# Assurance and Certification of Software Artifacts for High-Confidence Medical Devices

Mark P. Jones
OGI School of Science & Engineering, Oregon Health & Science University
Beaverton, OR 97006
Telephone: (503) 748 7554, Email: mpj@cse.ogi.edu

Position Paper, April 2005

**Introduction** Advances in technology have enabled the development of new generations of computer-based medical devices that offer high levels of functionality and interoperability. The primary purpose for these devices, of course, is to improve the quality and the effectiveness of patient diagnosis, treatment, and monitoring. To many of their users—which includes both the caregivers and their patients—the notion of a 'high-confidence' medical device is an oxymoron: just as they expect a sterile dressing to be, indeed, sterile, or a needle to have a point, so they expect computer-based devices to work correctly and safely, trusting in the physicians, manufacturers, and industry and government agencies that develop them, build them, and regulate their use.

On the face of it, these are not unreasonable expectations. However, the evidence suggests that it will become difficult to meet these goals as the sophistication and complexity of medical device software continues to increase. For example, many general purpose, desktop computers are prone to software bugs that cause frequent failures and crashes, and a high proportion are not correctly configured or patched to provide protection even against known vulnerabilities. Large-scale, professionally administered systems also suffer from many problems, including high-profile virus and worm infections and the persistent rumors of leaked personal data from compromised e-commerce sites and health centers.

Unless we take steps to address these failures, there is every reason to expect similar problems with the software that controls medical devices. Some users accept the failure of a desktop computer as a minor irritation that can usually be remedied, apart from some loss of time and work, by rebooting the system. Clearly, however, this is unacceptable in the domain of medical devices where a failure could result in compromised patient safety or security, potential injury, or, in the worst case, loss of life.

**Software Validation and Assurance** Process-oriented software validation, as described by the Code of Federal Regulations (21 CFR 820), is just one component in a broad spectrum of issues that are required for FDA approval of a medical device. However, according to the published FDA guidance on "General Principles of Software Validation," 242 of the 3140 medical device recalls that occurred between 1992 and 1998 were attributable to software failures, and, in a point that we will return to later, 192 of those were caused by defects introduced when changes were made to software *after* initial production & distribution. Process-oriented techniques are extremely valuable, but these numbers, and other related reports, are clear indicators that there is a need to do more. In particular, artifact-oriented techniques, which focus directly on software artifacts, will become an increasingly important counterpart to techniques that focus on the processes by which it is constructed.

Software validation, like other parts of the Quality System Regulation, is driven by the need to protect patients, and to provide them with assurance that the medical devices used in their care are both safe and effective. However, there is also a real need to provide developers and manufacturers with tools to meet these stringent goals. Without such support, there is a real risk, driven by ethical concerns as much as the threat of expensive lawsuits, that innovation will be stifled and that the potential of new functionality—which should ultimately be in the best interests of patients—will not be realized.

**Candidate Technologies** For several decades, industry and academia have invested significant effort in exploring techniques for building software systems that are worthy of their designer's and user's trust. These include process-oriented methodologies, testing, formal methods, and programming language technologies.

In the past, there have been difficulties in scaling some of these approaches to cope with problems of real-world, engineering significance. Recent work, however, confirms that this is changing. Intel Corporation, for instance, is now a significant consumer of *theorem proving* technology, which it is applying in several different areas, including the verification of software and microcode for new microprocessor designs. Elsewhere, Microsoft Corporation is now preparing internally prototyped tools from its SLAM project—which is based on software *model checking*—for inclusion in the next release of the Windows Device Driver kit. The resulting toolkit will empower driver writers to verify their code against the important—but previously informal—contracts that are needed to protect against the critical failures that can result when buggy driver code is executed in kernel mode.

New programming language technologies also show considerable promise for increased software reliability as well as programmer productivity. In the Timber project at OGI, for example, we developed a *domain-specific language* (DSL) for configuring components in large, distributed systems, which resulted in significantly smaller code (by a factor of more than thirty in the largest examples) while also preventing hundreds of defects that were detected in a corresponding non-DSL version.

**Certification and Change** As mentioned previously, changes to previously developed software systems have been recognized as a major cause of medical device recalls. Indeed, 'change' is one of the only constants in software development where the operational requirements, underlying platform, and assurance needs can all vary significantly during the lifetime of a product. In some domains, such as aviation, where stringent standards for software certification are already in place, the costs of *recertification*, even after just a small change, have considerably slowed the adoption of new technology. Of course, safety critical systems, including medical devices, warrant a conservative approach. However, it would also be preferable to avoid a model that delays or discourages the introduction of beneficial new developments (perhaps even bug fixes) so as to avoid or amortize certification costs.

Several commercial software packages have been developed in support of the Quality System Regulations, providing process-oriented tools to track the evolution of a medical device in response to many different forms of data gathered during its design, evaluation, and use in the field. Software like this can be used to help in preparing the supporting documentation that is needed for FDA approval of a new medical device.

As we extend the scope of validation and verification to include artifact-oriented techniques, we will also need to extend these tools to help manage the complexity inherent in real-world software systems. From a programmer's perspective, such tools provide 'make'-like functionality for quality systems that support analysis and, as much as possible, automate the construction of evidence of validity. In the Programatica project at OGI, for example, we are developing a prototype tool like this that integrates a broad and open spectrum of assurance techniques in a development environment for security-critical applications. Among other features, the Programatica tools incorporate fine-grained, automated dependency tracking to reduce the cost of recertification. Tools like these provide an evolution path for the introduction of artifact-oriented methods. At the same time, work in this area will play a key role in informing the development of new standards for meaningful, tractable, and agile certification of medical devices.

**Platforms for Evaluation** Many researchers with a background in artifact-oriented techniques for verification and validation are drawn to the potential of applying those methods to assure the safety and reliability of medical device software and systems. To be effective, however, this community needs access to open, and representative experimental platforms that can be used: as case studies; as baselines for comparison and evaluation; as drivers for development and application of new tools and prototypes; and as a focus for integration with work on high-fidelity organ and patient models. Unfortunately, it is currently hard to find suitable examples, mostly as a result of commercial pressures in the device industry, including issues of licensing and IP. This is one key area where high-level government funding might be used to enable broader access to industrially relevant examples. While it might be easier to provide access to a suitable platform within the scope of a single research program, the ideal would be to follow the model of "open source." The Linux operating system—widely adopted by operating systems researchers, but also by many other 'normal' users—is a well-known and compelling demonstration of the benefits of leveraging a large community of interested developers and users. Although the initial costs would almost certainly be high, it would be hard to overestimate the potential long term benefit to device manufacturers, and, more importantly, to society as a whole.

---

**Biography** Dr. Mark Jones is an Associate Professor at the School of Science and Engineering at Oregon Health & Science University (OGI). His area of expertise is in the design, implementation, and application of programming languages. He obtained a doctorate (D.Phil.) from the University of Oxford in England, and has worked as an Associate Research Scientist at Yale University, and as a Reader at the University of Nottingham, where he founded and led a research group on Languages and Programming. Jones was the Principal Investigator on the DARPA-funded Project Timber, dealing with the development of new programming language technology to support the design of reliable, real-time embedded systems. He is now leading the Programatica project, which is using the construction of a microkernel implementation with strong security properties to demonstrate and inform the design of tools for evidence management and validation of complex, high-confidence software.