - Project	700.8	695.6	Same
Months - start to shipment	11.7	10.7	9.3%
Designs Cancelled	14.3%	7.6%	88.2%
Months until Cancellation	4.8	3.0	60.0%
Designs behind schedule	39.4%	23.7%	66.2%
Months behind schedule	2.4	2.7	-11.1%

Table III-8

Once again we see a significant difference between on-time design completions and cancellation rates. Note that even though the cancellation rate for MDD developments is nearly half as those of non-MDD developments, the number of months before cancellation is also significantly less. Now let's look again at the comparative relationship between pre-design expectations and final design results. Table III-8 presents the relationship between pre-design expectations and final design outcomes. Design Table III-9 presents the percentage of final designs that are within 30% of the pre-design expectation.

	Datacom without MDD	Datacom with MDD	Improvement with MDD
Performance Systems	53.2%	72.0%	35.3%
Functionality	53.2%	68.0%	27.8%
Features & Schedule	57.5%	66.0%	14.8%

Table III-9

Summarizing the case for MDD we can note that the data presented to compare MDD development and those developments that don't use MDD was predicated on software application development as well as systems development. In each case, MDD outperformed non-MDD developments.

To further explore the advantage of using MDD we need to compare other aspects of the design, deployment and maintenance components of the product/systems service with Device Software Optimization (DSO) offered by Wind River and Green Hills Software, among others, as well as with Eclipse-based IDEs.

MDD provides the following capabilities that DSO and Eclipse-based IDEs cannot:

- Code reuse
- Easy code redeployment under frequent hardware changes necessitated by processor end-of-life events
- Integrated documentation
- The ability to integrate legacy code into a new format
- The ability to do systems level design – and systems-within-systems design, analysis and support
- Automatic code generation – rapid prototyping

DSO and Eclipse-based tools, however, can provide the following, all of which can be integrated into a common platform using MDD:

- POSIX conformance
- Code certification
- Security: MILS, EAL level 6+, and FIPS 140-2, protected memory
- DO 178 B Level A, ARINC 653, etc.

These are issues of distinct importance for mil/aero, automotive and medical applications.

IV. Why Can't we all Work Together – Creating a Common Platform for C and OO Systems and Software Developers to Co-create

The Special Concerns of the C Developer Using MDD

Even though the concept of MDD can solve many pains through visualization of requirements, model simulation and the ability to produce production quality code from the validated model, there are still some holes within UML and MDD that have been concerns for wide adoption by C developers.

The first concern is that the C developers' natural instinct is to think about the problem in a functional manner whereas most UML and MDD tools force the developer to think and design in an OO environment. Many C developers resist this transition from a C development environment to an OO environment. In cases where the C developer accepts the OO environment, the transition often causes increased ramp-up time for the project.

The second concern is found in the importance that external code plays in the application. Almost all C development today contains some form of external source code. This code may be some special IP that needs to be maintained and reused. This could be existing source code that via reuse will give the project a head start or it could be code purchased from a third party that specializes in developing code for a specific application. The same concern is valid about having to include code generated from a domain specific modeling tool. For most of these cases reverse engineering the code into a model is not an acceptable option.

The third concern is the ability to directly control the generated code. This concern often comes up when a project has real-time concerns or if an algorithm needs to be coded in just the right way to maintain fidelity or if the underlining hardware needs to be directly accessed for performance.

The fourth concern is an ability to debug and fine-tune the actual code running on the target hardware and ensure that all the changes are dynamically reflected in the model.

The fifth concern only applies to C developers targeting highly constrained hardware environments that require very efficient code. In this case they need to be assured that the size of the code will not be greater than the code they would generate by hand or else risk an increase in recurring hardware costs.

The level of comfort, skill transfer and risks associated with transferring from a C development environment to an OO environment is a major challenge and has been resisted by most C developers. A MDD tool supporting the C developer can ease the transition from textual programming to graphical modeling by allowing C developers to work in an intuitive manner in an environment natural to them. C developers typically think in a functional manner as opposed to an OO manner. Extending UML and providing functional modeling constructs based on the C language in addition to the OO constructs, allows modeling in a functional manner. Adding the C constructs of *files*, *functions* and *variables* into the graphical environment makes it easier and more intuitive for C developers when constructing models. It is important that the other MDD productivity enhancers including simulation, production code generation and reverse engineering support these constructs. Using the natural C constructs still brings the benefits of clear design intent and eased communication while enabling team based design.

Some C developers prefer to follow a structured modeling paradigm instead of an OO method. By providing graphically the concepts of Blocks and Flows, which dominate the typical structured methodologies, C developers are quickly able to adopt a UML based MDD environment without having to learn a new methodology. In order to provide the most gains to the C developer, UML tool vendors must support these concepts.

For the C programmer who prefers to work in an OO environment it is important that the MDD tool supports the OO concepts of Class and Object that UML provides for. This allows these developers to work in a similar environment to what the C++ and Java developers who are leveraging MDD to their benefit.

By providing these different design methods and notations the C developers can choose the one most intuitive to them. This overcomes the difficulties of skill transition that occurs when someone is forced to learn and use a new methodology. It also eliminates any risks associated with switching to designing with UML based models for applications. In addition providing a mechanism that allows users to migrate from one method to another or to even work with multiple methods within one design can ease the transition from textual based design to UML model-driven development.

C developers have one other challenge that is often not faced by the C++ developer and that is the need to get the code to look exactly the way they need it to. This can be due to a number of reasons such as having an algorithm that needs to meet very tight timing requirements or the need to directly access the hardware to ensure that registers are properly accessed. The only way to do this is to let the developer write the C code directly for these instances. Some tools force the developer to use proprietary language syntax for specifying actions, which is counterproductive for C developers.

Embedded devices are becoming more wide spread and are increasing in complexity everyday. To deal with this designers of embedded systems are leveraging the benefits of UML based MDD. They are now able to find and correct errors earlier in the process when they are less costly and to develop the systems more rapidly and with higher quality and lower project risk

EMF maintains that the evolution of Model-Driven Development (MDD) now requires that C developers' unique needs be met by UML development tools in order to insure that final designs are consistent with pre-design expectations.

C developers have unique needs for Model-driven development (MDD)

- Functional orientation
- Graphical modeling of C language concepts e.g., files, functions, variables etc.
- Flexible reuse of legacy code
- Code generation for constrained environments
- Need to work close to hardware
- C developers face the same challenges of complexity, productivity and time to market as OO developers

The ability to have C systems engineers working on the same platform as OO software developers is highly desirable and is currently available from at least one vendor. EMF expects that other UML vendors will respond to the needs of this major market opportunity.

V. Using MDD for designs and systems for which frequent hardware changes are expected – how MDA creates solutions available to developers

The embedded marketplace is undergoing unprecedented changes. Vendors are under pressure to accommodate environmentally friendly requirements for boards and backplanes. Moreover, the demand for higher data-transfer bandwidths is resulting in a new, high-speed architecture currently being deployed on desktop computers. This will soon be seen in the embedded marketplace.

The implications for software development, deployment and redeployment are staggering. Every change in peripheral chip sets, driver interfaces, and bus architecture will necessitate a software redesign or design-around effort. Because of these changes, as well as the advent of high-speed serial busses that increase speed from 5 GHz to 40 GHz in increments of 10 GHz every 16 months, software will need to be reconfigured or replenished every 16 months for many applications.

In these turbulent times for hardware configurations and deployments, a major problem arises regarding software. Will drivers need to be written every time new hardware or chip sets are incorporated? The answer is yes. How much of the proven and tested existing software will have to be rewritten? And how disruptive will hardware transformations be for systems that require software modification in order to be redeployed? The answer is a lot!

As PCIExpress is deployed in embedded applications, and as high-speed fabrics are used for enhanced I/O transport, we will see the appearance — and subsequent disappearance — of peripheral chips that support an ever-changing growth in high-speed serial applications. Currently, serial busses are operating in the 2.5 to 5 gbit range. That speed is expected to reach 10 gbit within 20 months — thereby reducing the availability of 5 gbit chips. As the technology transitions to 20 gbit, 30 gbit and 40 gbit capabilities, chips will have to be upgraded in established systems, or new hardware or architectures may be deployed as systems replacements.

So as medical, military, telecom/datacom, automotive and other mission-critical systems need to be upgraded due to hardware and chip availability, how can the (tested) software content be concomitantly upgraded without experiencing extensive delays?

A solution is currently available. The purpose of this paper is to bring the details of the solution, MDA, to the attention of program/project directors so that it can be integrated into the overall plan. MDA is a refinement of the established MDD technologies that specifically address the issues associated with changing hardware/target platforms using the concepts of platform independent models.

These details will address two situations:

- 1) New designs: planning for hardware obsolescence
- 2) Legacy designs: upgrading for current hardware changes while simultaneously enabling easy and rapid future upgrades

Model-driven architecture, a software design approach launched by the Object Management Group (OMG) in 2001 uses models in software development for specification writing and application development. MDA allows designers and developers to separate the

functionality and behavior of the system from the implementation details. Hence, the software model is isolated from the underlying hardware.

With MDA, the application can easily be ported from one environment to another by creating one or more platform independent models (PIM) that can later be translated into one or more platform specific models (PSM). Applicable to a broad range of concepts, MDA is useful in all kinds of software development projects including consumer products, mission-critical deliverables, financial services, healthcare, security services, aerospace and transportation. In order to maximize the benefits of MDA, embedded developers must focus their solutions on both their specific needs, as well as the special needs of real-time performance, compact code, security and specialized hardware control.

The goal of MDA is to create hardware-independent software by separating the specification of the systems operation from the implementation details of how it will use its platform. Because MDA enables portability, interoperability and reusability of the initial application, designs and implementations that will face hardware changes and component non-availability will greatly benefit from MDA. And because MDA is derived from MDD, all of the benefits of MDD apply.

There are additional benefits to an MDA design approach.

Common problems faced by embedded developers include the proliferation of component and distribution infrastructure environments, an ongoing evolution to new source level programming languages, and various modeling standards. How can developers build a system that integrates these disparate technologies while creating systems that will be robust and stable in the years to come as even more new technology comes into use?

MDA addresses these issues through modeling technology, which ensures applications that integrate today and in the future. This is significant when one considers that future systems modifications are likely to occur when the original developers are no longer available. In MDA, one develops a UML platform independent model of his application, which is then mapped into one or a set of appropriate infrastructure and implementation environments, such as CORBA, COM or the RTOS to create a platform specific model (PSM).

MDA separates the specification of the systems operation from the details of how it will use its platform, an approach that increases portability, interoperability and reusability of the initial application. This architectural separation is achieved by:

- Creating a system specification (PIM) without knowledge of the intended execution platform
- Defining the platform to be used
- Transforming the system specification into one for the selected platform (PSM)

To easily retarget the model for different environments, MDA first creates a model that focuses on the functionality and the behavior of the application. The original model is later translated to include the details of the target, RTOS, middleware and communication mechanisms.

When dealing with deployed developments that require frequent hardware or component upgrades, one faces two choices, each of which depend on design complexity, the

nature of the required changes, and whether further changes will be necessitated. These choices are:

- Re-write the drivers to accommodate the required changes.
- Import legacy code into an MDD model and create a Platform Independent Model (PIM).

The first option is much simpler than the second. However, it is frequently desirable to create the PIM using MDD since it permits the software and documentation to be integrated into the model (which is very important for systems that have lifetimes measured in decades — and for which the original developers may have long left the organization). The ability to visualize legacy code improves understanding of complex software, and diagrams enhance communication between developers and non-technical participants.

It is easier to respond to future hardware and component changes with a PIM that can be translated into Platform Specific Models (PSM) when the need exists and the new hardware is defined.

Granted that there is much work to be done by taking the MDD/MDA alternative, but for many designs the short-term effort is well worth the investment.

A better option is to begin one's design using MDD. Then the move to MDA is simple.

VI. Looking to the Future - Gaining Better and Cost Effective System Design Outcomes Using Model Driven Testing (MDT)

Year-over-year, extensive EMF surveys of embedded developers clearly show that embedded designs are becoming more complex while time-to-market requirements are shrinking the time available to both complete designs on-time and meet pre-design expectations. Software testing has emerged as a critical component of the development process – particularly for complex designs when market windows of opportunity are crucial to the monetary success of the product or development.

Recent EMF data has shown that MDT is beginning to emerge as a preferred practice for addressing complex systems. This simply adds to the advantage of using MDD for embedded developments. MDT can't be used without MDD.

The data is clear but not yet as dramatic as is the case with software development design outcomes. It is appropriate to cover the topic conceptually as MDT makes good sense and one can clearly see the benefit. By 2009 EMF expects to have additional supporting data to establish the trend, Figure VI-1 illustrates the traditional use of tools throughout the development cycle.

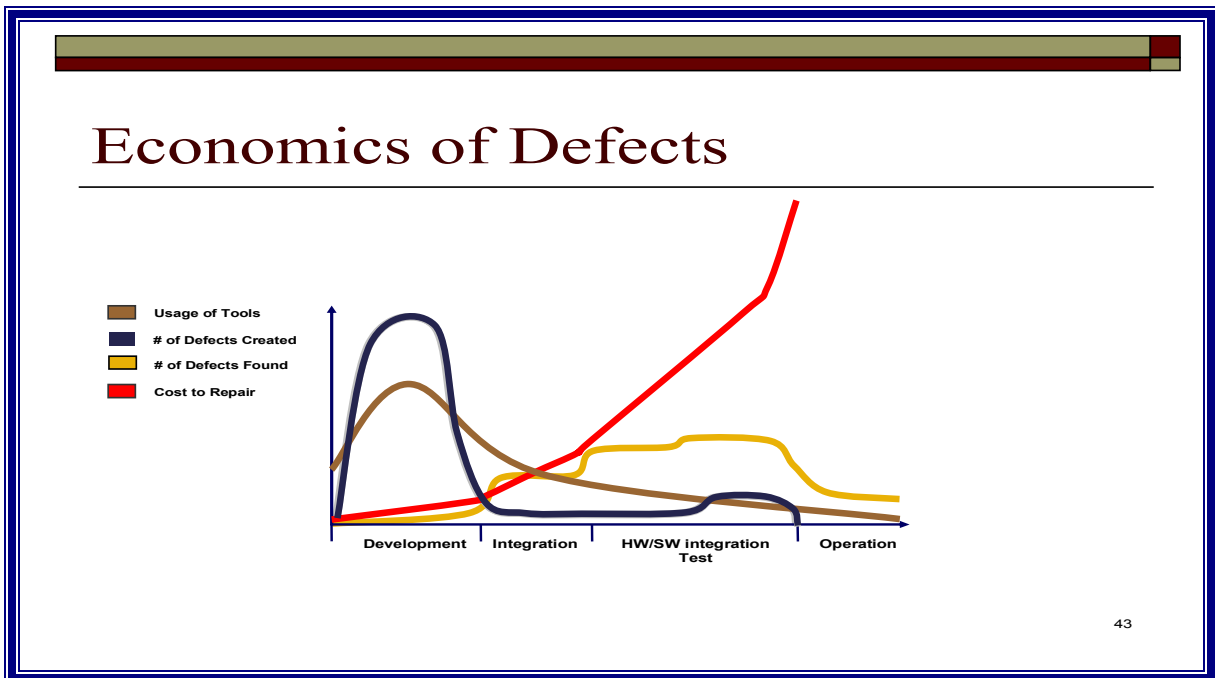
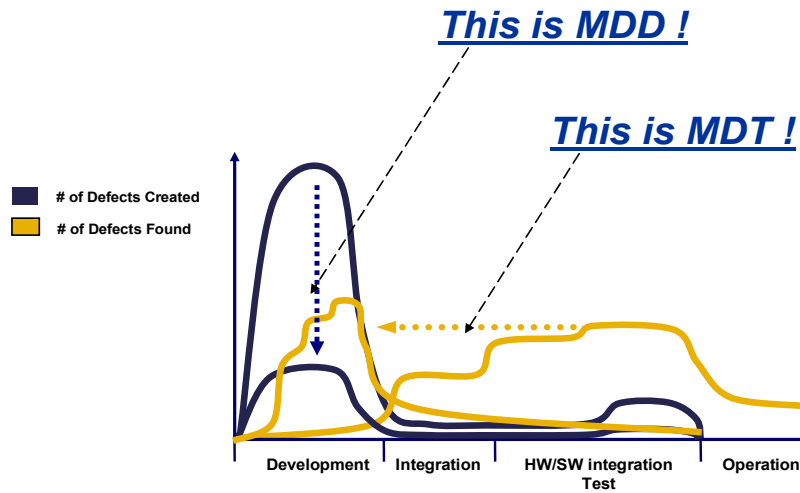


Figure VI-1: Economics of Defects

Figure VI-1 illustrates that the majority of defects are introduced during the development cycle, whereas the majority of defects are detected late in the integration and test cycle. The cost of repair increases significantly as the project moves into later phases.

MDD and MDT to the Rescue!



44

Figure VI-2: Illustrating MDD and MDT

Figure VI-2 illustrates the effectiveness of MDD over traditional development methods as well as the benefit of MDT. If the early data continues to hold true, EMF expects that the combination of MDD and MDT will significantly impact software and systems development, deployment and support.

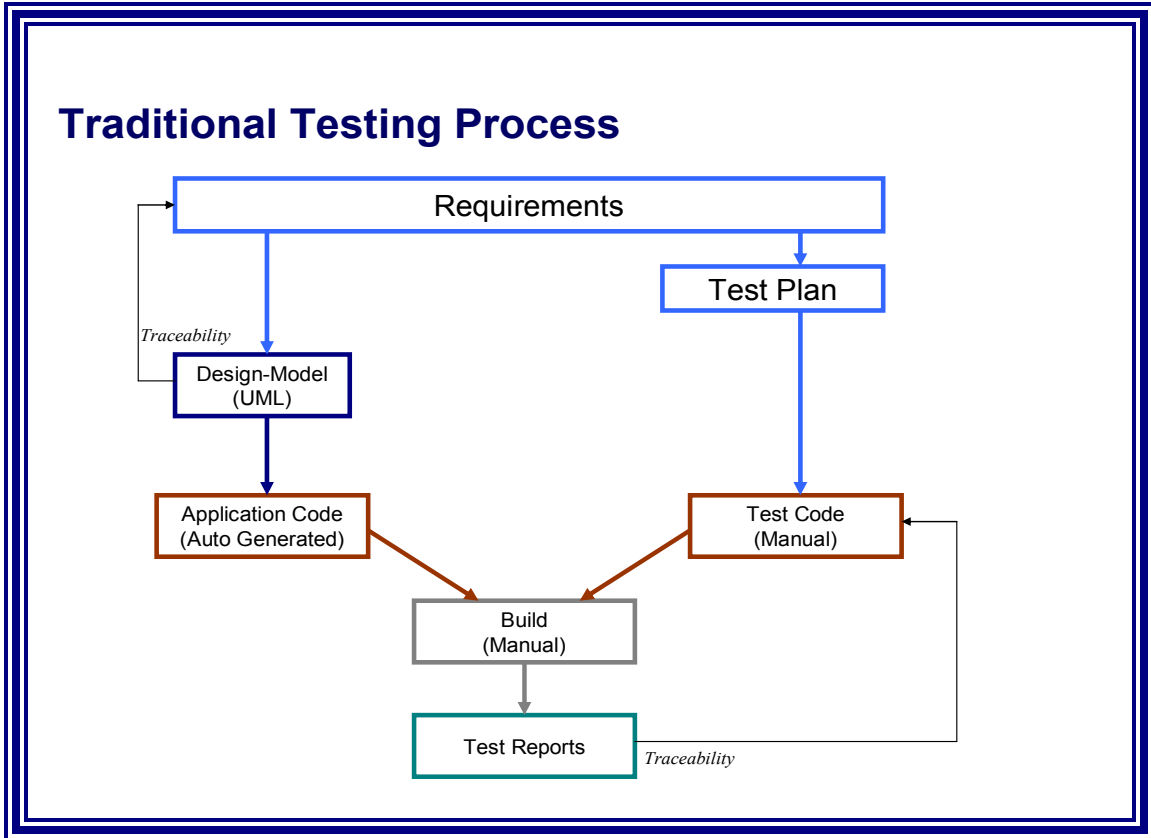


Figure VI-3: Traditional Testing Process

It is clear from Figure VI-3 that manual methods slow down the testing process. As UML is used for increasingly more complex developments, the time required and cost incurred for testing will increase significantly.

Now consider Figure VI-4 which presents the role of modeling in the testing process.

The Role of Modeling in the Testing Process

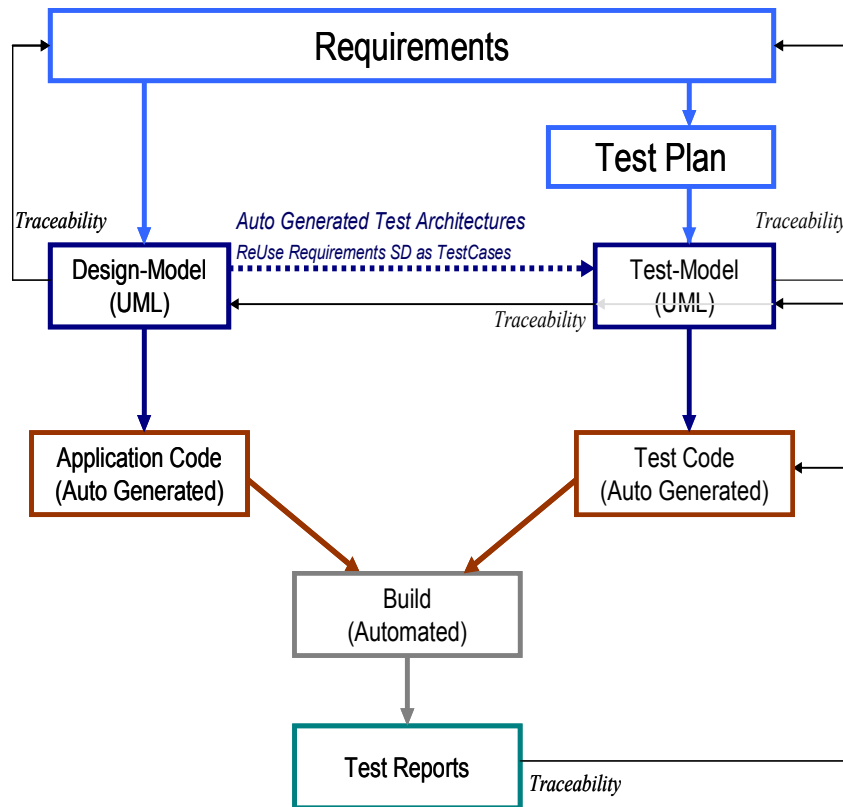


Figure VI-4: Model-Based Testing Process

Here the auto generated test architectures allow the design model to interact with the generated test model creating auto generated test code. Traceability is preserved throughout the design and testing process.

VII. Summary

These are exciting times. We find ourselves at the beginning of a new wave in the history of technology – we now have the tools to address the intricacies and nuances of complex system, and systems-within-systems. Many of our currently deployed complex systems in military/aerospace and in telecommunications, for example, have taken on a life of their own. They are decades old and much of the documentation from the original developers is fragmented or lost. Attempts at backwards compatibility can be frustrating and costly. The legacy of prior selections of technology, interface standards, contractors, and other choices create a powerful incumbency. It can be very difficult to update a system of systems without the cooperation of the incumbent or legacy sources.

The Boeing Corporation has enjoyed far more revenue from the management of the B-52 fleet than from the original procurement of the aircraft.

In telecom, this can be seen in terms of the difficulty of integrating a new system, such as a smart antenna, into existing infrastructure (i.e., base stations). While it is possible to replace a simple component, such as a simple antenna, complex systems with dynamic interfaces and control logic are daunting. This same problem is seen in the integration of new avionics or weapons with a military aircraft. In both cases the incumbent (the base station vendor or aircraft prime) can block nearly any customer action. In addition, in those cases where cooperation with the legacy provider is not good, the customer may find it nearly impossible to understand the pricing and costing of integration.

So how does NASA handle a planned Martian landing in 20 years knowing that their software and systems designs over that period of time needs to be compatible with the hardware and processors and communication protocols that will evolve over that time period? Accordingly, naval ships experience the same problem.

The answer to the telecom dilemma and NASA's program is found in being able to abstract the original design into a hardware independent platform and being able to import legacy software into a MDD model so that it can be modified, updated and redeployed to any new configuration.

This is the future that Model Driven Development (MDD) offers – and it is available today.

In this report, EMF has used real data derived from detailed and statistically accurate surveys of developers to determine design outcomes. EMF research has shown that MDD has emerged as a better design methodology for complex and time restrictive projects.

EMF concludes that MDD not only provides enhanced design outcomes, but that it is uniquely capable of addressing complex systems development, testing, deployment and maintenance. Interestingly, MDD provides a common platform from which RTOS/IDE and tools vendors can grow their markets.

EMF believes that CEOs, CFOs, and senior managers should consider the financial and strategic benefits that can accrue from using MDD, in addition to the benefits that are provided to the design team.

