WHITE PAPER

Best Practices for IEC 62304 Compliance with Model-Based Design

MathWorks®

Introduction

For medical device engineers, complying with the IEC 62304 safety standard often involves document-based requirements, hand-coding, and prototyping on physical devices.

Model-Based Design provides a faster, more cost-effective approach to creating high-integrity software for medical devices. At the center is a system model that spans requirements development, architectural analysis and specification, detailed design, implementation, and testing. You refine your design through simulation and rapid prototyping, then generate code and implement on embedded devices.

This paper will discuss how Model-Based Design with MATLAB® and Simulink® can be used in a process that is compliant with IEC 62304, particularly the higher severity safety class levels B and C. The paper is organized following the main activities identified in the IEC 62304 standard.

Terms

The following definitions will be used throughout the paper:

- *Verification*: confirmation through objective evidence that software development activities meet required outputs
- *Software unit*: the smallest software component that is designed, implemented, and verified
- *Software item*: a software component that may comprise one or more software units; typically, it is something of intermediate size, but also is used in reference generically to include anything from a unit to a software system
- *Software system*: a package of software containing one or more software items; it represents a complete package of functionality, but perhaps only a component of a full medical device
- *Subsystem*: a grouped component of a Simulink model, usually containing multiple basic or intrinsic blocks

The Software Development Process

Software Development Planning

The software development process described by the IEC 62304 Standard, section 5.1, begins with a plan for how the major activities and tasks will be accomplished during the project. The Standard recommends developing several different constituent plans, covering all aspects of the development process. An organization that adopts Model-Based Design should reference such tools as part of the development strategy in these planning documents. It's also important to include the artifacts generated during that process in the list of deliverables for the software process.

Some typical items that may be created include MATLAB® scripts and functions, Simulink models, data dictionaries, generated production code, S-Functions and other user block libraries, simulation input data (test vectors) and results, and generated documentation such as design documents and test reports. Industry standards, such as the MathWorks Automotive Advisory Board (MAAB) community's guidance for Simulink modeling, should be considered for applicability to the product under development. In addition, the IEC Certification Kit provides a collection of model design standards and tools to check compliance to standards such as IEC 62304.

Many organizations start from such a set of model design standards and extend or tailor them based on individual or organizational needs. In addition to model standards, IEC Certification Kit also provides a reference workflow and tool validation documentation that were evaluated by the TÜV SÜD. It found that Model-Based Design tools from MathWorks are suitably validated for use in safety-related development according to IEC 62304. Finally, note that the development tools themselves should be managed and version-controlled as part of the software configuration.

Software Requirements Analysis

A significant benefit of Model-Based Design is that it enables designers to better understand system and software functional requirements. By representing the initial design ideas as live, executable models, the consistency and correctness of requirements can be established in a much more concrete way than by analysis of traditional textual requirements documents. In addition to the ability to model a software item's behavior, Simulink and associated physical modeling tools in the Simscape™ family can model the environment with which the item interacts, including software algorithms, electromechanical components, and patient physiology. Simulation allows a software system model to interact in a deep and meaningful way with its relevant operational environment in order to elaborate and help verify functional requirements. Alternatively, or in addition, models can simulate based on laboratory, clinical, or analytically defined datasets. Requirements that are used to derive Simulink models are inherently testable via simulation.

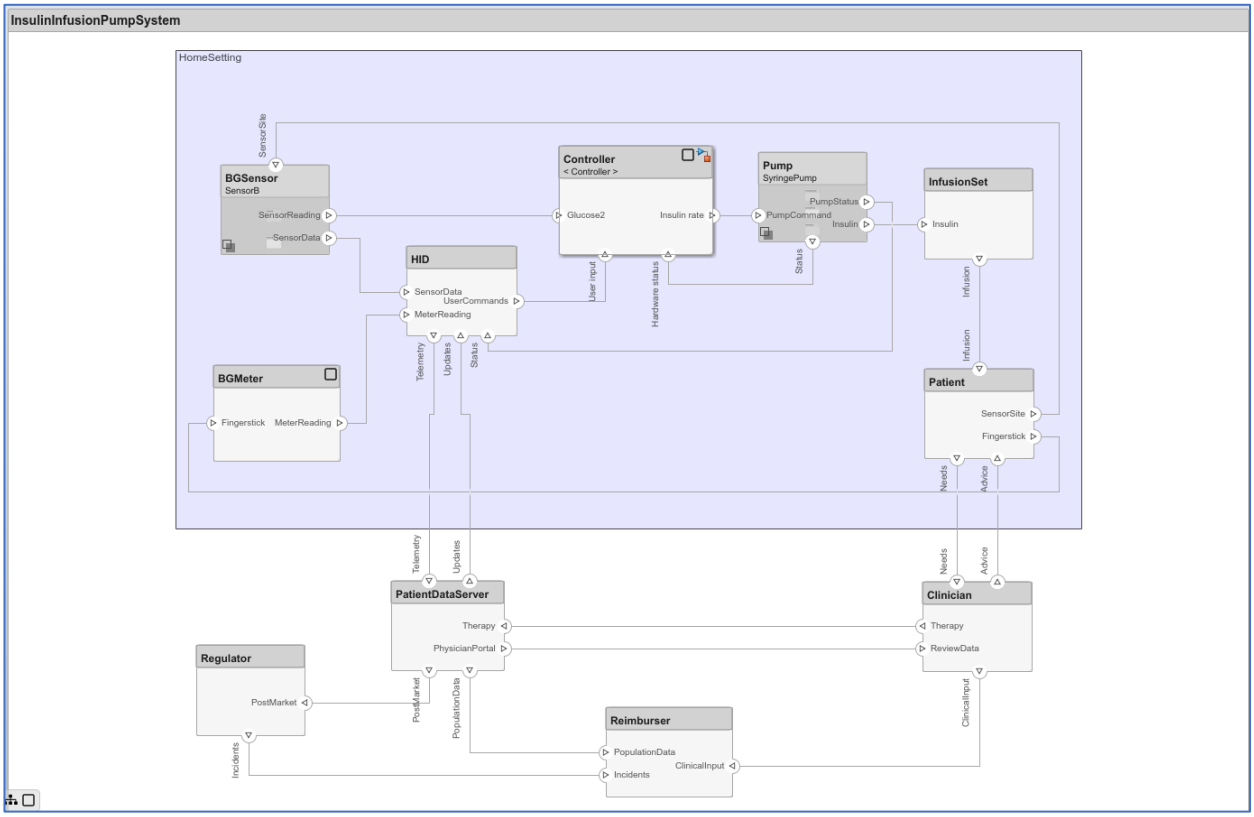
Requirements Toolbox™ provides a linking facility to establish traceability between requirements and components of models, such as Simulink blocks and Stateflow® chart elements. Reports generated from Simulink can support traceability analysis. The Standard

requires special attention for those software items that protect from harm. Such risk control measures will have associated requirements, and like all requirements, those requirements can link to elements of the Simulink model. A best practice is to tag risk control requirements with a unique keyword to use requirements and reporting filter capabilities to facilitate risk control analysis.

Software Architectural Design

According to section 5.3 and annex B.5.3 of the Standard, the goal of software architectural design is to divide the software into major structural components, indicate their external properties, and demonstrate the relationship among the components. System Composer™ enables the definition, analysis, and specification of architectures and compositions for model-based system engineering and software design. System Composer can be used to establish the system and software components, specify their interfaces and dependencies, and allocate and trace system and safety requirements to these components.

One application would be to separate risk control measures into a component or subcomponent to more intuitively track risk control measures and potentially lower the software safety class for other parts of the software architecture.



Example of system architecture in Simulink. Controller is linked to a Simulink behavior model representing a software item.

Requirements Toolbox can relate functional requirements and inputs and outputs to representative constructs in the System Composer model. Reviewing traceability reports can help support the verification of the software architecture against the software requirements.

A common scenario is that the path from Simulink models to implementation (for example, production code generation from Simulink models via Simulink Coder™ and Embedded Coder®) will be used only for portions of the software system. There may be very practical reasons, such as the existence of low-level driver code from a third-party component manufacturer or a desire to reuse existing source code from a similar product or product line. Model-Based Design supports modular implementation of software units or items. A Simulink model representing the complete software system design can also be fully implemented.

Graphical representation of the software architecture is a helpful means to verify as well as express it. Furthermore, interface consistency can be enforced with stereotyping—essentially reuse of a type of interface—for the data shared between components. The architecture can be reviewed interactively in System Composer or by review of documentation generated from the model.

While the Standard recommends review as the primary architecture analysis technique, Model-Based Design allows the opportunity to verify that the software architecture meets functional requirements through simulation. The architectural components specified in System Composer can be linked to executable behavior models in Simulink, and a system-level Simulink simulation can be created to execute the system architecture or subsections of the same. Such verification is always welcome earlier in the software development process, as the cost of defect resolution has been reported numerous times to be a strongly increasing function of the stage in the software development life cycle during which the defect was detected. Furthermore, coverage analysis (for example, modified condition/decision coverage) of the portions of the models that were executed during functional verification can reveal incomplete requirements or unnecessary design elements, both of which are less expensive to resolve earlier in the development cycle.

Software Detailed Design

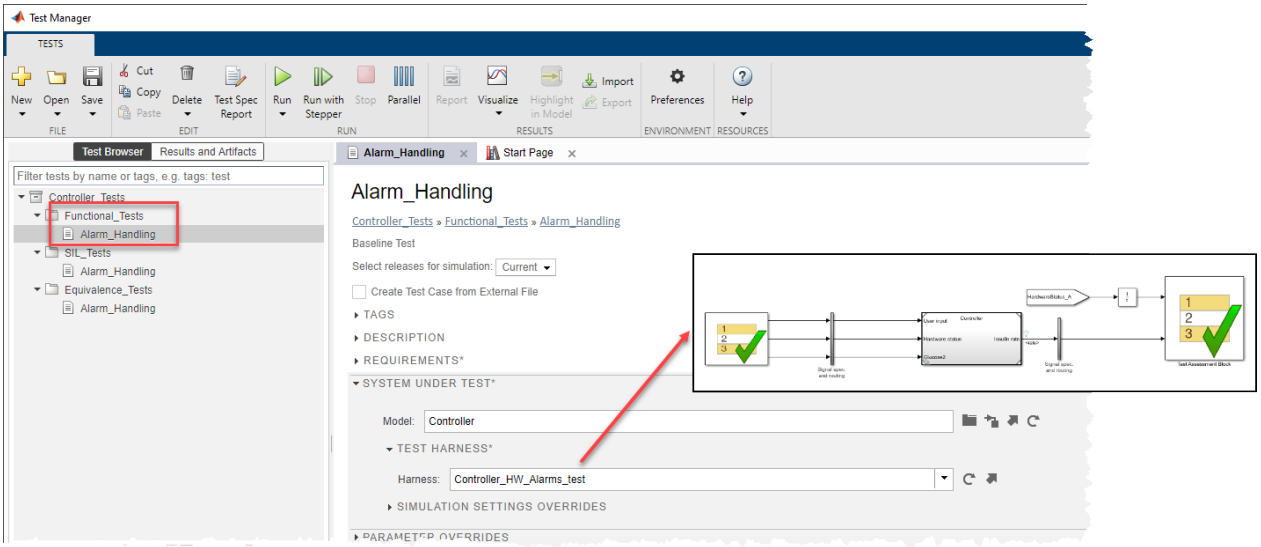
Software detailed design can be accomplished in Simulink by creating a hierarchy of behavior models and Subsystems to elaborate a software architecture. If the architecture is expressed in System Composer, the two tools integrate in order to import the components and interface specifications and link the architecture and behavior models. Such a link supports software system design verification activities, where consistency of the software design with the architecture must be proven.

Software items are further subdivided until the unit level is reached, once again revealing the structure, properties, and dependencies between the units. Software safety class C requires a documented, verified design for each software unit. This design must specify all aspects of the unit necessary to support software implementation. The Simulink model or Subsystem that represents the software unit can be an important part of this detailed design. The model clearly shows the software unit’s interfaces and relationship to other software units and items, not only revealed by the block diagrams but also demonstrated by simulation capability.

The principles of verification of the software architecture described above can be applied to a further elaborated Simulink model containing detailed design information for safety class C software units. Software unit design verification must demonstrate that the unit meets its requirements. Links can be traversed from model to requirements documents and vice versa and synchronized when changes occur. Reports and highlighting can demonstrate the degree of requirements link coverage in a Simulink model to support verification.

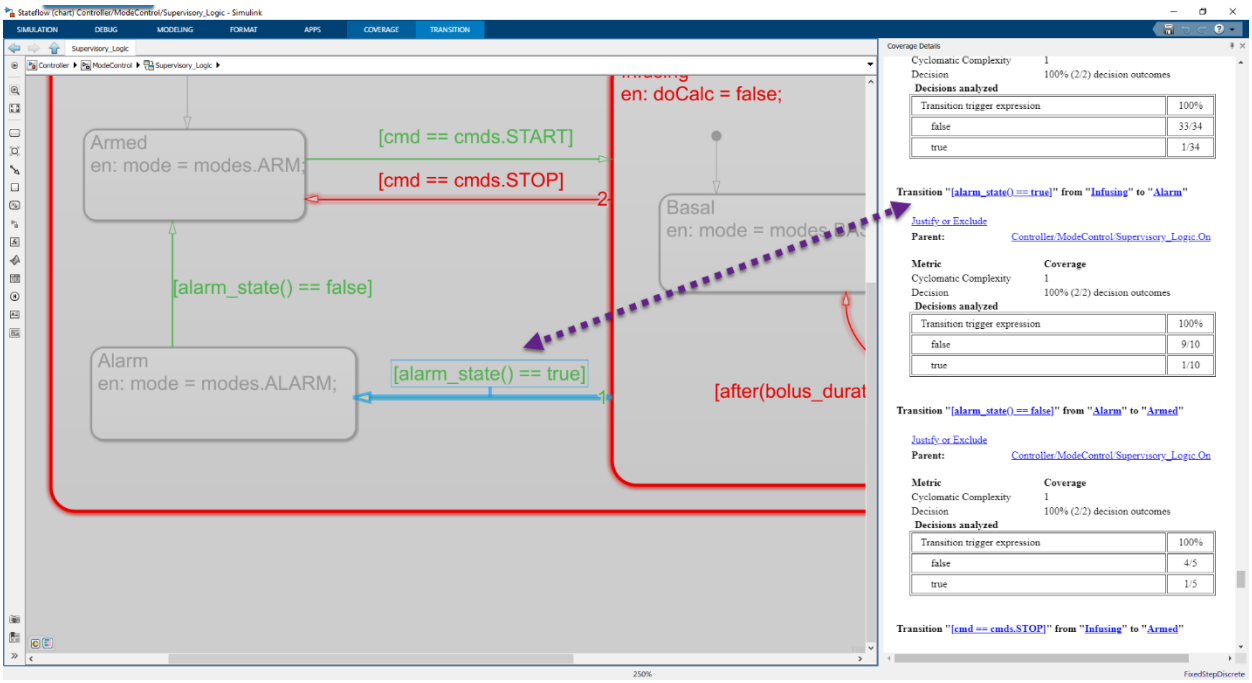
Modeling standards can assist analysis of the unit design. One option offered by Simulink Check™ is to compare models to industry standards, such as IEC 62304 and MISRA-C. Organizations can customize and augment this list of automated checks.

Software unit design verification can include functional verification through model simulation. This is usually accomplished by creating a test harness model referencing the software unit model or containing a library link to the software unit's Subsystem. The test harness model can be configured to run various unit test scenarios, expressed in a variety of possible ways. Simulink Test™ is a tool to manage and execute test scenarios and provide expected results. These test cases can be linked to relevant system and software requirements.



Simulink Test implementing an alarm handling test case by calling a test harness model to exercise a design model. In the harness. The Test Sequence block provides the test vector inputs while a Test Assessment block checks the outputs.

Simulink Coverage™ can be used to instrument the models to collect execution, data range, and various types of logical decision and branch coverage during simulation to evaluate the completeness of unit test scenarios.



Model highlighted with coverage results. The behavior (Stateflow chart) on the left shows that the alarm was detected, and the HTML report on the right provides a summary artifact. Embedded hyperlinks provide traceability to the model under test.

While in principle requirements-based functional test cases should provide a high degree of model coverage during their execution as part of the unit test plan, in practice this is not always the case. One example is that defensive aspects of the design, such as input range checking, might not be exercised via functional requirements-based testing. Simulink Design Verifier™ can help derive additional test cases based on model structure. One of its features is the use of formal model analysis methods to produce test cases to meet coverage objectives for a model (for example, 100% modified condition and decision coverage, or MC/DC) or to show that such a coverage object is unachievable due to limitations in the design. One can use these additional test cases to understand and document missing derived software requirements and/or to modify the design to improve its testability.

Software Unit Implementation and Verification

Simulink Coder and Embedded Coder can be used to transform a Simulink model into a software implementation in the C or C++ programming language. The details of software implementation are inferred from the detailed design expressed in an elaborated model, and the software architecture is faithful to the design in terms of code interfaces and packaging into functional and file-based code modules. The code generation tool can provide bidirectional traceability from model elements to code and can also include the description of

requirements linked to model elements as comments. This provides a basis for managing and tracking the implementation of such functional features and risk control measures.

The transformation to software may reveal additional derived requirements, such as unit run-time performance constraints and legacy code interfaces, or compliance to organizational coding standards. These items can be addressed by modifying the detailed design model and iteratively generating code. This process completes when unit verification succeeds.

An important area of unit verification is functional verification. Model test scenarios and harnesses that were created to perform functional detailed design verification on the software unit's Simulink model can be used as a baseline to perform implementation functional unit verification. The generated code can be compiled on the host—that is, the computer that the developer is using to design and create software—and invoked as an S-Function in the test harness model for direct model to code comparison using the same test vectors (so-called software-in-the-loop, or SIL, testing). In addition, it is possible to perform such tests on target-compiled code via target support packages for Embedded Coder. This processor-in-the-loop (PIL) technique can be used to help verify that the executable software has the same behavior as the Simulink model even when it is being executed as if part of the final device. Code coverage should be captured as part of the testing process to demonstrate that no unintended functionality was introduced. Simulink Coverage supports instrumenting the generated code in SIL simulations.

For safety class C software units, software unit acceptance includes additional criteria. Some, such as event sequence, data and control flow, and initialization of variables, are quite naturally elaborated as part of the Simulink detailed design model. Examples include automatic block sorting based on data availability vs. function-call triggered execution, explicit signal flow vs. Data Store Memory access, and options for explicit or default initialization of signals and states in the model. By using model coverage as a measure of adequacy of testing, functional test scenarios will be created to exercise these features of the software. As alluded to earlier, run-time performance or coding style aspects of the implementation can be more subtly represented in the model, and iterative model update and code generation may be required to meet the criteria. Simulink Check provides model advisor checks to enforce compliance to modeling standards like MISRA C:2012, and Polyspace® products can be used to analyze generated or handwritten C/C++ code for robustness and critical run-time errors. Two primary features are static analysis of code against industry standards (MISRA and JSF++, for example) and the detection of potential run-time errors via formal software analysis methods.

The results of functional unit verification activities can be captured with Simulink Report Generator™ and added to the overall unit verification documentation package; that is, the technical file for regulatory compliance.

Software Integration and Testing

The software integration process in Model-Based Design may have two distinct phases. The first (from a unit-to-item or bottom-up perspective) is when the integration tasks are performed within Simulink. The model hierarchy can be used to compose elaborated software items from

verified units or other items, and code can be generated at any level in the hierarchy one chooses. The Simulink semantics specify interfaces, rate and order of execution, and call tree. When code is generated from a larger item or the entire architecture model, these semantics are translated. All the verification, test, and documentation capability presented above for software unit testing can be applied at higher levels in the model hierarchy. In addition, note that test harness models and Simulink Test procedures can be created and managed for regression testing during incremental integration. Automating such iterative testing to increase its repeatability is recommended.

Often, the integrated code generated from a model for one or more software items would be integrated further with other software to complete the software system. While the benefits of the Model-Based Design environment will not be available to this process, there is no special consideration required for such an integration stage due to the origin of the code. Any IEC 62304–compliant integration plan can be followed. Tools and techniques such as continuous integration may form the mechanism to build the externally developed software with the software generated using Model-Based Design, for example.

The Simulink model history feature can become part of a software problem resolution process for items developed with Model-Based Design. At each save of a model, the user can provide a rationale for the changes. These entries could reference items tracked in a software problem resolution system.

Software System Testing

By this stage in the development process, Model-Based Design tools have provided significant insight into the completeness and correctness of the requirements, test cases necessary to exercise the design and code against those requirements, and documentation to help demonstrate the unit level and integration level verification of the software items that compose the software system. System testing will likely be performed outside of the Model-Based Design environment, however, in a manner compliant with the Standard, section 5.7, and following typical organizational practices.

A Simulink model of the system environment can be used during system testing to provide stimuli to the software. This approach is commonly taken with the assistance of dedicated deterministic hardware and software systems and is often referred to as hardware-in-the-loop, or HIL, testing. Depending on the complexity of the mathematics involved, HIL systems can often achieve real-time emulation of the system environment for closed-loop control performance evaluation. Simulink Real-Time™ can be used to implement HIL testing.

The Software Maintenance Process

Model-Based Design assists in the evaluation of the impact of a proposed software change through simulation of the affected software items. Furthermore, the model is an important design artifact that can help assess the coupling of a proposed change to various software items. Once the decision is made to implement a change, the appropriate activities selected by the organization as part of the software maintenance plan will be carried out, following the recommendations given above for the software development process.

Software Risk Management

Model-Based Design supports software risk management primarily by the capability to link requirements to model elements, which was covered earlier. This link and the reporting capability of Requirements Toolbox allow engineers to trace from a documented hazard to control measures in the model (design) to the implementation of these measures in a generated software item and finally to the simulation tests to assess their effectiveness. The traceability extends to the generated lines of software via comments and is navigable due to the code generation HTML report and the Navigate to Code feature in Simulink and Embedded Coder. Embedded Coder can also produce a detailed block to code traceability report as additional evidence.

MathWorks publishes lists of known major issues in its tools. These external bug reports can be reviewed during a project and items of interest can be managed as part of an anomaly list.

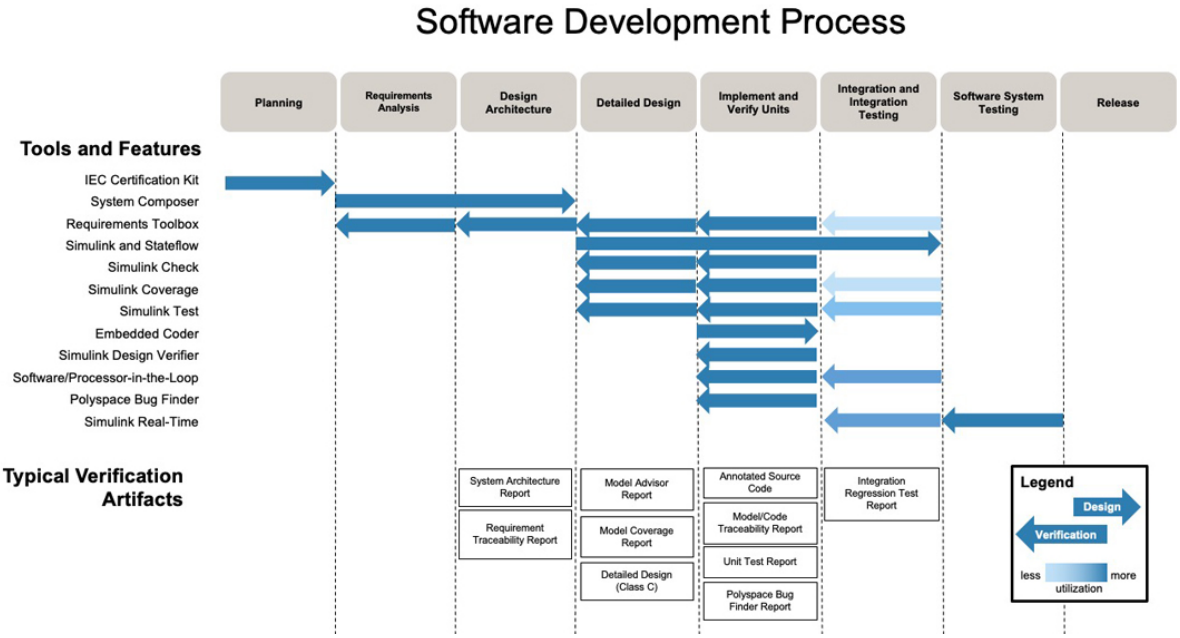
Software Configuration Management

The models and data created as part of the software engineering process must be identified and managed along with the resulting software items. They become a part of the design history of the software and are necessary in order to recreate or maintain it. Some typical artifacts that may be created during development activities include Simulink models, MATLAB scripts and functions, data dictionaries, generated production code, S-Functions and other user block libraries, simulation input data (test vectors) and results, and generated documentation such as design documents and test results. Information from a configuration management system, such as version number, save date, and author, can be incorporated into a Simulink model as informative text displayed in a Model Information block.

Summary

You can use Model-Based Design tools from MathWorks to comply with the software life cycle process requirements of IEC 62304. Simulink and associated tools are of particularly high value in the software development and maintenance processes, due to capabilities far in excess of paper-based methodologies to complete the key activities of requirements analysis, software design, unit implementation, and testing. Links are supported between process artifacts (such as requirements documents) and models, allowing a medical device manufacturer’s risk analysis and problem resolution processes to remain closely synchronized with system and software models. Model-Based Design tools provide the ability to contribute

to documentation of the software and its development process as well. For a summary view, see the diagram below, which shows some common usage scenarios of the tools and typical documentation artifacts that they can produce.



Applicability of Model-Based Design tools for IEC 62304–compliant software development.

Learn more about using MATLAB and Simulink for medical devices:

mathworks.com/solutions/medical-devices.html

References

1. “Caterpillar Automatic Code Generation,” Jeffrey M. Thate (Caterpillar), Larry E. Kendrick (Caterpillar), and Siva Nadarajah (MathWorks), Proceedings of the Society of Automotive Engineers World Congress 2004 (2004-01-0894)
2. “Rapid Deployment of Aerospace Flight Controls,” Edward L. Burnett (Lockheed-Martin Aeronautics), 2006, http://www.mathworks.com/programs/techkits/pcg_tech_kits.html
3. “GM Powertrain Automatic Code Generation Process,” 2005, http://www.mathworks.com/programs/techkits/pcg_tech_kits.html
4. “Medrad Ensures Safety of MRI Vascular Injection Pump Using MathWorks Tools,” 2004, http://www.mathworks.com/company/user_stories/userstory6313.html
5. “Alstom Generates Production Code for Safety-Critical Power Converter Control Systems,” 2005, http://www.mathworks.com/company/user_stories/userstory10591.html
6. “Achieving Six Sigma Software Quality Through the Use of Automatic Code Generation,” Bill Potter (Honeywell International), 2005, http://www.mathworks.com/programs/techkits/pcg_tech_kits.html
7. IEC 62304, “Medical Device Software – Software Life Cycle Processes,” International Electrotechnical Commission, Edition 1.0b, 2006
8. “Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow,” MathWorks Automotive Advisory Board - Version 2.1, 2007
9. “Weinmann Develops Life-Saving Transport Ventilator Using Model-Based Design,” https://www.mathworks.com/company/user_stories/weinmann-develops-life-saving-transport-ventilator-using-model-based-design.html, 2013
10. Solis-Lemus J A, Costar E, Doorly D, Kerrigan E C, Kennedy C H, Tait F, Niederer S, Vincent P, and Williams S, “A Simulated Single Ventilator / Dual Patient Ventilation Strategy for Acute Respiratory Distress Syndrome During the COVID-19 Pandemic,” Royal Society Open Science 7:200585, August 2020
11. *Software Requirements*, Karl Weigers, 2nd Edition, 2003, Microsoft Press
12. “Understanding and Controlling Software Costs,” Barry W. Boehm and Philip N. Papaccio, 1988, IEEE Transactions on Software Engineering 14(10), pages 1462-1476
13. MISRA C:2012. The Motor Industry Software Reliability Association: Guidelines for the use of the C language in critical systems, 2012
14. MISRA C Compliance for C code generated by Real-Time Workshop Embedded Coder can be found at http://www.mathworks.com/support/solutions/en/data/1-1IFP0W/?solution=1-1IFP0W
15. “Certify embedded systems developed using Simulink and Polyspace products to IEC 61508 and ISO 26262,” http://www.mathworks.com/products/iec-61508/
16. “Sound Verification Techniques for Developing High-Integrity Medical Device Software,” Jay Abraham (MathWorks), Paul Jones (FDA/CDRH), and Raoul Jetley (FDA/CDRH), Proceedings of the Embedded Systems Conference, San Jose, CA, 2009
17. The MathWorks external bug report database can be accessed at http://www.mathworks.com/support/bugreports
18. “Configuration Management of the Model-Based Design Process,” Gavin Walker (MathWorks), Jonathan Friedman (MathWorks), and Rob Aberg (MathWorks), Proceedings of the Society of Automotive Engineers World Congress 2007 (2007-01-1775)