EMF's Guide for Medical Device Company Executives

How to develop better products, save money, meet FDA/CDRH requirements more efficiently, and avoid recalls, lawsuits, and nasty criminal complaints that can put you in Jail

Jerry Krasner, Ph.D., MBA

March 2009

Embedded Market Forecasters

American Technology International, Inc.

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 has extensive clinical research and medical industrial experience, including the successful filing of more than a dozen 510k submissions.

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. He has been the principal investigator of several NIH funded clinical research programs.

Dr. Krasner was formerly Chairman of Biomedical Engineering at Boston University, and Chairman of Electrical and Computer Engineering at Wentworth Institute of Technology.

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.

EMF's Guide for Medical Device Company Executives

Jerry Krasner, Ph.D.

January 2009

Table of Contents

I. The Challenge

On July 31, 2008 a Senate Bill cosponsored by Senators Edward Kennedy (D – MA) and Chuck Grassley (R– IA) was filed that would require senior officers or directors of drug and medical device companies to certify under penalty of perjury that all information submitted for a product's approval is accurate and in compliance with federal regulations.

The Drug and Medical Device Accountability Act Bill expired at the end of the two year Senate session on December 31, 2008, but is being refilled for the new Senate session.

The Bill provided that product applications later found to have contained false or misleading information would be subject to stiff fines (up to $5,000,000), assessed both to companies and their senior officers, who, in addition, could face jail sentences of up to 20 years.

EMF recommends that medical device CEOs advantage themselves by addressing the impact and consequences of the anticipated new law now before they are forced to do so under a less favorable time frame. Strategically, the best approach that a medical device company can take is to utilize the best technology that the industry has to offer – an approach that includes:

- Requirements Definition and Management
- Change and Configuration Management
- Quality Management/Testing – Software Verification and Validation Tools
- Modeling
- Release Management
- Documentation
- Team Collaboration

Fortunately for medical device company executives, a currently available technology exits that can integrate all of the above mentioned technologies in an efficient and comprehensive manner. This technology, known as Model Driven Development (MDD), can be integrated into a company's exiting development processes and use their existing tool sets. MDD can integrate other developmental, testing, validation and tracking tools provided by many embedded vendors.

Currently, MDD is the most effective strategic approach to dealing with the entire design and deployment process. Most importantly, to executives who must sign off on the Drug and Medical Device Accountability Act, it provides the best assurance that what they represent is factual based on an accurate and fully documented trail. This paper is intended to present a roadmap illustrating how executives can be in compliance with the Bill. In addition, by utilizing MDD, companies can increase their ROI and significantly reduce the chance of a product recall or a patient injury. EMF has created a second paper that details a step by step description of how developers can integrate these processes into their current development programs, and can be located and downloaded at www.embeddedforecast.com .

II. Creating Value by Adopting Best Practices for Medical Device Developments

Let’s examine ways in which companies and their senior officers and directors can protect themselves against inadvertently certifying information that could be viewed as misleading or inaccurate.

On May 11, 2005 the CDRH issued a non-binding “Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices”. Given the potential consequences of improper filings contained in the Drug and Device Accountability Act of 2008 it would constitute a best practice for any CEO to embrace the guidance provided by the CDRH.

1) Risk assessment and management is a very important part of your filing with the CDRH and can be your best approach for protecting you under the Act. Be exceedingly careful in documenting the “Level of Concern” section of your application. It would be wise to include systems failure documentation that is available through modeling or formal methods for certification. Make sure that you specify the correct level of concern and document your approach to rectifying potential failures.
2) Submit a Device Hazard Analysis for all software devices – include all hazards (hardware and software) associated with the products intended use. This can include the user GUI and how it might be operated by personnel who work under stressful conditions.
3) Submit a Software Requirements Specification (SRS) that includes functional, performance, interface, and developmental requirements for the software, including hardware, OS and programming language requirements.
4) Include an Architecture Design Chart (flowchart or similar illustration) that describes the relationships among the major functional units in the software device. There should be sufficient information to allow for the organization of the software relative to the functionality and intended use of the software device.
5) The software design specification should present information to demonstrate that the work performed by the software development engineers was clear and unambiguous, with minimum ad hoc design decisions.
6) Submit a summary of the Life Cycle plan and the Life Cycle processes employed. It will be useful to include an annotated list of the control/baseline documents generated during the software development process, and a list or description of software coding standards.
7) Include verification and validation documentation and base it on the claimed Level of Concern. Whenever software is changed, a validation analysis should be conducted to validate the specific change and also to determine the extent to which this change may impact the entire systems operation. Documentation and tracking is essential.
8) Include a revision level history of software revisions generated during the course of product development.

III. Recommended Best Practices that your company should consider

1) Take advantage of modeling technologies. Choose a development platform that can easily integrate testing and management tracking information, easily upgrade legacy product code when improvements are added, or when underlying hardware has changed. Model Driven Development (MDD) technology is not only

effective for meeting this need, but EMF data shows that the expected ROI derived from using MDD is significantly higher than that for development methods that don't employ MDD.

2) Reuse Your Existing Assets: MDD offers code visualization and "reverse engineering" capabilities which in essence provides a company credit to the CDRH for pre-existing software without having to start from the beginning again. Your legacy code can be imported to a MDD model and redeployed as a product upgrade, or it can be redeployed to a new hardware platform.
3) Validate Design Requirements Early: MDD is particularly useful in this regard. This will reduce development costs and development time from design start to shipment (EMF data year-over-year confirms this).
4) Automated testing, the ability of your software to automatically generate test parameters, examine outcomes and automatically document test outcomes, should be considered as a best practice.
5) Traceability must be complete and audited to protect the CEO under the act. This should be considered a good design practice. However, it is important to initiate traceability early in the development process in order to gain management efficiencies. Consider automating traceability as a best practice.
6) Document third party code -know what's in your application even it's not yours: Software provided by a third party for which adequate documentation may not exist is known as Software of Unknown Pedigree (SOUP). As a CEO you can follow two paths: explain the origin of the software and the circumstances surrounding the software documentation, or; use MDD to import the SOUP and apply rigorous testing and validation analysis to it. Some MDD tools allow SOUP to be either integrated into the product or to be treated as a separate legacy component for which analysis can verify and validate systems operation. Formal code certification tools can also be used to ensure that SOUP code is safe.
7) Initiate and document training programs participation for all areas of development, testing, deployment and support – BEFORE submitting the 510k application.

Summary

In order to assist companies to be compliant with the tenants of the new Drug and Medical Device Accountability Act, EMF has set out a series of guidelines and recommended best practices for medical device executives to consider. EMF has strived to create a strategic awareness of how currently available technology can be used to address issues pertaining to the law as well as to use the same technology to create value by reducing costs and enhancing design outcomes and time-to-market issues.

In the following appendices, EMF lays out supporting information to acquaint medical device executives with the technologies that can be employed to address the issues raised by the new law as well as supporting data to show the advantages of using MDD.

Executives should be delighted to see that they don't have to change development and testing tools already in use, or change established processes in order to implement MDD into their development, testing and documentation efforts.

EMF will release a more technical paper for medical device developers that will detail, step-by-step, how to implement these procedures.

APPENDIX A

CEO's Guide to Essential Components for Systems and Software Development

Within the "systems and software development domain" EMF identifies 6 essential components which are included in our assessment of systems best practices:

- Requirements Definition and Management
- Configuration Management
- Quality Management/Testing – Software Verification Tools
- Modeling and Architecture
- Release Management
- Team Collaboration

Requirements Definition, Traceability and Management:

Requirements Management tools are used to define, prioritize, and maintain requirements for software systems and applications. In many cases when developments fail to meet pre-design objectives, it is because requirements are not clearly defined or clearly tracked to ensure that developments stay on course. Key to this is the ability to analyze and maintain changes to such requirements. Requirements traceability is also included under requirements management. It is necessary to be able to trace each requirement back to its origin and document every change made during the course of the development. Being able to trace architectural, design, test and implementation elements back to the requirements that spawned them is important.

Sophisticated requirements management tools have been available for a long time, and apparently haven't been sufficiently employed by military contractors or the military. An overview/analysis of such tools is certainly useful to systems developers, senior managers, CEOs and CFOs.

High Rely, Inc., a Phoenix, AZ based certification and software consulting company, has published an excellent detailed evaluation and subgroup breakout of Requirement Management and Traceability tools. Their comparative analysis includes configuration management and documentation subcomponents.

EMF is a leading market analysis organization and a supplier of fact-based survey data that reflects market conditions, developer preferences and measurable design outcomes. EMF does not evaluate competitive products. High Rely is a respected software consulting/development and certification organization (e.g., DO-178B certification, CDRH 510k) and is very qualified to make such comparisons. The following is High Rely's competitive evaluation published herein with their permission. EMF believes that this breakout and assessment is important for readers of this report, and EMF is grateful for High Rely's cooperation.

High rely specifies the attributes of these tools that they deem most important in order to achieve traceability:

- Ability to classify requirements
- Requirements relationship with user-defined attributes

- Advanced filtering, grouping, sorting for reports and documents generation
- Automatic change management
- Import/export tools for interfacing with other tools and applications
- Unique traceability tools for impact and gap analysis
- Bottom-to-top and top-to-bottom traceability
- Availability for common tagging schemas to be employed, while also allowing for customized tagging schemas

High Rely breaks out requirements and traceability tools with the following groups and subgroups that they deem important:

- **Capturing Requirements/Identification**
    - Input document enrichment/analysis
    - Input document change/comparison analysis
    - Automatic parsing of requirements
    - Interactive/semi-automatic requirement IDs
    - Batch mode operation

- **Identify inconsistencies: If so what kind of …**

- **Configuration management**
    - Access control (modification, viewing, etc.)

- **Documents and other Output Media**
    - Standard specification output (if so, what kind)
    - Quality & consistency (spelling, data dictionary)
    - Presentation output
    - Custom output features & markings
    - WYS/WYG (what you see is what you get) previewing of finished output
    - Status reporting

High Rely developed a capability comparison matrix for requirements management tools in these categories, which is presented in Table A-I. For each of the above characteristics they list each tool according to:

- ➢ Fully provided (F)
- ➢ Partially provided (P)
- ➢ Not provided (N)

The following tools are presented in Table A -1

- CRM – CaliberRM
- IBMR - IBM Requisite Pro
- RaQ - RaQuest 2.4
- Cont - Contour
- DCSE - Dassault CSE
- DOOR – IBM -Telelogic DOORS
- Rely - RelyTRACE

	CRM	IBMR	RaQ	Cont	DCSE	DOOR	Rely
Capturing Requirements/Identification	F	F	F	F	F	F	F
Input document enrichment/analysis	F	F	F	N	F	F	F
Input document change/comparison analysis	P	F	F	N	P	F	P
Automatic parsing of requirements	F	F	N	F	P	F	P
Interactive/semi-automatic requirement IDs	F	F	P	F	F	F	F
Batch mode operation	F	F	F	F	N	F	F
Identify inconsistencies: If so what kind of …	F	F	F	P	F	F	F
Configuration management	F	F	F	F	F	F	F
Access control (modification, viewing, etc.)	F	P	F	F	F	F	N
Documents and other Output Media	F	F	F	F	F	F	F
Standard specification out put (if so, what kind)	F	P	F	P	P	F	F
Quality & consistency (spelling, data dictionary)	F	P	P	F	F	F	N
Presentation output	F	P	F	F	P	F	P
Custom output features & markings	P	F	N	F	F	F	F
WYS/WYG previewing of finished output	F	F	P	F	F	F	F
Status reporting	F	F	F	F	F	F	P

Table A-I: Requirements & Traceability Tools (source High Rely, Inc., with permission)

Configuration Management

Developers need a system that enables them to keep track of reported software changes and support the ability to track defects throughout the entire software lifecycle. In addition, developers and managers need to be able to manage multiple versions of the same unit information. Such tools fall under the name of revision control, source control or source code management (SCM). Traditional configuration management and revision control systems models used a shared server for all functions which ran the risk that if multiple developers were using the same file at the same time they might wind up overwriting each other's data. These shared control systems solved the problem by either using a file locking method or a version merging model.

High Rely recommends in their evaluation of configuration management and revision control tools that developers use a SCM tool that smoothly provides graphical differencing and merging – but they also state that there is a need to take manual steps to ensure data integrity and avoid conflicts.

Table A-II presents High Rely's assessment of configuration and change management, and revision control tools:

- **SVN – Subversion**: Best choice for small companies
- **AllChange – Intrasoft**: Flexible and customizable
- **CM Synergy – IBM - Telelogic**: One of the best choices for medium to large size projects
- **ClearCase – IBM**: One of the most popular choices for medium to large size projects
- **Perforce**: Not as powerful as ClearCase for the price
- **PVCS – Merant**: Offers basic support for configuration management
- **Razor – Visible Systems**: Need attaching shell scripts both before and after Razor events
- **Visual SourceSafe – Microsoft**: Offers basic support for configuration management
- **RCS Pro-Component SW**: Offers basic support for configuration management

Table A-II

Quality Management/Testing – Software Verification

There is a need to improve verification efficiency. There are many sources of changes in the software, ranging from bug fixing, to function improvement or the introduction of new functions. When something has to be changed, all products of the software life cycle have to be updated consistently, and all verification activities must be performed accordingly.

The level of verification for certified safety-critical software is much higher than for other non-safety-critical software. For level A, DO-178B avionics software, the overall verification cost may account for up to 80% of total costs. Verification is also a bottleneck for project completion. So, clearly, any change in speed and/or cost of verification has a major impact on the project time and budget.

There is a difference between software verification and software validation. Verification testing is used to insure that the software conforms to its specification. Validation testing is customer-centric and is used to provide assurance that the software relates to what the customer requires.

Under verification tools, High Rely lists the following tools with comments in Table A-III.

- **VectorCast** – avionic specific
- **GCover** – Green Hills integrated path coverage
- **IPL** – Traditional coverage
- **IBM - Telelogic Logiscope -** Traditional coverage
- **LDRA** – modern analysis, strong in Ada
- **IBM - RationalTest RealTime** – full suite, structural coverage
- **Coverity Prevent SQS** – resolve critical defects in C, C++ and Java
- **PolySpace** – verify C, C++ and Ada

Table A-III

High Rely broke down their analysis along the lines of:

- Static analysis tools
- Functional verification tools
- Structural coverage tests

As High Rely is expert in certifying to the DO-178B process, their focus is towards such certification. This expertise can be applied to medical devices as well.

In EMF's 2008 survey of 455 embedded developers, the data showed that RationalTest RealTime was the most familiar and most used testing tool by respondents (33.5% and 22.4%). Vector Software VectorCast/C/C++/Ada was second among respondents (22.2% and 16.3%).

Modeling and Architecture - Model Driven Development (MDD)

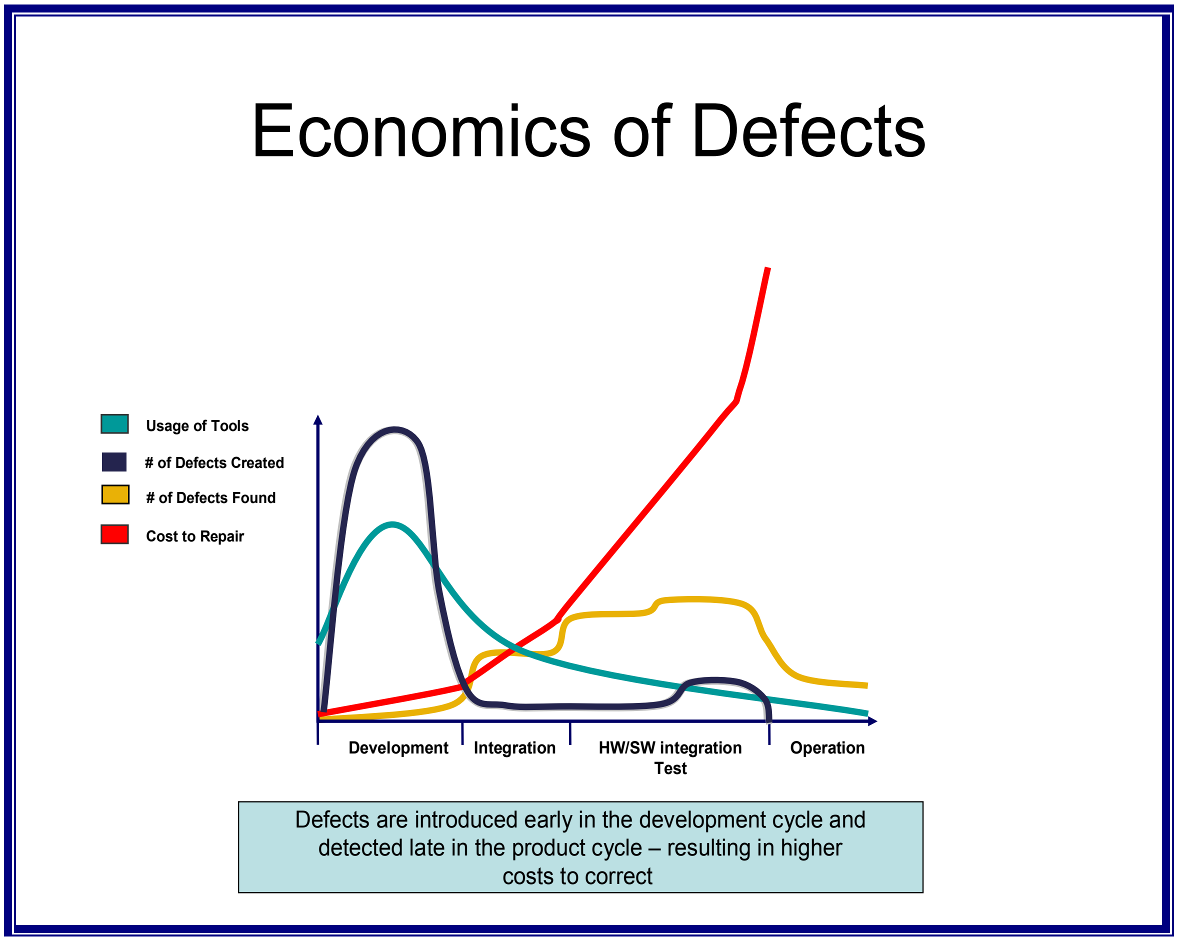
Annual surveys by Embedded Market Forecasters (EMF) of embedded developers have shown that software development is responsible for more than 80% of design delays and associated design complications. This data also reports on embedded developer responses to design complications. When asked how close their final design was to pre-design expectations (for performance, systems functionality, features and schedule) approximately 40% of respondents indicated that their final design was NOT within 20% of their pre-design expectation. This problem takes on greater proportions when it becomes the reason for system delays and systems design failures.

Whether the system is poorly conceived, specified or whether crucial algorithms fail to adequately address systems performance, traditional methods of embedded software development are yielding to a process known as MDD. MDD is used to more clearly define design specifications, test systems concepts and to automatically generate code and documentation for rapid prototyping as well as for software development.

Moreover an MDD platform can be used integrate a company's exiting tools and processes. EMF suggests that MDD be made an essential component of a medical device development program.

One of the major advances in software engineering design has been the use of the Unified Modeling Language™ (UML®) for enabling embedded design efficiencies. Pioneered by companies including TogetherSoft, BridgePoint, IBM/Rational, and Telelogic (now IBM) for embedded and/or real-time applications, its value is the ability to address very complex designs and (with certain commercial offerings) the ability to go from State Diagrams to source code (automatic code generation). In addition, UML offers code reuse, legacy upgrades and the ability to re-port software to changing hardware configurations. This is of huge importance given the frequent end-of-life changes in chip availability.

Figure A-IV illustrates the traditional use of tools throughout the project development cycle, from design to deployment.



Economics of Defects

Defects are introduced early in the development cycle and detected late in the product cycle – resulting in higher costs to correct

Figure A-IV: Economics of Defects

Figure A-IV 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.

Figure A-V illustrates the role of MDD and model driven testing (MDT) in the development cycle. MDD permits most defects to be detected early in the design cycle.

MDD and MDT

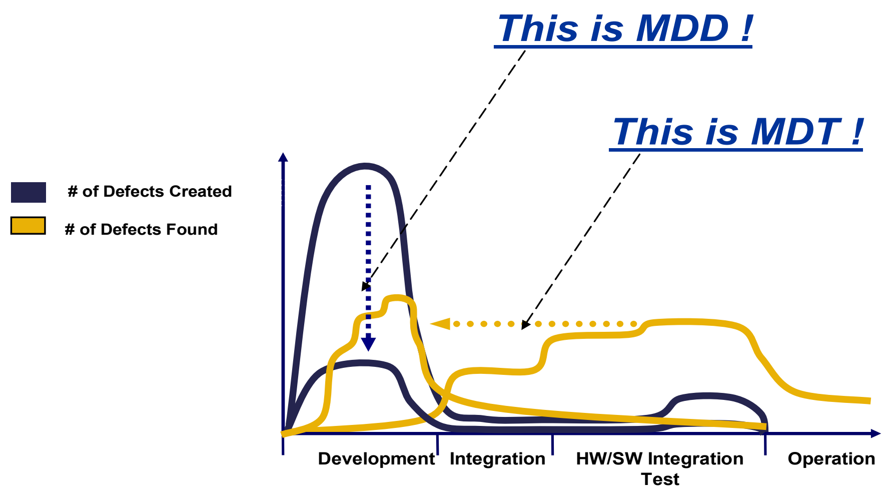


Figure A-V: Illustrating MDD and MDT

Figure A-V illustrates the effectiveness of MDD over traditional development methods as well as the benefit of MDT. EMF expects that the combination of MDD and MDT will significantly impact software and systems development, deployment and support.

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

- Design 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
- Ability of C (structured) and OO developers to work on the same system within their respective familiar GUI

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 and team-wide collaboration
- Easy code redeployment under frequent hardware changes necessitated by bus and processor end-of-life events
- Automatic/Integrated Documentation for design history file
- Graphical Design Reviews
- Early Validation
- Executable Designs

EMF data has shown that:

- The use of MDD tools by embedded developers has reduced design delays and cancellations.
- The use of MDD tools by embedded developers has significantly improved the relationship between pre-design expectations and final designs.
- UML is the most popular graphical representation for simulation-modeling tools for discrete embedded system designs.
- UML, simulation and code generation enable faster design iterations that produce desired performance, functionality and capabilities.
- Using UML, simulation and code generation, design cycles are more predictable and result in faster product shipments with lower project risk.
- UML, simulation and code generation contribute significantly to a reduction in design, development and implementation costs.

Release Management

There is a need to close the gap between writing code, building an application, and releasing it into production. Release Management is a rapidly growing discipline within software engineering for managing software releases. Software products are typically in an ongoing cycle of development, testing and release. Add to this an evolution and growing complexity of the platforms on which these systems run, and it becomes clear there is a need for a process that seamlessly integrates all aspects of the development process to insure the success and long-term value of a product or project.

CDRH's CFR 21 Part 11 of the Code of Federal Regulations deals with guidelines on electronic records and electronic signatures Part 11 defines the criteria under which electronic records and electronic signatures are considered to be trustworthy, reliable and equivalent to paper records. Support for CFR 21 Part 11 is suggested.

The better release management tools automate the software delivery process through an adaptive build and release management framework to enable development teams to standardize on repetitive tasks, manage compliance mandates, and share information through a centralized web interface accessible 24/7 worldwide. These tools should be able to correlate data from disparate source control, testing and defect tracking tools to provide a coordinated view of product development.

Although project managers have done this in the past, an automated process is desirable to insure that details are not overlooked thereby enabling these managers to be more productive.

Team Collaboration

We have come a long way from the old ways of development – particularly systems and systems-of-systems developments – that were predicated on local development teams being able to interact (assuming that they employed some measures of requirements management practices). Today, development teams span the globe and are inter-operated across many time zones. The need for a team-collaborative infrastructure is certainly an important best practice.

An example would be a platform that brings the tool integration qualities Eclipse allows individuals to the team level.

Conversely, one can be part of a small, local team divided by differing work schedules or department affiliations. In either case, you collaborate with subject matter experts separated by time, distance or organization. At issue is to determine the type of infrastructure that would provide the best work environment for the individual developer, the small group and the larger collaborative team.

In order to optimize the development lifecycle, it is important to closely link people, best practices, processes and tools. This makes sense since process improvement enhances effectiveness while development tools increase efficiency.

APPENDIX B

CEO's Guide to Cost Containment and Enhanced ROI

Example: Comparative Telecom Equipment Development ROI Analysis

In a previous paper (*The Economics of Embedded Development, December 2008*), EMF compared similar telecom development programs that used MDD with those that didn't. Parts of that paper are reprinted here as an example to illustrate for medical device companies an additional reason why MDD should be considered.

This real world example of a comparative ROI evaluation is presented for Telecommunications equipment developments based on EMF's 2008 detailed survey of embedded developers (455 respondents).

The 2008 EMF Survey data can be used to compare direct costs between MDD-based telecom developers and non-MDD telecom developers (developers using traditional hand coding). EMF conducts statistically accurate surveys of embedded developers, who answer more than 70 detailed questions that provide insights to what they use, how long it takes them to get a product to market, how close to their final design is to their pre-design expectation, etc. Our data presented herein is predicated on the responses of 455 developers.

Table B-I presents:

- Total number of lines of code – including written, reuse and open source
- Number of software developers on design – and number of other developers and support staff.

	Telecom and Not MDD	Telecom and 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 B-I

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.

Given the common baseline between alternatives using these parameters, we can then look to the following for comparative insights.

- Time taken from design start to actual shipment date

- Percent of Designs Cancelled
- Average number of months before cancellation
- Percent of designs completed behind schedule
- Average number of months behind schedule

If there is a compelling reason to replace standard telecom design methodologies with MDD, we should be able to see significant improvement in the closeness of the final design outcome to the pre-design expectation.

Table B-II presents a baseline comparison between MDD and non-MDD designs.

	Telecom and Not MDD	Telecom and MDD	Improvement with MDD
Total lines of Code - Project	458.9	464.4	Same
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 B-II

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 B-III.

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 B-III

The calculations are straight forward. The Cost of Application Software equals the number of project months from start to shipment multiplied by the number of software developers multiplied by the monthly cost per developer ($10,000).

The calculation for cancellation costs and late completion costs equals the number of software developers multiplied by the monthly cost/developer plus the number of support staff multiplied by the monthly cost/support personnel. This number is multiplied by the number of months it takes before a project is cancelled (or for late completions the number of months the project is late) multiplied by the percentage of project cancellations (or late completions).

The comparison between MDD use and no MDD use is significant – and the fact that the projects matched up nearly exactly makes the conclusion more significant.