LDRA
Software Technology

Technical Briefing

Ease the Heartache of Medical Device Software Certification

Achieving Cost-effective Compliance with IEC 62304 - Amendment 1:2015

www.ldra.com

Background

Given that the EU[1] and FDA[2] definitions of what constitutes a medical device encompass a large majority of medical products other than drugs, it is small wonder that medical device software now permeates a huge range of diagnostic and delivery systems. The reliability of the embedded software used in these devices and the risk associated with it has been an ever-increasing concern as that software becomes ever more prevalent.

On June 15, 2015, the International Electrotechnical Commission, IEC, published Amendment 1:2015 to the IEC 62304 standard "Medical device software – software life cycle processes"[3] as their latest response to that concern. The set of processes, activities, and tasks described in this standard established a common framework for medical device software life cycle processes.

In practice, for all but the most trivial applications compliance with IEC 62304 can only be demonstrated efficiently with a comprehensive suite of automated tools. This document outlines the key software development and verification process of the standard, showing how automation minimizes their cost and provides a sound foundation for effective maintenance after product launch.

Classification

One of the more significant changes concerns the new risk-based approach to the safety classification of medical device software. The previous concept was based exclusively on the severity of the resulting harm. Downgrading of the safety classification of medical device software from C to B or B to A used to be possible by adopting hardware-based risk mitigation measures external to the software. The new amendment now replaces this concept with safety classification as shown in a decision tree (Figure 1).

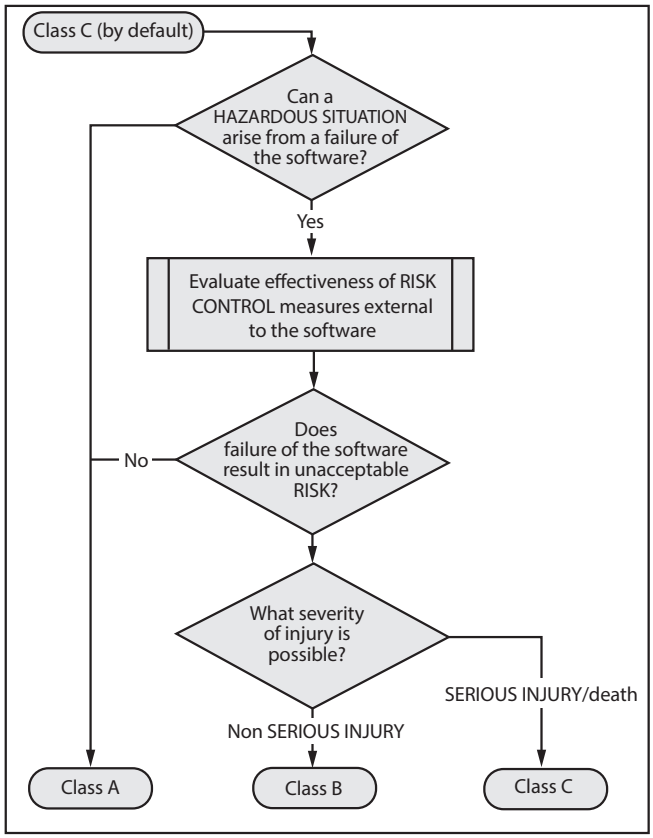

Figure 1: Safety classification according to IEC 62304:2006 +AMD1:2015

[1] "Directive 2007/47/ec of the European parliament and of the council". *Eur-lex Europa. 5 September 2007.*
[2] https://www.fda.gov/MedicalDevices/DeviceRegulationandGuidance/Overview/
[3] IEC 62304:2006/AMD1:2015 Amendment 1 - Medical device software - Software life cycle processes

The three classes defined in the standard range from class A – "The software system cannot contribute to a hazardous situation" - though to class C - "The software system can contribute to a hazardous situation … and the resulting possible harm is death or serious injury."

Partitioning of software items

The classification assigned to any medical device software has a tremendous impact on the code development process. It is therefore in the interests of medical device manufacturers to invest the effort to get it right the first time, minimizing unnecessary overhead by resisting over classification, but also avoiding expensive and time-consuming rework resulting from under classification.

IEC 62304:2006 +AMD1:2015 helps to minimize development overhead by permitting software items to be segregated. In doing so, it requires that *"The software ARCHITECTURE should promote segregation of software items that are required for safe operation and should describe the methods used to ensure effective segregation of those SOFTWARE ITEMS,"* and permits "*any mechanism that prevents on SOFTWARE ITEM negatively affecting another.*"

The standard uses an example where a software system has been designated Class C. That system can be segregated into one software item to deal with functionality of limited safety implications (software item X), and another to handle highly safety critical aspects of the system (software item Y).

That principle can be repeated in a hierarchical manner, such that software item Y can itself be segregated, and so on – always on the basis that no segregated software item can negatively affect another. Software items that are divided no further are defined as software units.

In practice, any company developing medical device software will carry out verification, integration and system testing on all software regardless of the safety classification, but the depth to which each of those activities is performed varies considerably.

For example, subclass 5.4.2 of the standard states that *"The MANUFACTURER shall document a design with enough detail to allow correct implementation of each SOFTWARE UNIT"* applies only to Class C code. In other words, that level of design documentation is not obligatory for Class A or Class B software.

Clause 5. Software Development PROCESS

The LDRA tool suite has been shown to ease the path to compliance both with IEC 62304 and with functional safety standards in other safety critical sectors by automating both the analysis of the code from a software quality perspective, and the required validation and verification work. Equally important, the tool suite provides traceability throughout the life-cycle, complete with artefacts to provide evidence to both internal and external auditors.

The V diagram in Figure 2 illustrates how the LDRA tool suite can help through the software development process described by IEC 62304. The tools also provide critical assistance through the software maintenance process (clause 6) and the risk management process (clause 7). Clause 5 of IEC 62304 details the software development process through eight stages ending in release.

Sub-clause 5.1 Software Development Planning outlines the first objective in the software development process, which is to plan the tasks needed for development of the software in order to reduce risks and communicate procedures and goals to members of the development team.

The foundations for an efficient development cycle can be established by using tools that can facilitate structured requirements definition, such that those requirements can be confirmed as met by means of automated document (or "artefact") generation.

The preparation of a mechanism to demonstrate that the requirements have been met will involve the development of detailed plans. A prominent example would be the software verification plan to include tasks to be performed during software verification and their assignment to specific resources.

Software Requirements Analysis (Sub-clause 5.2) involves deriving and documenting the software requirements based on the system requirements.

Achieving a format that lends itself to bi-directional traceability will help to achieve compliance with the standard. Bigger projects, perhaps with contributors in geographically diverse locations, are likely to benefit from an application lifecycle management tool such as IBM® Rational® DOORS®[4], or Siemens® Polarion® PLM®[5] . Smaller projects can cope admirably with carefully worded Microsoft® Word® or Microsoft® Excel® documents, written to facilitate links up and down the development process model.

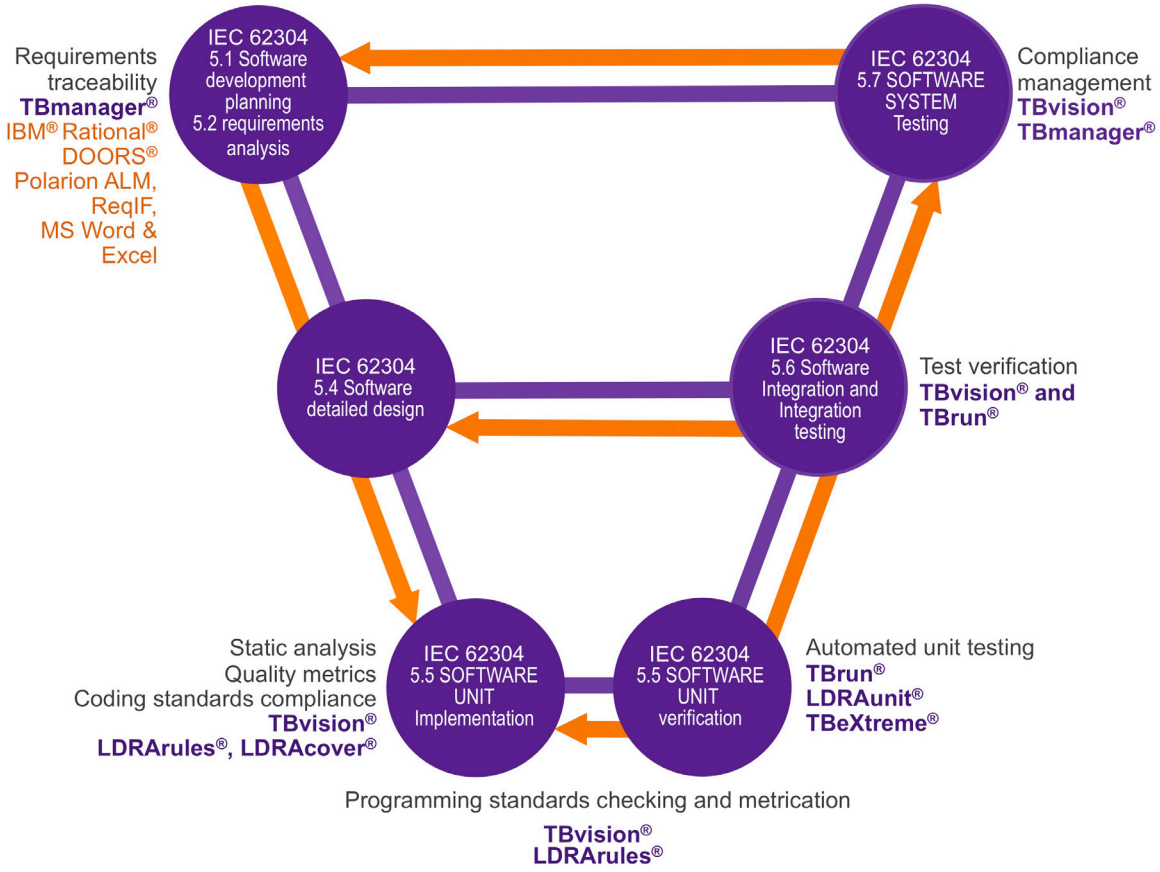

Figure 2: Mapping the capabilities of the LDRA tool suite to the guidelines of IEC 62304:2006 +AMD1:2015

This bi-directional traceability would be easily achieved in an ideal world. But most projects suffer from unexpected changes of requirement imposed by a customer. What is then impacted? Which requirements need re-writing? What elements of the code design? What code needs to be revised? And which parts of the software will require re-testing?

[4] http://www-03.ibm.com/software/products/en/ratidoor

[5] https://polarion.plm.automation.siemens.com/

A requirements traceability tool alleviates such concerns by automatically maintaining the connections between the requirements, development, and testing artefacts and activities. Any changes in the associated documents or software code are automatically highlighted such that any tests required to be revisited can be dealt with accordingly (Figure 3).

Software Architectural Design (Sub-clause 5.3) requires the manufacturer to define the major structural components of the software, their externally visible properties, and the relationships between them. Any software component behaviour that can affect other components should be described in the software architecture, such that all software requirements can be implemented by the specified software items. This is generally verified by technical evaluation.

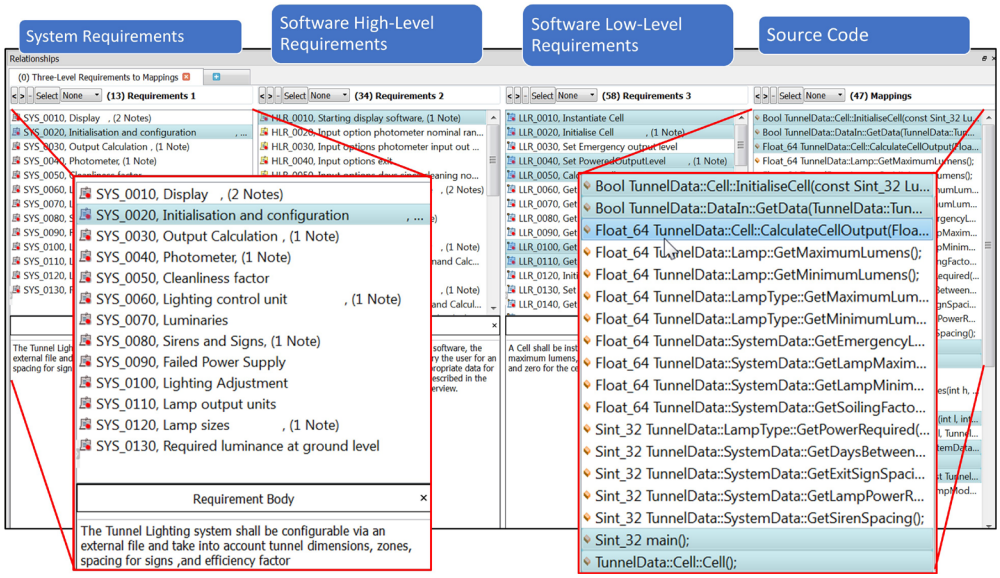

Figure 3: Automating requirements traceability with the TBmanager component of the LDRA tool suite

Developing the architecture means defining the interfaces between the software items that will implement the requirements. Any third-party software integration must be in accordance with **Sub-clause 4.4 , "Legacy Software".**

If a model-based approach is taken to software architectural design - for example, using MathWorks® Simulink®[6] , IBM® Rational® Rhapsody®[7] , or ANSYS® SCADE[8] then the LDRA tool suite's integration with the chosen modelling tools will make for seamless analysis of generated code and ensure traceability to the models.

Software Detailed Design (Sub-clause 5.4) involves the specification of algorithms, data representations, and interfaces between different software units and data structures to implement the verified requirements and architecture. Because implementation depends on detailed design, it is necessary to verify the detailed design before the activity is complete, generally by means of a technical evaluation of the detailed design as a whole, and of the verification of each software unit and its interfaces.

Later in the development cycle, the LDRA tool suite can help by generating graphical artefacts suited to the review of the implemented design by means of walkthroughs or inspections. One approach is to prototype the software architecture in an appropriate programming language, which can also help to find any anomalies in the design. Graphical artefacts like call graphs and flow graphs are well suited for use in the review of the implemented design by visual inspection.

[6] https://uk.mathworks.com/products/simulink.html
[7] http://www-03.ibm.com/software/products/en/ratirhapfami
[8] http://www.ansys.com/products/embedded-software/ansys-scade-suite

Software Unit Implementation and Verification (Sub-clause 5.5) involves the translation of the detailed design into source code. To consistently achieve the desirable code characteristics, coding standards should be used to specify a preferred coding style, aid understandability, apply language usage rules or restrictions, and manage complexity. The code for each unit should be verified using a static analysis tool to ensure that it complies in a timely and cost-effective manner.

Verification tools such as the TBvision component of the LDRA tool suite largely offer support for a range of coding standards such as MISRA C and C++, JSF++ AV, HIS, CERT C, and CWE. The better tools will be able to confirm adherence to a very high percentage of the rules dictated by each standard, and will also support the creation of, and adherence to, in-house standards from both user-defined and industry standard rule sets.

IEC 62304 also requires strategies, methods, and procedures for verifying each software unit. Amongst the acceptance criteria are considerations such as the verification of the proper event sequence, data and control flow, fault handling, memory management and initialization of variables, memory overflow detection and checking of all software boundary conditions.

The TBrun® unit test component of the LDRA tool suite provide a graphical user interface for unit test specification which is used to create tests according to the defined specification and to present a list of all defined test cases with appropriate pass/fail status, requiring a minimum of specialist knowledge. By extending the process to the automatic generation of test vectors, the tool provides a straightforward means to analyse boundary values without creating each test case manually. Test sequences and test cases are retained so that they can be repeated ("regression tested"; Figure 4), and the results compared with those generated when they were first created.

Thorough verification also requires static and dynamic data and control flow analysis. Static data flow analysis produces a cross reference table of variables, which documents their type, and where they are utilized within the source file(s) or system under test. It also provides details of data flow anomalies, procedure interface analysis and data flow standards violations.

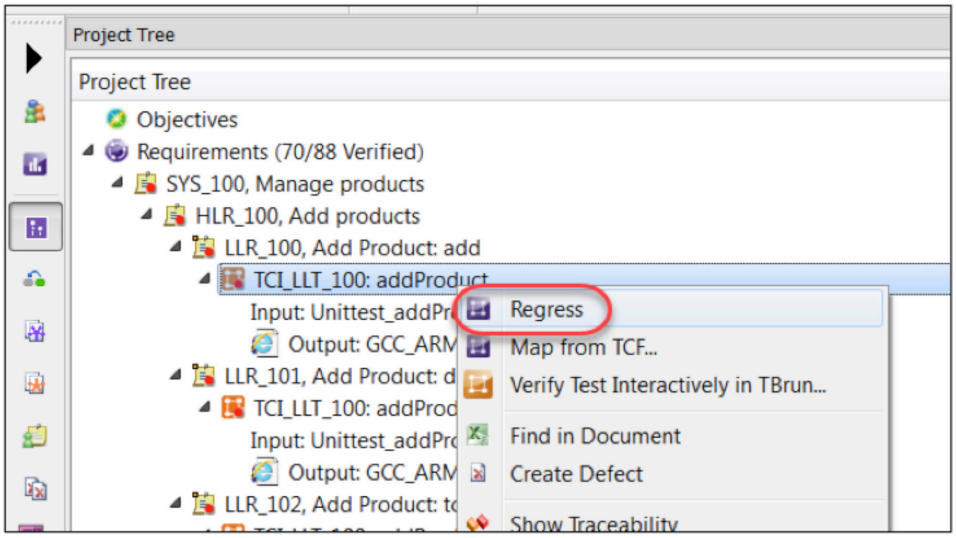

Figure 4: "Regression Testing" – Re-running unit tests to show that the functionality they describe still holds true, using TBmanager and TBrun components of the LDRA tool suite

Dynamic data flow analysis builds on that accumulated knowledge, mapping coverage information onto each variable entry in the table for current and combined datasets and populating flow graphs to illustrate the control flow of the unit under test.

Software Integration and Integration Testing (Sub-clause 5.6) focuses on the transfer of data and control across a software module's internal interfaces and external interfaces such as those associated with medical device hardware, operating systems, and third party software applications and libraries. This activity requires the manufacturer to plan and execute integration of software units into ever larger aggregated software items, ultimately verifying that the resulting integrated system behaves as intended. Integration testing can also be used to demonstrate program behaviour at the boundaries of its input and output domains and confirms program responses to invalid, unexpected, and special inputs.

To show which parts of the code base have been exercised during testing, the LDRA tool suite has the capability to perform dynamic structural coverage analysis, both at system test level and at unit test level. Mechanisms for structural coverage such as statement, branch, condition, procedure/function call, and data flow coverage vary in intensity, and so are specified by the standard depending on classification.

A common approach is to operate unit and system test in tandem, so that (for instance) coverage can be generated for most of the source code through a dynamic system test, and complemented using unit tests to exercise such as defensive code. It is advisable to re-run (or "regression test") these test cases as a matter of course and perhaps automatically, to ensure that any changed code has not affected proven functionality elsewhere.

Software System Testing (Sub-clause 5.7) requires the manufacturer to verify that the requirements for the software have been successfully implemented in the system as it will be deployed, and that the performance of the program is as specified.

Clause 6. Software Maintenance PROCESS

With the advent of the connected device and the Internet of Things, system maintenance takes on a new significance. For any connected systems, requirements don't just change in an orderly manner during development. They change without warning - whenever some smart Alec finds a new vulnerability, develops a new hack, compromises the system. And they keep on changing throughout the lifetime of the device.

For that reason, the ability of next-generation automated management and requirements traceability tools and techniques to create relationships between requirements, code, static and dynamic analysis results, and unit- and system-level tests is especially valuable for connected systems. Linking these elements already enables the entire software development cycle to become traceable, making it easy for teams to identify problems and implement solutions faster and more cost effectively. But they are perhaps even more important after product release, presenting a vital competitive advantage in the ability to respond quickly and effectively whenever security is compromised.

Many software modifications will require changes to the existing software functionality – perhaps with regards to additional utilities in the software. In such circumstances, it is important to ensure that any changes made or additions to the software do not adversely affect the existing code.

The LDRA TBmanager® component of the LDRA tool suite helps alleviate this concern by automatically maintaining the connections between the requirements, development, and testing artefacts and activities – not just during development, but onwards into deployment and the maintenance phase.

IEC 62304 with its many sections, clauses and sub-clauses may at first seem intimidating. However, once broken down into digestible pieces, its principles offer sound guidance in the establishment of a high quality software development process, and a sound foundation for subsequent product maitenance. Such a process is paramount for the assurance of true reliability and quality—and above all the safety and effectiveness of medical devices. When used with a complementary and comprehensive suite of tools for analysis and testing, it can smooth the way for development teams to work together to effectively develop and maintain large projects with confidence in their quality.

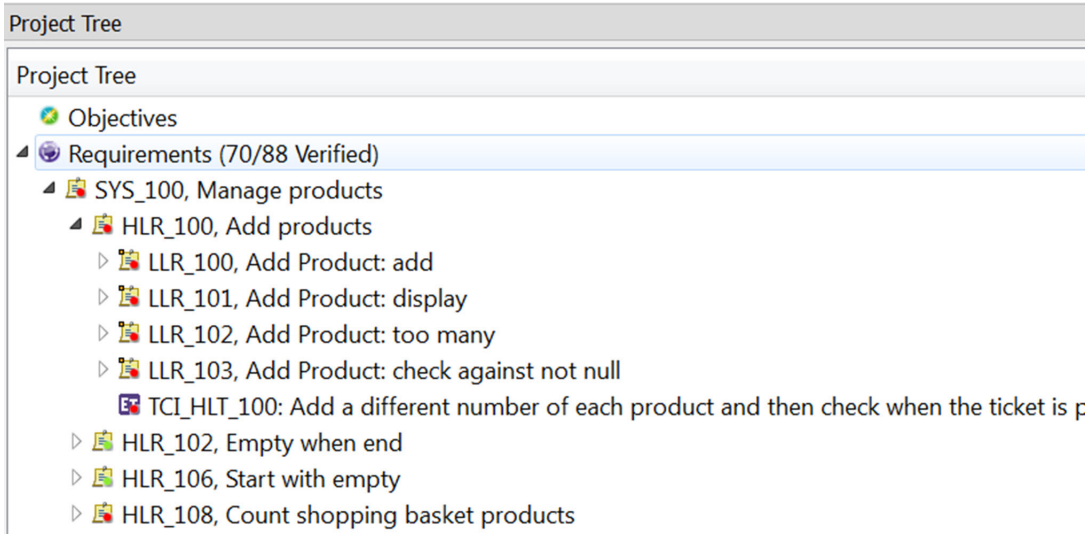

Figure 5: Showing functions requiring retest with the TBmanager component of the LDRA tool suite

Figure 5 shows a display from the TBmanager requirements traceability component of the LDRA tool suite. In this example, a system has been subject to a change request for the "Add products" requirement. Those parts of the system which are potentially affected by the change are easily identified by means of a red dot, whereas unaffected functions carry a green dot

As shown in the illustration, there is a test case file (tcf) associated with each of four low-level requirements – "add", "display", "too many" and "check against not null". Those files retain the test vectors associated each of these low level requirements, meaning that they can all be re-run at the touch of a button and the code's functionality confirmed.

Conclusion

A software functional safety standard such as that prescribed by IEC 62304 with its many sections,clauses and sub-clauses may at first seem intimidating. However, once broken down into digestible pieces, its guiding principles offer sound guidance in the establishment of a high quality software development process. Such a process is paramount for the assurance of true reliability and quality—and above all the safety and effectiveness of medical devices.

The increasing demand for the connectivity of medical devices places a new emphasis on the need to respond quickly and effectively to vulnerabilities discovered or compromised after product launch, demanding that the high standards of the development process must be upheld not only leading up to initial product release but into maintenance and beyond.

The "best practise" guidance given by the standard is therefore something to be embraced, not feared. When complemented by a comprehensive suite of tools for analysis and testing, it can smooth the way for development teams to work together to effectively develop and maintain even large projects with confidence in their quality.

www.ldra.com

LDRA UK & Worldwide
Portside, Monks Ferry,
Wirral, CH41 5LH
Tel: +44 (0)151 649 9300
e-mail: info@ldra.com

LDRA Technology Inc.
2540 King Arthur Blvd, Suite 228,
Lewisville, Texas 75056
United States
Tel: +1 (855) 855 5372
e-mail: info@ldra.com

LDRA Technology Pvt. Ltd.
Unit No B-3, 3rd Floor Tower B,
Golden Enclave, HAL Airport Road,
Bengaluru
560017
India
Tel: +91 80 4080 8707
e-mail: india@ldra.com