Complying with Software Regulations in the Medical Device Industry

The Food and Drug Administration determined that 24% of all medical device recalls in 2012 were because of software failures. One of the most notable software failures occurred in October 2011 with a well-known software security expert demonstrated how he could hack into an insulin pump made by one of the world’s largest and most well respected medical device manufacturers. Fortunately, the incident took place on stage at the annual Black Hat Briefings conference. Once the conference was over, an investigation was launched by the House Energy & Commerce Committee who commanded the Government Accountability Office to determine the safety of medical devices. Their conclusion was that some devices are not safe as we thought them to be and that many of the issues could be isolated to the source code in the medical devices.

Since then, the number of recalled medical devices due to software failures has increased. According to the FDA, they can recall between 13 and 75 devices per day. Most notably was an incident that involved a U.K based healthcare company. This software issue would unexpectedly fail to deliver oxygen, anesthetic vapor, and nitrous oxide to patients who were hooked up to the machine when its USB ports were occupied by other devices. With the demand for innovative products in the healthcare sector, software quality continues to be an important focus for medical device companies because of the regulations they must adhere too.
The Problem is Complexity

As systems become more capable, it becomes harder to test all of the ways they will be used in advance. Once you test software and fix all of the problems found, the software will always work under the conditions for which it was tested. The reason there are not more software tragedies is that testers have been able to exercise these systems in most of the ways they will typically be used. But all it takes is one software failure and a subsequent lawsuit to seriously damage a company’s reputation and possibly put it out of business.

Test-and-fix approaches are vital dynamic testing approaches. Whether performed on individual units or the entire system, these dynamic approaches share one common shortcoming: they all rely on test cases.

Test case scenarios are constructed from the same source documents that developers use, such as requirements and specification documents. These documents are much more comprehensive at defining what the finished product should do, rather than what it shouldn’t do. Developers inject about 100 defects into every 1,000 lines of the code they write.¹ Many of these defects will have no impact on the test case scenarios designed for testing.

If Quality Cannot be tested in, Then What?

According to the FDA’s Center for Devices and Radiological Health (CDRH), “Software quality assurance needs to focus on preventing the introduction of defects into the software development process and not on trying to ‘test quality into’ the software code after it is written.”

The CDRH’s guidance on the development of software for medical devices is intended to help medical device software companies comply with Title 21 of the Code of Federal Regulations, which is administered by the FDA.

The CDRH developed separate guidelines for software because of software’s inherent complexity. “The development process for software should be even more tightly controlled than for hardware, in order to prevent problems that cannot be easily detected later in the development process.” The CDRH acknowledges how seemingly insignificant changes in software code can create unexpected and very significant problems elsewhere in the software program, and states, “the software development process must be sufficiently well planned, controlled, and documented to detect and correct unexpected results from software changes.”²

The guidelines use the terms “verification” and “validation” (also referred to as “V&V”) to encompass software quality process requirements. While the CDRH admits that the terms are sometimes interchanged, validation generally refers to traditional, dynamic testing of the “by objective evidence.” Verification, on the other hand, refers to confirmation by examination.

². General Principles of Software Validation; Final Guidance for Industry and FDA Staff, January 11, 2002
In a software development environment, software verification is confirmation that the output of a particular phase of development meets all of the input requirements for that phase. Software testing is one of several verification activities intended to confirm that the software development output meets its input requirements.

**Other verification activities specifically listed include:**

- Walk-throughs
- Various static and dynamic analyses
- Code and document inspections
- Module level testing
- Integration testing

Code walkthroughs and peer reviews are a crucial part of a holistic approach to finding and removing defects in software. No single approach will provide 100% coverage. However, as Capers Jones points out in one of their studies, "A synergistic combination of formal inspections, static analysis and formal testing can achieve a combined defect removal efficiency levels of 99%."³ Where tool-assisted code review is in code and document inspections, as well as providing a central location for reviewing test cases/plans and the results of static analysis tools.

Measures such as defects, estimates of defects remaining, testing coverage, and other metrics are all used to develop an acceptable level of confidence before shipping the product.⁴ Software verification and validation are difficult because a developer cannot test forever, and it is hard to know how much evidence is enough. While safety is the primary measure, the V&V program should be sufficient to prove compliance.

The CDRH is clear that all code reviews must be in writing. “Otherwise, there is no proof it has been performed.” Statements about the code in general, specific lines, and specific issues, must all be tied to the person, time and date of their identification. If needed, this data should be presented as both comments and metrics to allow accounting of the development process. Firms may perform, manage and document the process manually, as long as they use “appropriate controls to ensure consistency and independence.” Source code evaluations should be extended to verification of internal linkages between modules and layers (horizontal and vertical interfaces), and compliance with their design specifications. Documentation of the procedures used and the results of source code evaluations should be maintained as part of design verification.⁵ The CDRH does not go into detail as to how code reviews and evaluations should be performed. While thousands of organizations have successfully implemented and defended peer code reviews successfully, many have failed.

**The difference most often comes down to poor implementation strategies that can be readily addressed:**

- **Reviews are too long.** After just a few hours, attention wavers and effectiveness decreases. All-day code reviews can seem almost painful. Keep reviews short, no more than one or two hours per day. In that time, developers will be able to review between 150 and 300 lines of code.

---

⁴ General Principles of Software Validation; Final Guidance for Industry and FDA Staff, January 11, 2002
⁵ Capers Jones, Combining Inspections, Static Analysis and Testing to Achieve Defect Removal Efficiency Above 95 %, January 2012
• **Reviews are seen as an additional task.** It is especially true when a review backlog builds up. Rather than let them become a bottleneck, make reviews a daily activity or take them as they come in. Let the code review serve as a break from a hard problem or a way to transition between tasks and a colleague’s comments as just their opinion. Make it easy for reviewers to annotate the specific code in question and to get other reviewers to weigh in.

• **Comments are seen as subjective.** It is easy to discount a complexity. Not surprising, this rate of review also provides the highest rate of defects identified per line of code (defects/LOC).

• **Remote reviews can be challenging.** Distributed teams are a given, and bringing teams together for reviews is at odds with the need for regular, brief reviews. Instead, facilitate remote reviews with tools designed for remote collaboration in general and peer code review, specifically.

• **Documentation is not automated.** The administrative burden of documenting, archiving and distributing this living document can be overwhelming. Use tools that make compliance documentation an automatic byproduct of the review.

Companies who have been successful with code review adoption have largely employed a strategy of facilitating the needs of developers first and then allowing the needs of the project and the company to naturally follow.

One of the most important contributions a company can make to successful adoption of code reviews are the tools it provides its teams. The right tool set will enable each development team to find its own best way to do code reviews, enabling a bottom-up approach to code review design and ensuring fuller achievement of potential gains and regulatory compliance.

**Some characteristics of a code review tool set to look for include:**

1. **Supports team-designed rules and processes.** Teams should be able to determine review intervals, workflows and specific tasks to be accomplished during the review while the tool supports and manages adherence.

2. **Supports each team’s preferred mode of interaction.** Whether side-by-side, remote real-time or asynchronous, or a combination, the team should decide. The tool should support before and after views of code and document changes and threaded contextual chat with references to files and line numbers.

3. **Supports the team’s preferred IDE.** To make reviews a “normal” part of developers’ work routines, developers should not need to leave their “regular” development environment to review code.

4. **Provides seamless integration with SCM systems.** To start reviews easily and expedite them, developers should be able to point to the code that needs review and have those files extracted automatically. Tools add tangible value to this process by automating the collection and distribution of these files.

5. **Ensures that documents are integrated within the review process.** A standardized peer review process enables all project-related documents (e.g. PDF, MS Office, HTML, images, schematics, intranet and web-based document management system) to be reviewed the same tool, making document reviews less frustrating for developers.

6. **Enables accurate reporting.** Meaningful metrics play a critical role in the reporting process to indicate progress and current status.
A peer code review process can be implemented within waterfall, Agile and other methodologies with equal success. Useful metrics used in meeting review milestones and audit requirements include man-hours spent in review, defect data, and lines of code inspected, as well as review approval and electronic signature status. The point to focus on is that not only will implementing peer code reviews make the products your company produces better; it will make the processes and the people that produce them better as well.

Code reviews are a powerful tool eliminating defects, but achieving compliance can be burdensome. Even in organizations where code reviews have been “adopted,” they are skipped as much as 30% of the time, primarily because they are inadequately supported. Too often, organizations believe they can have ad-hoc development processes, and then use an inspection process at the end to remove all defects. It just will not happen. Industry statistics indicate that for every four errors pulled out, one new error is injected. Therefore, only portions of defects are actually removed if the attempt is applied only to the end of the implementation process.

To approach zero defects, inspection must be an iterative process. For years, it was believed that the value of inspections is in finding and fixing defects. However in examining code inspection data, it becomes clear that inspections are beneficial for an additional reason. They make the code easier to understand and change. An analysis of data from a recent code inspection experiment shows that 60% of all issues raised in the code inspections are not problems that could have been uncovered by latter phases of testing or field usage because they have little or nothing to do with the visible execution behavior of the software.

Rather they improve the maintainability of the code by making the code conform to coding standards, minimizing redundancies, improving language proficiency, improving safety and portability, and raising the quality of the documentation — benefits which are not possible from automated testing.

In Conclusion

Peer reviews create an environment of shared understanding and collaboration. As developers review and comment on each other’s code, whether in real-time or asynchronously, they all get better. In the end, code review provides a platform for continuous process improvement, leading to improved standards, better developers, better efficiency, a higher quality finished product, and the peace of mind that comes from knowing the organization can prove compliance.

6. Mario Bernhart, Andreas Mauczka, Thomas Grechenig Research Group for Industrial Software (INSO) Vienna University of Technology Austria, 2010

SMARTBEAR

About SmartBear Software

SmartBear is the choice of more than four million software professionals and over 25,000 organizations in 90 countries that use its products to build and deliver the world’s best software applications. SmartBear’s user-centric application management solutions support key software delivery processes of development, testing, API readiness, and application performance management across desktop, web, and mobile platforms. Get started at www.smartbear.com.