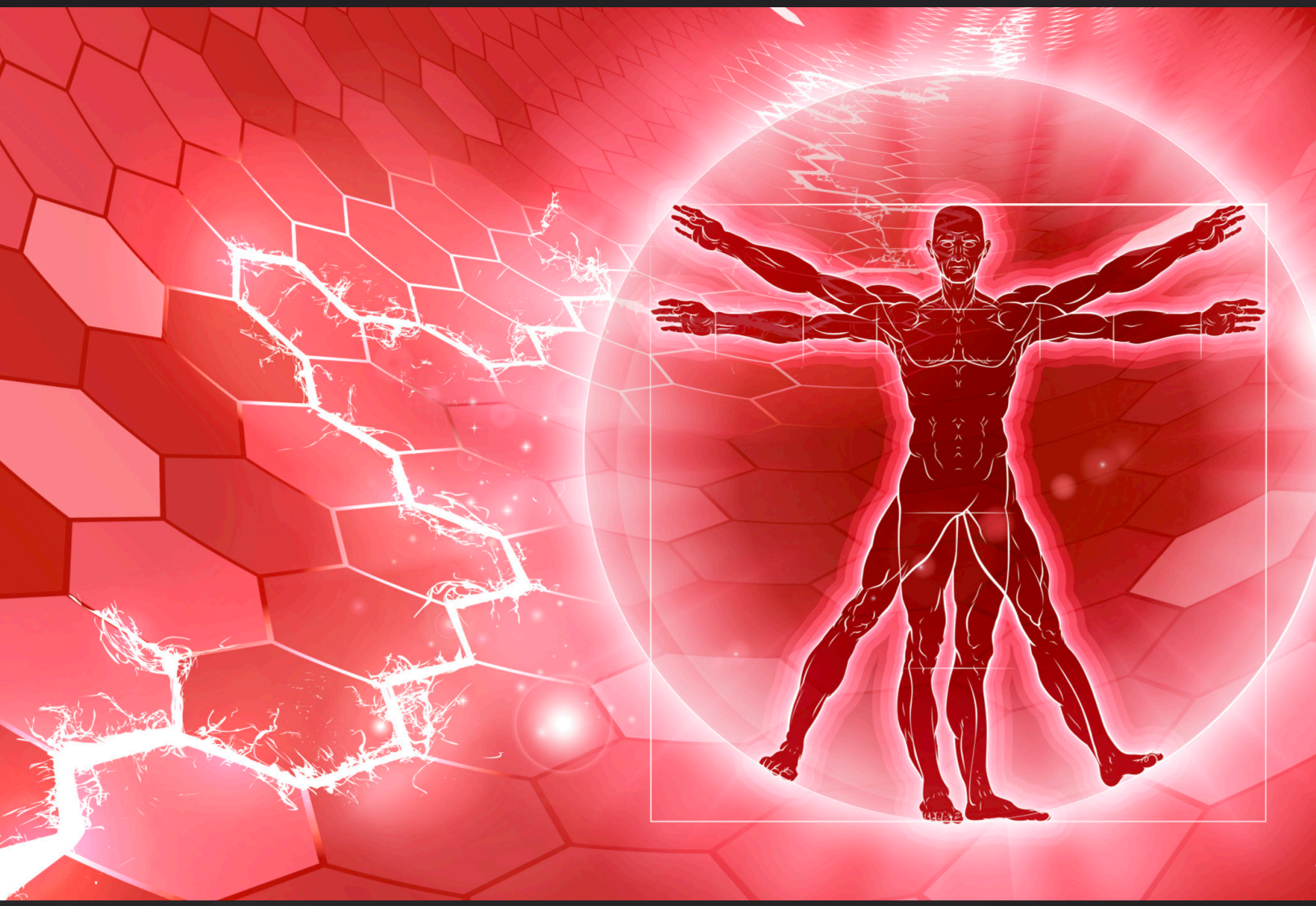
GRAMMATECH

USING STATIC ANALYSIS FOR OVERLAPPING SAFETY AND SECURITY REQUIREMENTS FOR MEDICAL DEVICES



TRUSTED LEADERS OF SOFTWARE ASSURANCE AND ADVANCED CYBER-SECURITY SOLUTIONS

WWW.GRAMMATECH.COM

INTRODUCTION

Software and embedded systems used in medical devices are subject to strict and varied regulations. These include both the development process and quality management throughout the product lifecycle. New medical device regulation will not make the overall situation clearer. However, there are some similarities between the different rules that make the application of multiple standards to the same project a little easier. Achieving compliance with the use of static code analysis improves productivity and reduces compliance effort.

The move to digitization and automation is happening in the medical industry as it is in others – almost every medical device requires software. Wireless connectivity is becoming increasingly important in order to exchange data or connect sensors increasing the probability of security risks.

The positive aspects of this shift to connectivity is better monitoring of patient vitals and statistics, new therapeutic approaches and more precise diagnoses. However, this comes with several downsides such as increased risks from software or hardware failures, potential vulnerabilities to attacks, and ultimately, endangerment to life and limb of patients. The benefits are worth the extra effort needed to mitigate these risks. Rather than giving up on these benefits, the goal must be to make the software as safe and reliable as possible. Ultimately, the regulations for medical devices are aimed precisely at managing and reducing the impact of these risks. The Medical Devices Regulation EU 2017/745 (MDR), with the end of the transitional period for the implementation as of May 26, 2020, the requirements will be significantly tightened again. These new regulations impact both software requirements and workflows for embedded developers.

There are basically two scopes for standards and regulations: One is the quality assurance in development and the other is quality management throughout the lifecycle of a product, from concept to end of life. The MDR makes it clear that the latter is the required scope (MDR 2017/745 section 17.1):

> *"For devices that incorporate software or for software that are devices in themselves, the software shall be developed and manufactured in accordance with the state of the art taking into account the principles of development life cycle, risk management, including information security, verification and validation."*

The fact that new developments in the medical field must meet higher quality standards is made obvious by the fact that the quality control of products already on the market has been poor. Improvements are needed to prevent patient injuries. It's also important for standards to take into account that software-based or even software-only products undergo significant changes over their lifetime. For example, by integrating new functions via an update or even if software library used in development is replaced by a newer version. Upgrades and updates alone are a significant source of concern which can render a hitherto problem-free system vulnerable -- even updates with bug fixes themselves may contain new bugs.

MINIMIZE RISKS TO PATIENTS

For software developers, four standards form the foundation of the requirements of MDR:

- EN 60601-1 – Medical Electrical Equipment – Part 1- General Requirements for basic safety and essential performance
- IEC 62304 – Medical device software – Software lifecycle processes
- IEC 82304 – Health software – General requirements for product safety
- ISO 14971 - Medical devices — Application of risk management to medical devices

The foundation for medical device safety is outlined in EN 60601-1, the basic standard for medical electrical equipment. It requires these devices to be developed and manufactured in such a way that they are “first-fail-safe” meaning that no first fault of the system shall render further use of the device unsafe.

On the contrary, the basic assumption of the generic functional safety standards like IEC 61508 for electrical, electronic and programmable electronic systems start with the assumption that it is not possible to produce mass-produced products that are entirely safe to use at an economically justifiable expense over the entire service life. Instead, the goal of these standards is to minimize the number of errors and minimize the risk to users. Medical devices must justify the overall risk to the patient in order to be approved for use unlike automobiles, for example, which despite modern technology, have an inherent risk when used.

RISK CLASSIFICATION

Crucial for planning the relative development effort to be made are the risk classes. Although these vary in detail in each different framework, the same basic concept is used: The relative impact on patient, user or medical staff of a failure of the software or device determines the classification. Devices that have the potential to cause death or injury, for example, have the most scrutiny and stricter rules for all phases of the Software Development Lifecycle (SDLC).

The C Programming Language a Weak Point for Security and Quality

The programming languages C and C ++ continue year after year to be the most popular languages used in embedded software development. When C was first defined in the 1970s, the focus was on speed and flexibility. The Internet did not exist, cyber security was not an issue. The prolific use of pointer arithmetic, in particular, is responsible for many security issues and serious run time bugs, the root cause of widespread buffer overruns. In addition, for the sake of flexibility, the definition of a “correct” C program is very loose. There are many ambiguities that the compiler has to interpret and resolve. In addition, undefined and unspecified behavior of a C program does not violate the standard, for example, C99 standard names 191 acceptable deviations and compatible compilers attempt to do the most reasonable thing. In such cases, where there maybe more than one choice for a solution it’s up to the compiler to resolve the situation. From the point of view of functional safety, this is not ideal, programming errors are created very easy when using acceptable C language to the compiler but results in errors that created easy but often difficult to recognize.

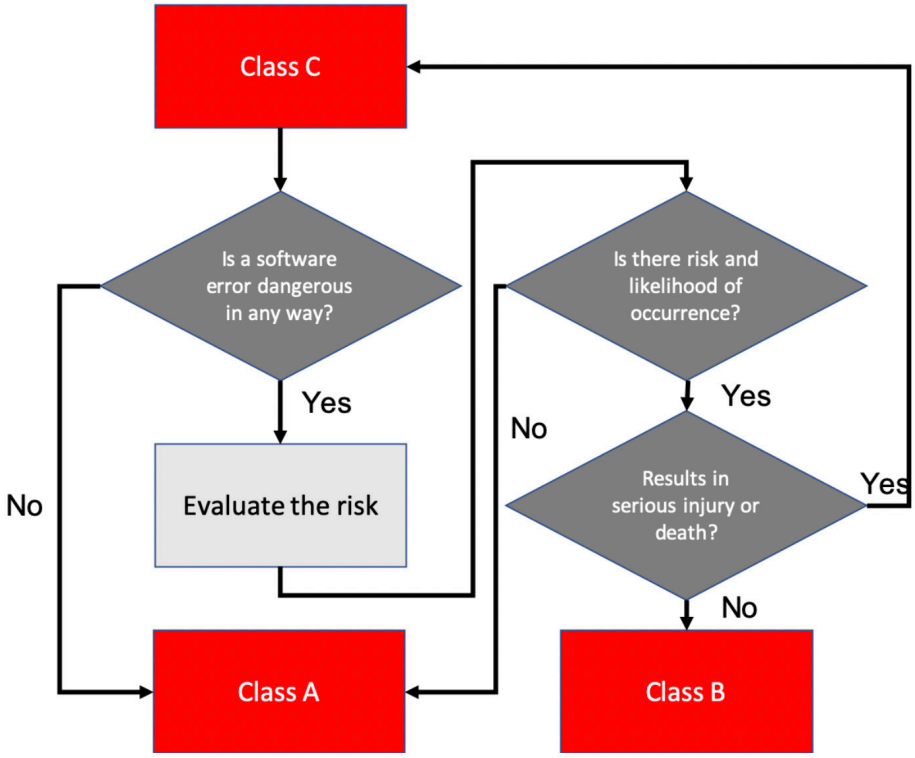


Figure 1: By default, IEC 62304 assigns Class C to all devices. The product must then prove its harmlessness for a lower rating.

The MDR distinguishes between risk classes I, IIa, IIb and III. In a class I device, there is little risk that patients will be significantly harmed by malfunction. A class III device, such as implantable pace maker, failures can lead to death or serious injury. Class IIb are active devices that are used to administer medicinal product and can have significant risk to patients due to failures but not on the order of a Class III device, for example. These classifications are similar to IEC 62304, which uses the classes A, B, and C.

Unlike IEC 62304, which classifies the highest risk as class C wherein products are assumed to be in the highest risk class until it is found that failure of the system cannot cause serious injury, in which case the classification is lowered to B. With the new MDR, the classification criteria are tightened with many products assigned to a higher class than might have been in the past.

The possible transition from one medical device class to another has a serious impact on the design, development, testing and deployment of medical devices and software. Manufacturers need to be aware of this change in philosophy outlined in the MDR.

VERIFICATION AND VALIDATION

All safety and security standards for software rely heavily on validation and verification to both prove proper function but also patient/user risk reduction. For example, IEC 62304 requires, in Section 5.5.2,

that the medical device manufacturer establish a Software Unit Verification Process. More precisely, the process must introduce a strategy, methods and procedures to check every piece of code for formal correctness and accuracy. In addition, the software must be validated and tested to see if it provides the required function. However, it becomes clear that the authors of IEC 62304 were more concerned with medical software as part of an embedded systems rather than separate application software. The standard makes few statements about the validation of the software alone. For embedded systems, this makes sense, because in this context, the software cannot be validated without the hardware. The complete medical device must always be regarded as a system. To address medical or health software, the missing guidelines for application software are now covered by IEC 82304.

IEC 82304 explicitly applies to stand-alone software and is not limited to medical products in the strict sense, but also applies to products that generally pursue an improvement in individual health. In many parts, IEC 82304 explicitly refers to IEC 62304, so both frameworks essentially overlap. For quality management guidelines to be used throughout the life cycle, for example, IEC 82304 refers to IEC 62304. The new standard is also dedicated to the validation of software alone, something that plays less significance in IEC 62304. IEC 82304 addresses the widespread criticism of the lack of concrete guidance on how software can be validated in IEC 62304. The good thing is that tools and processes in place to support medical device software can be leveraged for stand-alone medical software as well.

RISK MANAGEMENT IS MANDATORY

In addition to guidelines for validation and verification during the development process, medical device and software standards also address quality management throughout the lifecycle of the product, most notably ISO 14971, “Application of risk management to medical devices”.

From the earliest stages of designing and developing a new medical device or application, the focus is on risk assessment, whereby analysis of possible injury or patient harm and the probability of its occurrence is required. These risks must be minimized and repeatedly reviewed over the lifetime of the product. For example, IEC 62304 requires that the risk management process follow ISO 14971. This requirement means that risk management is an ongoing process because software and hardware of a medical device can change, so risk reviews are needed and mitigations need to re-evaluated. For medical device software, this is required with each product update.

The ongoing nature of risk and quality guidelines, verification and validation are not one-off processes before going to market, but integral parts of development and subsequent revisions.

DRAWBACKS OF TRADITIONAL FUNCTIONAL TESTING

Testing is crucial for validation, proving the software meets its purpose, mostly it's functional requirements. However, in the case of verification, classic testing has two drawbacks:

1. Executable code and real or simulated target environments must be available which is difficult to achieve at very early stages of the development process.

2. Testing can only find errors if three conditions are met simultaneously:

    - The test case must execute the portion of the code with the fault,
    - The error must lead to an error condition, and
    - The error condition must lead to a deviation of the result from the expected result.

In order to find errors in code more reliably, static code analysis is an established technique. In some safety-critical areas, such as aviation, static analysis is enshrined in the standards.

STATIC ANALYSIS

In static code analysis, the code is examined using a model, known as an Intermediate Representation (IR) which describes the control and data flows and algorithms are used to traverse this model during the analysis. All control and data flows are considered during analysis which means errors can be detected that might be missed in typical testing techniques. Examples of complex errors that advanced static analysis tools can discover include buffer overruns or null pointer dereferences which may lead to insecure access to data sources. Static analysis tools are also capable of ensuring conformance to industry coding standards and corporate programming guidelines.

Advanced commercial static analysis tools like GrammaTech CodeSonar give the developer insight and tools to help detect and fix safety, security and quality issues. In addition, the tool generates comprehensive reports, which simplify documentation requirements during audits. Another unique feature of CodeSonar is binary code analysis which means object files, libraries and even third-party binary products can be analyzed. This is useful in managing the software supply chain of third-party applications, middleware and operating systems -- increasingly being used in embedded development.

IEC 62304 AND APPLICABILITY OF STATIC ANALYSIS TO MEDICAL DEVICE SOFTWARE

Although IEC 62304 (or MDR) doesn't call out specific development tools, it does indicate the need for rigorous testing, acceptance criteria, and traceability. Performing these functions without tools isn't practical given the scope of most medical device software projects. For example, (excerpts from the standard):

- Section 5.5.2 of IEC 62304 requires a software unit verification process: The MANUFACTURER shall establish strategies, methods and procedures for verifying each SOFTWARE UNIT.

- Section 5.5.3 requires: The MANUFACTURER shall establish acceptance criteria for SOFTWARE UNITS prior to integration into larger SOFTWARE ITEMS as appropriate, and ensure that SOFTWARE UNITS meet acceptance criteria...does the software code conform to programming procedures or coding standards?

- Section 5.5.4 provides additional acceptance criteria:

  5.5.4 Additional SOFTWARE UNIT acceptance criteria

  When present in the design, the MANUFACTURER shall include additional acceptance criteria as appropriate for:

  a) proper event sequence;

  b) data and control flow;

  c) planned resource allocation;

  d) fault handling (error definition, isolation, and recovery);

  e) initialisation of variables;

  f) self-diagnostics;

  g) memory management and memory overflows; and

  h) boundary conditions.

Many of these acceptance criteria are well suited to static analysis. In fact, the case for static analysis is so strong, the FDA has used GrammaTech CodeSonar to analyze medical device software to evaluate the quality of the source code following a series of infusion pump failures.

ACCELERATING MEDICAL DEVICE DEVELOPMENT AND STANDARDS COMPLIANCE

Static analysis tools offer support for rigorous testing and documentation required for pre-market approval in the following ways:

- Static analysis finds bugs that coverage-based testing techniques miss: Unit testing is often measured by the level of coverage, such as statement and decision coverage. Although rigorous, there are defects that make it through this type of testing that static analysis tools can discover.

- Static analysis detects defects early: Preventing defects from occurring at the developer's desktop is the ideal situation -- it saves money in testing and remediation that increases as a project progresses.

- Static analysis can handle SOUP: Software of Unknown Pedigree/Provenance (SOUP) requires special handling in medical device software, and good static analysis tools are capable of evaluating the quality and security of third-party and commercial off the shelf software (including binary-only executables and libraries).

- Static analysis accelerates documentation, traceability and testing evidence: Documenting the results of software unit acceptance is critical to proving compliance to certification standards. Static analysis tools have rich reporting features to help support certification requirements.

CONCLUSION

The numerous standards for medical products cause significant costs for medical device manufacturers. With the new MDR guidelines they may increase in order to meet the stricter standards, as risk assessment requires more scrutiny in order to reduce the classification of a device. All of this makes it more important to have efficient and effective monitoring of the development process in terms of safety, quality and security for the complete product life cycle. Advanced static code analysis significantly simplifies the verification of software. Errors are detected early and can be fixed with less effort, before the integration into the software build, at the developer's development environment. Similarly, changes to the code or to libraries can be closely monitored to ensure the integrity of the entire system throughout the product cycle. Static analysis provides both faster time-to-market and higher code quality - and ultimately, both aspects benefit the patient's health.

GrammaTech, Inc. is a leading developer of software-assurance tools and advanced cyber-security solutions. GrammaTech helps organizations develop and release high quality software, free of harmful defects that cause system failures, enable data breaches, and increase corporate liabilities in today's connected world. GrammaTech's CodeSonar is used by embedded and enterprise developers worldwide.