Software Fault Management for Medical Devices

John Greenland, LDRA Technology

From hearing aids to Positron Emission Tomography (PET) imaging equipment, medical devices are increasingly impacting (and hopefully improving) our quality of life. Because we now rely on medical devices so heavily and because the devices' software is so critical to their operation, software fault management and the ability to reduce faults throughout the development lifecycle have become hot-button issues.

To implement fault management and fault reduction, medical device companies need to look at a variety of different methodologies focused on at least two levels: fault reduction throughout the development lifecycle and fault handling at application runtime. During the software development lifecycle, requirements traceability, static analysis, dynamic analysis, and testing strategies can all contribute significantly to software quality and fault reduction when applied at the proper stage in the development process. Once faults have been minimized using proven techniques during development, fault handling at run-time can protect safety-critical software from unforeseen events that can creep up during the operation of complex software applications.

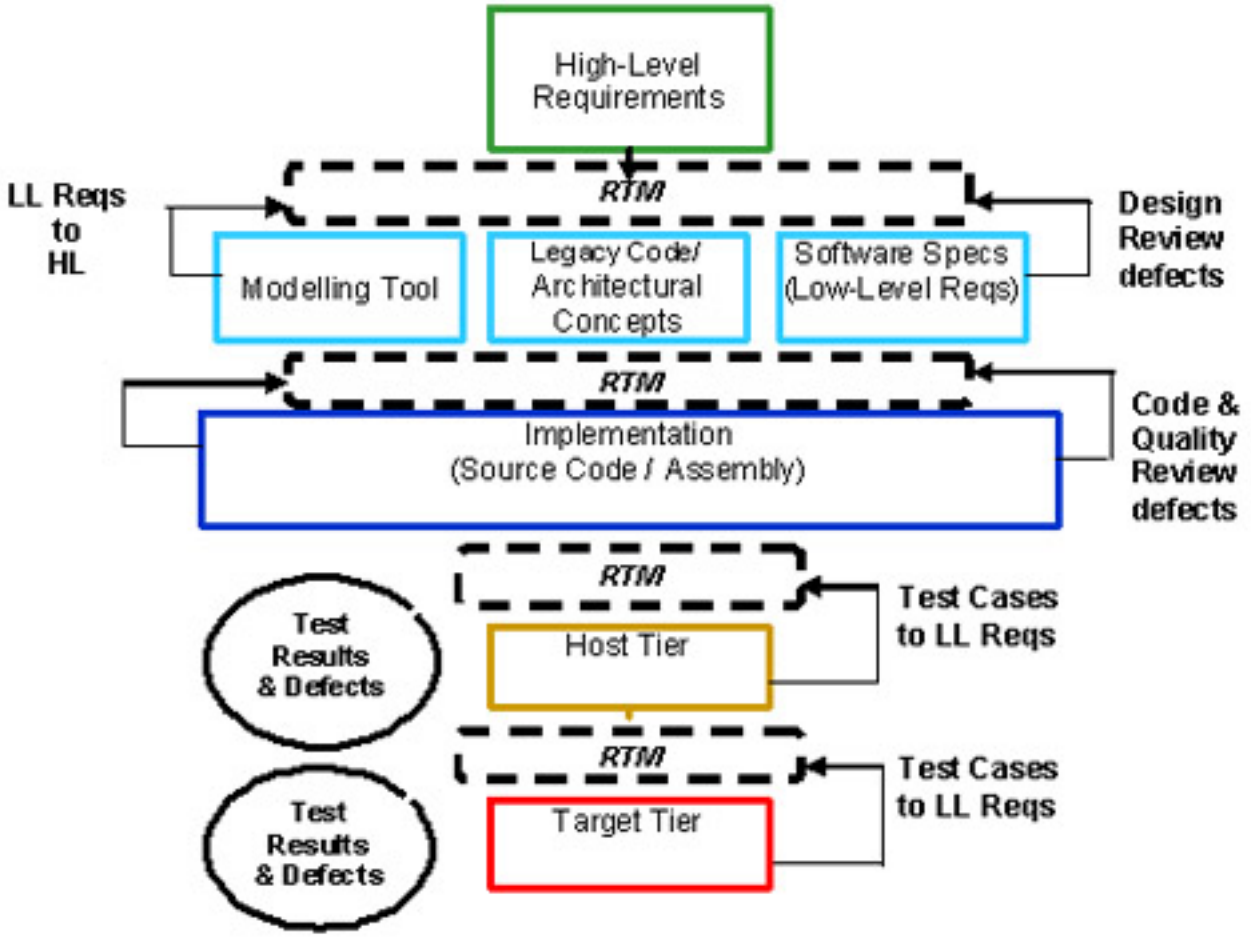
The first step in software fault management in the development process revolves around static analysis. To implement static analysis standards, the medical devices industry has looked to the automotive industry with its introduction of MISRA-C:1998, MISRA-C:2004 and the MISRA C++:2008 programming standards. The MISRA standards provide considerable assistance for ensuring software adheres to a strict quality model with the obvious benefits of the quality improvements that this engenders.

As a largely rules-based standard, however, MISRA satisfies only a portion of the static analysis picture. Also important are code complexity analysis and runtime error analysis. Tools that support all three areas of static analysis are instrumental in providing the first line of defense for producing software applications with the highest reliability and least number of faults.

Although static analysis of highly reliable software systems is a good first step, important areas of the software development lifecycle, such as requirements analysis, structural coverage analysis and unit testing, can be overlooked. Enhancing the software development process with tasks in these areas has been employed to great effect in other industries where safety-critical applications are required.

When looking at the software process requirements for this market, it is important to start with the FDA guidelines for software embedded in medical devices. These guidelines lay out the tasks required to meet software quality and reliability objectives, and these tasks relate very closely to the overlooked areas that we outlined above: requirements traceability, coverage analysis, and proper

verification and validation procedures. Let's look at these areas in a little more detail.



Requirements traceability process

Requirements traceability provides the backbone for a reliable software system, and starts with very early design and carries through all the way to latest stages of testing and verification, and even into the iterative process of release and bug fixing in the back end of a product's lifecycle. Through studies, it has been shown that in many cases 60-70% of software defects can be traced back to requirements traceability issues. A key component in the efficient implementation of requirements traceability is the automation of traceability throughout the development lifecycle and management of the Requirements Traceability Matrix (RTM).

Because many organizations use manual processes to track requirements through the lifecycle, certain results or artifacts can fall through the cracks as requirements are passed from the systems group to the design group to the development group to the testing and QA group. Automating the relationships between the inputs and outputs generated by the different phases and groups responsible ensures that all results and artifacts are properly updated and maintained. Not only that,

management also now has immediate, updated insight into which areas of a project might become bottlenecks as progress in different phases of development can be charted with a high degree of confidence. In no area does this have a bigger impact than in the verification and validation phase of a project.

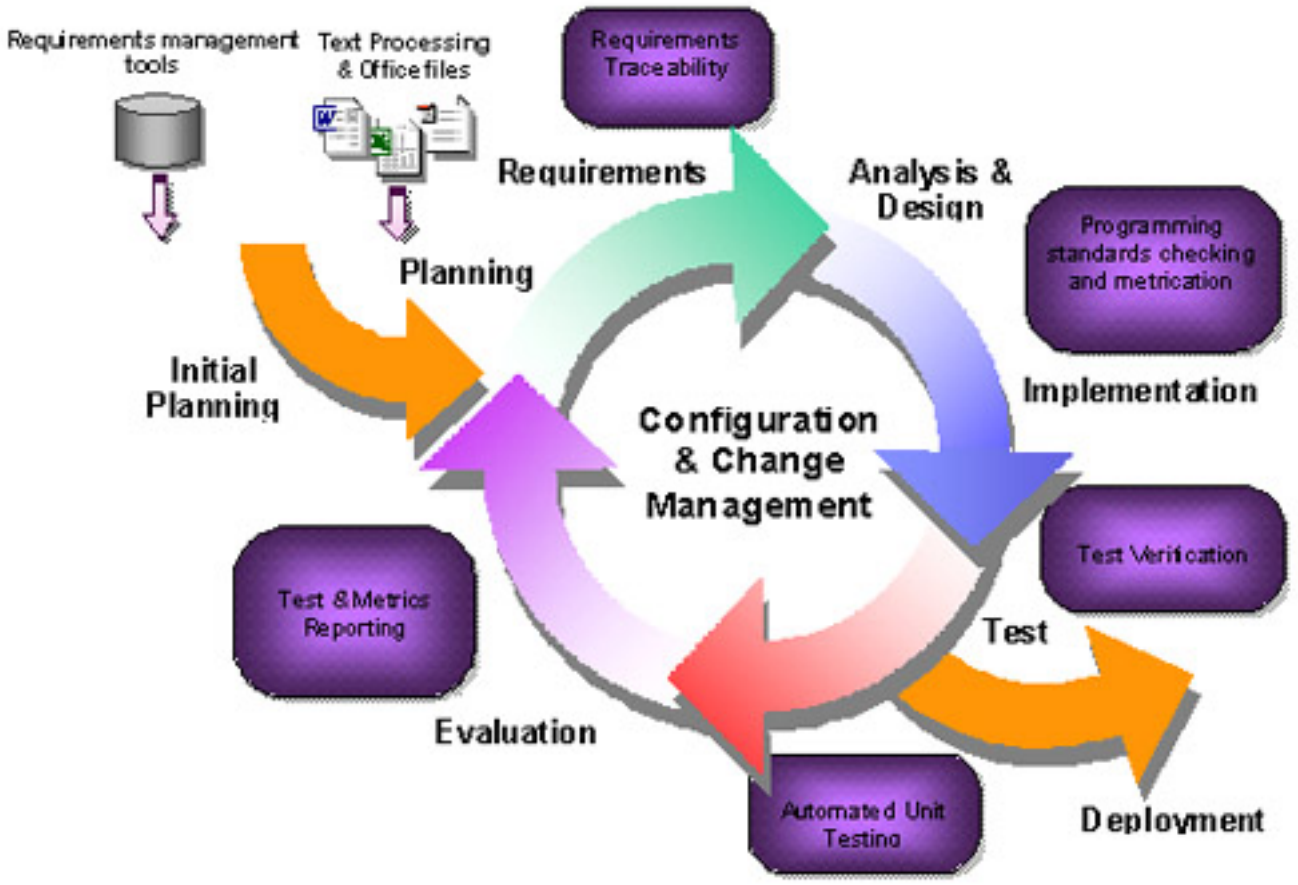
Automating the creation and updating of relationships between testing and verification activities with source code and higher level requirements can have a huge payoff in the overall quality of a software project. With tools that enable such automation, verification and validation results can automatically be assigned to procedures or functions in the source code that represent a requirement or set of requirements. Because this relationship is now automatically updated, it is no longer necessary for test or QA engineers to remember to update results or metrics to keep everything in sync. Not only that, the engineers can spend more of their valuable hours designing applicable tests rather than monitoring results.

Such automated systems can be applied both in implementation of top down (functional) testing and bottom up (structural coverage) testing. In fact, there are now tools on the market that will generate 100% automated structural coverage testing by just pushing a button. This testing can be used to generate coverage results at a variety of levels:

• Statement Coverage—this criterion requires sufficient test cases for each program statement to be executed at least once; however, its achievement is insufficient to provide confidence in a software product's behavior.
• Decision (Branch) Coverage—this criterion requires sufficient test cases for each program decision or branch to be executed so that each possible outcome occurs at least once. It is considered to be a minimum level of coverage for most software products, but decision coverage alone is insufficient for high-integrity applications.
• Condition Coverage—this criterion requires sufficient test cases for each condition in a program decision to take on all possible outcomes at least once. It differs from branch coverage only when multiple conditions must be evaluated to reach a decision.
• Modified Condition/Decision Coverage (MC/DC) —this criterion requires that each condition must be shown to independently affect the outcome of a decision.
• Path Coverage—this criterion requires sufficient test cases for each feasible path, basis path, etc., from start to exit of a defined program segment, to be executed at least once. Because of the very large number of possible paths through a software program, path coverage is generally not achievable. The amount of path coverage is normally established based on the risk or criticality of the software under test.
• Data Flow Coverage—this criterion requires sufficient test cases for each feasible data flow to be executed at least once. A number of data flow testing strategies are available.
For reference, other safety-critical applications require 100% MC/DC coverage for certain certifications, so testing to that same level for safety-critical health applications in medical devices would be a possible target. Using such automated structural coverage testing tools, combined with the automation of certain areas of functional testing infrastructure, it is possible to ensure requirements traceability all the way through to the validation and verification phase of development.

One last area of automation and traceability to concentrate on is regression analysis. Once the functional and structural coverage testing infrastructure is set up and producing valid results, it is imperative that this automated system continues to function effectively as the software application changes over time in response to modified requirements or bug fixes. This type of regression analysis can also be automated with the proper tools, so that the system will automatically know that certain tests need to be re-run or modified as a result of changes in certain parts of the source code. Again, this relates back to the traceability set up earlier in the process between high-level requirements, low-level requirements, source code, functional test results, and structural coverage test results.

As we have seen, there are many areas to explore to reduce the occurrence of software faults during the development lifecycle. However, as with any complex software application, medical device systems can still encounter software problems even with the best development process in place. In the case where a software problem can cause malfunctions that are life threatening, techniques such as using watchdog timers, designing in redundant processes, restarting threads or processes, or rebooting the entire system to help mitigate the risk of software faults. These methods can either be implemented purely in software or with a joint hardware/software approach.



Automated tools applied to the development model

Software quality and reliability in medical devices can only be ensured by implementing proper software development processes throughout the development lifecycle and designing runtime fault handling systems that can catch the eventual software problems that creep up in today's complex software systems. It is important for medical devices companies to look at both the current FDA software development guidelines and good practice models used in various industries for implementing many of the overlooked areas of the development process, such as requirements traceability and integrated verification and validation procedures. Key in enabling efficient adoption and use of these processes is the automation of these various processes using tools designed to eliminate many of the pitfalls with manual integration of the different phases and groups responsible for the entire software development lifecycle. Using these models and tools, companies producing medical devices can ensure higher software and product quality and reliability while promoting higher efficiency of their software development resources.

John Greenland is currently serving as VP Business Development for LDRA Technology, Inc., where he is responsible for certain sales areas, partnerships, and new market development. He possesses over 20 years experience in the embedded software development tools industry, working in various technical, sales, and marketing roles.