Product development
White paper
June 2009

Rational® software

A new approach to verifying and validating medical device development.

Automating product development and compliance processes using IBM Rational software

Irv Badr, senior manager, Rational software,
IBM Software Group

Contents

Introduction

The list of features in medical devices is rapidly rising: Even the simplest of medical devices—such as diagnostic and monitoring systems—house more and more system components, which add greater functionality to the device at a low cost. But by adding components, device software becomes complex and burdens compliance testing and premarket certification activities, such as 510(k) submissions mandated by the U.S. Food and Drug Administration (FDA).

However, you can address those issues by using a model-based verification and validation process in your medical device development. First, this process can help you better manage the complexity of the software by abstracting it as a model. Second, the process can help you verify and validate an evolving system by leveraging an executable model earlier in the development process. Last, the early validation and verification of a system can help you reduce the total development time and shorten the FDA submission process, a required step for bringing a device to the U.S. marketplace.

Addressing challenges unique to the medical device industry

The typical workflow of developing embedded software products is often requirements gathering, analysis, system design, detailed design, testing and project management. But in the medical device industry, there is an additional step: compliance. The FDA regulates products developed for the U.S. marketplace through Quality System Regulations (QSR) 21 Code of Federal Regulations (CFR) Part 820 §30, which essentially requires:[1]

- *Proper documentation to be maintained in a Design History File (DHF).*
- *Compliance with Title 21 CFR Part 11, which governs the use of electronic signatures in DHFs.*
- *International marketplaces to comply with International Organization for Standardization (ISO) 13485:2003 and meet the European Union Medical Device Directive (EU MDD) 93/42 regulations.*
- *Compliance with Current Good Manufacturing Practices (CGMP) and Good Documentation Practices (GDP), both dictated by the FDA.*

Highlights

Traditionally, development teams have used source code to establish traceability between requirements and their implementation.

Traditional approaches to device compliance

To meet compliance goals, medical device development teams create a DHF at the beginning of a project. The file may contain informal, handwritten requirements and design notes as well as printouts and source-code excerpts of formal architecture and design documents. Some source-code-centric development teams also include requirements documents, ad hoc formal design documents and source-code listings that reflect how the teams are implementing requirements as source code. This workflow helps teams establish traceability between requirements and their exact implementation, as required by QSR. The traceability can prove that the device is being used for its intended purpose; however, the method of establishing traceability is not mandated. Because source code is readily available, many teams use it for traceability and as the medium for representing system architecture and design. This method is common among teams that lack a formal modeling approach in the development process.[2]

System design verification

Figure 1 shows a recommended development process from the FDA Design Control Guide.[3] Note that QSR compliance can be achieved in conjunction with GDP compliance by adhering to the iterative, or waterfall, development steps shown. Testers can perform system verification against the requirements by measuring design output against design input.

Design input comprises specifications defined by user requirements, which identify the intended use of the device. It usually includes formal or semiformal text documents and a few models reflecting a set of specifications against which the system is to be built. Design output, in turn, comprises procedures defined by the device maker that help ensure that the completed prototype aligns with the design input. It may include, in the case of embedded software, a list of the source code that belongs to the application.

Highlights

Because development teams need to ensure that they implement specifications and meet design goals, they must verify the device by measuring design input against design output.

Ultimately, development teams need to ensure that they implement specifications and meet design goals, so there is a need for traceability between the input and output milestones.

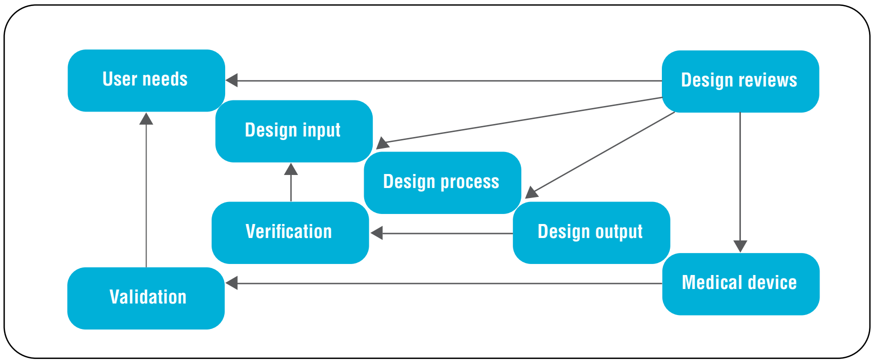

Figure 1: A design verification step verifies the design output against the design input.

Design input and output milestones are integral parts of the medical device development process. Because the milestones are applied to a large number of devices, no governing body specifies or mandates which design verification methods to use. As a result, device makers use an array of tools and processes for this purpose and rely on documents and source-code listings as described for design input and output.

System design validation

Because many teams consider source code to be the ultimate measure of a system's implementation status, they often traverse the source code to predict the final behavior of the actual device. In this source-code-centric approach, teams execute the application on the actual device and step through its lines of source code to observe the resulting system behavior and validate the system.

Highlights

Because verifying devices by measuring input against output can be time consuming and error prone, a better approach to system verification is to model and collect system data to prove the intended use of the device.

While effective, this method of validating the system has proved to be costly and error prone. Unit testing the evolving system on the target device can be cumbersome and slow. Further, it may not be possible to run the device through all intended use scenarios without incurring heavy expenses or logistical obstacles. For example, incomplete or nonexistent hardware may hamper final system testing on the device. Moreover, an incomplete platform may render faulty results for the tested application.

As software and system bugs are found and corrected during unit testing, teams may need to update associated design and requirements to reflect the changed source code. Depending on the size of the application, the updates may be time consuming and susceptible to errors. And some changes could be missed, causing an incorrect mismatch between the DHF and the implemented system.

Model-driven verification and validation

Neither internal design controls nor QSR require that you actually operate a medical device to verify and validate its system. As far as the FDA is concerned, collected system data can prove the intended use of the device. This data can be collected from the actual device as well as from a simulated execution of the device—in fact, automated application development tools offer virtually unparalleled efficiency gains.

Highlights

In a model-driven approach to design verification and validation, you can establish traceability between design components and requirements and test the design in difficult-to-create scenarios.

Figure 2 shows a validation- and verification-based process, which hinges on iterative development of design output based on design input. The figure also illustrates auditable traceability between requirements and system validation, and between architecture and design activities and system verification. Modeling tools can provide requirements traceability and executable model features automatically.[4] They can also report the latest design throughout the product lifecycle.

Additionally, you can leverage an automated requirements management tool to establish traceability between design components and requirements.

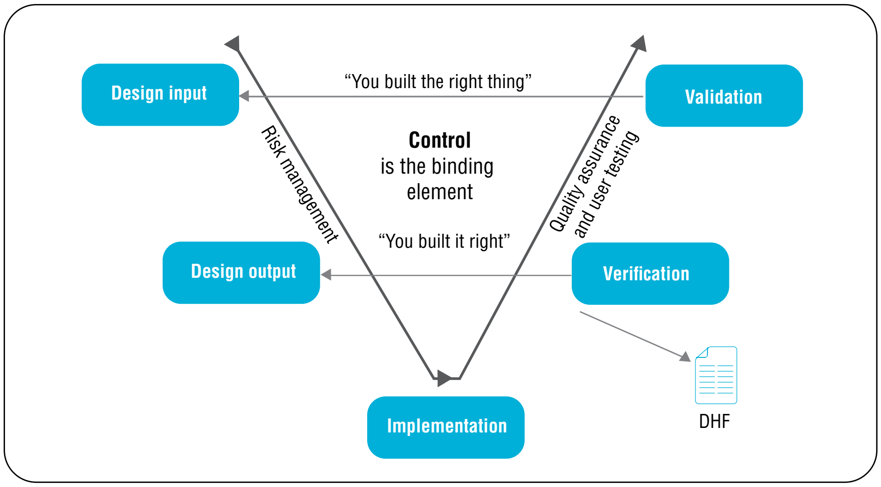

Figure 2: In a traditional approach, the validation- and verification-based process hinges on iterative development of design output based on design input.

In a model-driven process, you can architect and design an application in the modeling tool.[5] You can also automatically generate unit test cases from design models, as shown in figure 3. Here, models for both the design and the test case are executable.

Highlights

First, you want to verify the underlying design, test the design and depict actual usage scenarios for the device. Most defects and design oversights are caught during this model verification phase. Unit testing can put the design through possible scenarios, many of which may be difficult to create on the actual device.

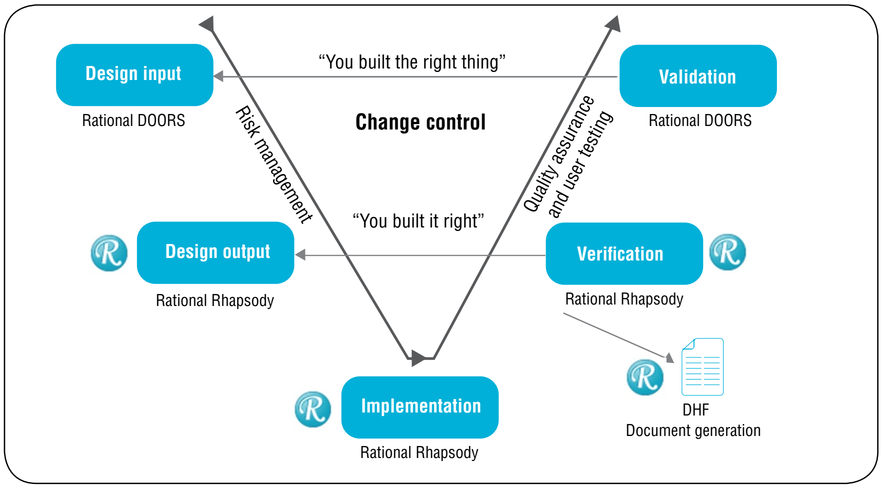

Figure 3: Automated requirements management tools can automatically provide requirements traceability and executable model features and establish traceability between design components and requirements.

Traditional system debugging is much slower than in a model-driven approach because the former requires many more lines of code—compared to model elements—to implement the same system.

Then, if you detect defects or oversights, you can immediately correct them in the design, thus eliminating the need to manually change the code and the design documentation. In traditional development, defects are fixed inside the source code after executing it on the real device. This happens later in the development process. As a result, traditional system debugging is much slower than in a model-driven approach because the former requires many more lines of code—compared to model elements—to implement the same system.

Finally, for system validation, you can automatically trace test cases to the system's operational requirements. Requirements traceability features between a formal requirements management tool and the modeling environment automate the traceability. In other words, a fully automated validation and verification process can be established.

Highlights

Model-driven development is particularly important for medical devices because it enables testers to help ensure patient safety during unforeseen or complex events that are difficult to fabricate.

Automated verification and validation approach in action

Figure 4 is a design for a blood oxygen monitor, which uses a finger clip sensor to measure blood oxygen levels as well as pulse rate. The design shows a state machine responsible for depacketizing the sensor data. By inputting either real or simulated sensor data, the state machine can quickly verify correct operation.

Figure 4 also verifies that both oxygen levels and pulse rate are within safe ranges. This test case is run alongside design verifications to help ensure patient safety during unforeseen or complex events that are difficult to create. For example, unit tests can model the monitor's behavior in the unlikely event that it doesn't detect a pulse while monitoring a patient's blood oxygen level. To test the monitor in the same real-world scenario, you would have to actually stop a patient's heart.

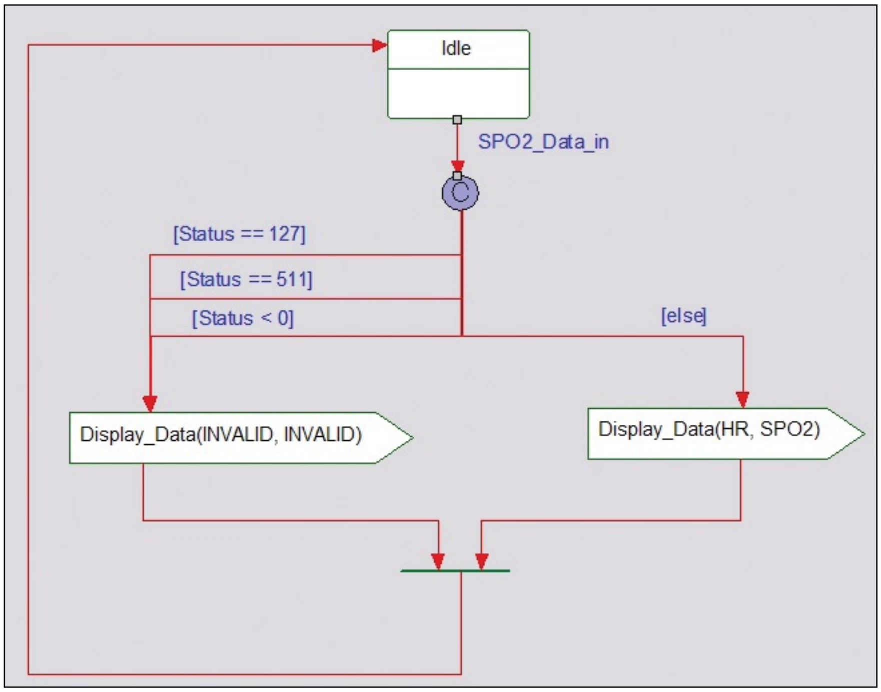

Figure 4: Using model-driven device development, you could model a blood oxygen monitor's behavior in the event that it doesn't detect a pulse while monitoring a blood oxygen level. To test the monitor in the same real-world scenario, you would have to actually stop a patient's heart.

Highlights

Rational lifecycle development software automates processes and meets validation and verification requirements from governing bodies.

Development and management tools for automated processes

IBM Rational® software provides lifecycle development tools that address the validation and verification requirements of QSR. The IBM Rational DOORS® requirements management platform can create the design input and manage system requirements and validation tests as structured and traceable sets of objects. Additionally, Rational DOORS supports 21 CFR Part 11–compliant electronic signatures, thereby enabling you to legally and more securely use electronic records in your design and development processes.

IBM Rational Rhapsody® software can model the medical device throughout the design process—from defining system development specifications to deploying software on the device. You can use Rational Rhapsody to model complex medical devices for an arbitrary combination of embedded and desktop systems. The embedded system may be a simple blood oxygen monitor or a large device such as a computed tomography (CT) scanner. Other systems may have an array of interrelated platforms, including headless desktop computers and monitoring workstations running conventional operating systems such as Linux®, UNIX® or Microsoft® Windows® platforms.

Further, Rational Rhapsody supports language-independent and operating system–independent modeling and can deploy the same models on nearly any platform. This is described as platform-independent modeling (PIM), where a single set of models can be used on many different or undefined platforms. PIM increases productivity by allowing you to leverage designed components in future generations of unknown platforms. Also, PIM enables you to develop your design before hardware is available, enabling you to validate functional behavior early in the design when it is less costly to fix defects. You can then validate the target-specific characteristics of the design when the hardware becomes available.

Highlights

For conventional, stand-alone medical devices that typically use a single-board embedded computer and a realtime operating system, Rational Rhapsody can provide system architecture and design support as well as automatic source-code generation. These features help enable rapid retargeting of the model from host execution to the target device. The tool is capable of supporting out-of-the-box, realtime operating systems or devices with no operating system. It also provides target-hosted, design-level debugging, which can prove to be valuable during the validation and verification process when target-level verification is necessary.

Rational Rhapsody can automate design documentation by extracting information from the model through customizable templates. It can then accurately document the design implementation and provide an integrated paper trail that originates from requirements management and modeling activities and follows through to validation and verification.

An improved, lower-cost development process

By using Rational automated verification and validation tools, you can help decrease development costs while creating a more reliable medical device with less risk of failure in the field.

When designing medical devices, you can address QSR design guidelines and regulations as well as system and software development best practices. By using an automated verification and validation approach, you can help decrease development costs while creating a more reliable medical device with less risk of failure in the field. Additionally, the automated approach provides live content for the DHF, which can be automatically produced and managed with the right development tools.

The suite of Rational lifecycle management solutions is designed to automate the development process through requirements management; system and software modeling; and automated, model-based testing tools such as Rational Rhapsody and Rational DOORS.

For more information

To learn more about IBM Rational Rhapsody and Rational DOORS software, contact your IBM representative or IBM Business Partner, or visit:

ibm.com/software/rhapsody

About the author

Irv Badr has nearly 20 years of development experience in embedded software and modeling technology. He has designed a communication infrastructure for medical devices, networking nodes, digital cable transmission and industrial controls. Irv also served as technical lead for sales and marketing of realtime operating systems and modeling tools.

After earning his bachelor's degree in biomedical engineering from the University of Illinois, Irv went on to earn a technical master of business administration degree from Northwestern University. Currently, he is a senior manager for Rational software, a unit of the IBM Software Group.

Endnotes

[1, 4] Irv Badr, "Developing Platform Independent Embedded Applications," *Embedded Systems Magazine*, July 2005.

[2, 5] Irv Badr, *Rapid Development through Agile Modeling*, Telelogic, February 2005.

[3] U.S. Food and Drug Administration, "Design Control Guidance For Medical Device Manufacturers," March 11, 1997.

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
June 2009
All Rights Reserved

References in this publication to IBM products or services do not imply that IBM intends to make them available in all countries in which IBM operates.

The information contained in this documentation is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this documentation, it is provided "as is" without warranty of any kind, express or implied. In addition, this information is based on IBM's current product plans and strategy, which are subject to change by IBM without notice. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this documentation or any other documentation. Nothing contained in this documentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM (or its suppliers or licensors), or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

IBM customers are responsible for ensuring their own compliance with legal requirements. It is the customer's sole responsibility to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer's business and any actions the customer may need to take to comply with such laws.