Integrating Failure Mode Effect Analysis into the Medical Device Approval Process

Gerhard Steinke
Seattle Pacific University
gsteinke@spu.edu

Vania Kurniawati
Seattle Pacific University
vania@spu.edu

Jim Nindel-Edwards
Globys Corporation
jimne@ieee.org

Abstract
Medical devices that utilize computer software are becoming common place in today's health care environment. Device failures can have life threatening consequences. The medical device approval process issued by the FDA should enhance the software testing requirements. In this paper we suggest that Failure Mode Effect Analysis (FEMA) should be a standard component in the testing of software in medical devices that can have life threatening consequences.

Keywords: Software Testing, Quality Control, Failure Mode Effect Analysis, Medical Devices

Introduction

There is an increasing trend toward more automation across all domains, from traffic control and the manufacturing realm to aviation management and medical devices. In a sense, systems are becoming more intelligent and require less human intervention. Humans are increasingly becoming more reliant on these intelligent systems to manage both mundane and sophisticated functions in life. Intelligent systems have helped humans immensely in critical functions such as in aviation: more embedded systems have been added to automate operations in the cockpit, which have been proven to be beneficial in many ways such as reducing the stress and fatigue that pilots have to endure on longer trips. Additionally, auto-pilot functionality has also been proven to be better than human pilots at making critical decisions under severe circumstances. Intuitively, computers are better at producing consistent and objective results, whereas humans may be tempted to trust their 'gut feelings' when put under pressure.

Medical devices that utilize computer software are becoming common place in today's health care, ranging from insulin pumps, to devices that dispense drugs, and those that monitor heart rhythms. Many

medical devices have been used successfully to provide better patient care, often with costs savings, and in most situations actually address some areas of human driven errors (Berman, 2004).

At the same time software to program and/or control medical devices can, and have, introduced errors that affect patient outcomes. The errors are in part due to the hardware, the software, or a failure of the interface between the hardware and software in the medical device and the person using the device. In critical systems like these, where failures may result in substantial losses, a more rigorous safety standard is expected of the system software. More and better testing requirements and processes should ensure that more errors are identified before medical device products are approved and released.

The FDA has an approval process for medical devices with differences depending on the criticality level of the application of the medical devices. The testing requirements are not clearly specified in the FDA medical device approval process. In previous research we suggested that human computer interaction testing can improve device quality and user experience (Nindel-Edwards & Steinke, 2009). While human computer interaction testing is helpful and necessary, it is usually conducted late in the development process. Addressing test findings is often resisted because of the expense and timing concerns for fixing a product so late in the development cycle. In this paper we suggest Failure Mode Effect Analysis (FMEA) will catch serious errors and problems earlier on in the system development process—and should be part of the process before medical devices are approved by the FDA.

Medical Device Failures

A few examples of problems with the human computer interface of medical devices will provide some background for this paper.

The Therac-25 device was developed to provide radiation to cancer patients. The presence of numerous flaws in the software led to massive radiation overdoses, resulting in the deaths of three people (Leveson, 1993). It was poor user interface controls that caused prescription and dose rate information to be entered improperly (McQuaid, 2009).

A problem with a flow control knob is cited by the FDA (Sawyer, 1996): "A physician treating a patient with oxygen set the flow control knob between 1 and 2 liters per minute, not realizing that the scale numbers represented discrete rather than continuous, settings. There was no oxygen flow between the settings, yet the knob rotated smoothly, suggesting that intermediate settings were possible. The patient, an infant, became hypoxic before the error was discovered. Testing of this device should have identified this problem."

A volumetric infusion pump is a medical device that delivers intravenous fluids and medicine to patients. The Baxter Colleague triple channel infusion pump generates fault codes under the condition of changing a fluid supply at the same time the supply goes to zero (an understandable and appropriate behavior). Unfortunately this fault also stopped the other two channels from continuing to operate, causing a life threatening situation for patients. At least 9 patients' deaths have been attributed to miscommunication between the care giver and the software that runs these medical devices (Infusion Pump Recall, 2009).

The *Journal of the American Medical Association* (Koppel et al., 2005) reported on an examination of a hospital computer system and suggested that computers increased the error risk. They said, "…the problem is that hospital computer systems are not designed for the way real-life hospitals work." It appears that their recommendation is for more testing.

Further examples can be found in an article published by Dick Sawyer (1996). The complexity of technology and software is growing: The deployment of medical devices to the field such that "Caregivers and clinical engineers ... are becoming lost in a swirl of technology, and [we] face

unanticipated interference between devices." (Lee and Pappas, 2006). It is clear that some medical devices have failed, at least in part due to the software in the devices.

FDA Standards for Testing Medical Devices
The Federal Drug Administration has established standards for medical devices and classifies these based on criticality levels of the device usage, ranging from Class I (least critical) to Class III (life support/life sustaining) (FDA, 2006):

- Class I – Devices used in non-life sustaining tasks, subject only to "General Controls".
- Class II – Also non-life sustaining, but subject to both "General Controls" and "Special Controls" (FDA, 2006a).
- Class III – Devices that sustain or support life, subject to both types of controls and may not be introduced to market without FDA oversight and pre-approval requiring in many cases clinical trials.

Class III medical devices require the most scrutiny, and accordingly are the devices that cost the most to develop, test, and support. It is interesting to note that Class II devices can, and are, applied in situations where if not used, or tested appropriately, can result in serious injury and possible patient mortality. And yet the approval process for Class II is significantly reduced from Class III devices. Also, from the FDA perspective, it seems that the difference between hardware and computer controls with associated software is not material. The devices are simply classified as I, II, or III, based on usage. Perhaps the testing could also be based on the complexity of the device and the associated software—the more complex, the more testing should be required.

FDA Recommendation for Software Development
There are software standards published by the FDA in 2002 that provide guidelines and requirements for the software used in medical devices, regardless of class (Quality System Regulation, 2006). Various FDA documents specify that the "general controls" associated with software in medical devices follow existing standards for requirement driven software projects, e.g. the classic waterfall model overlaid by a risk based analysis termed "Level of Concern". This is summarized in the *Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices* documented in the following decision matrix (FDA, 2006):

The criticality of devices is divided into three classes: Class I, II and III. The software is divided into three areas as well: Minor Concern, Moderate Concern and Major Concern. There is some relationship between the two. That is to say that a medical device could be a class III (highest standards) yet the software associated with the device be only a "minor concern" if the software component was actually peripheral to the Class III rating. This in fact was the situation with the first model of the Therac as the hardware provided the protection and the associated software was of "moderate concern". When the hardware interlock was removed the software moved from "moderate" to "major" without the appropriate review of the software component.

SOFTWARE DOCUMENTATION	**MINOR CONCERN**	**MODERATE CONCERN**	**MAJOR CONCERN**
Level of Concern	A statement indicating the Level of Concern and a description of the rationale for that level.		
Software Description	A summary overview of the features and software operating environment.		
Device Hazard Analysis	Tabular description of identified hardware and software hazards, including		

	severity assessment and mitigations.		
Software Requirements Specification (SRS)	Summary of functional requirements from SRS.	The complete SRS document.	
Architecture Design Chart	No documentation is necessary in the submission.	Detailed depiction of functional units and software modules. May include state diagrams as well as flow charts.	
Software Design Specification (SDS)	No documentation is necessary in the submission.	Software design specification document.	
Traceability Analysis	Traceability among requirements, specifications, identified hazards and mitigations, and Verification and Validation testing.		
Software Development Environment Description	No documentation is necessary in the submission.	Summary of software life cycle development plan, including a summary of the configuration management and maintenance activities.	Summary of software life cycle development plan. Annotated list of control documents generated during development process. Include the configuration management and maintenance plan documents.
Verification and Validation Documentation	Software functional test plan, pass / fail criteria, and results.	Description of V&V activities at the unit, integration, and system level. System level test protocol, including pass/fail criteria, and tests results.	Description of V&V activities at the unit, integration, and system level. Unit, integration and system level test protocols, including pass/fail criteria, test report, summary, and tests results.
Revision Level History	Revision history log, including release version number and date.		
Unresolved Anomalies (Bugs or Defects)	No documentation is necessary in the submission.	List of remaining software anomalies, annotated with an explanation of the impact on safety or effectiveness, including operator usage and human factors.	

The key reports associated with the FDA specifications are listed in the first column above:

- Software Requirements Specification (SRS),
- Architecture Design Chart,
- Software Design Specification (SDS),
- Traceability Analysis,
- Software Development Environment Description,
- Verification and Validation Documentation,
- Revision Level History, and

- Defect Reports.

The key deliverable is the Verification and Validation (aka V&V) documentation. While a common term in the software industry, we'll reflect here on the FDA's definition (FDA, 2002):

- Software verification provides objective evidence that the design outputs of a particular phase of the software development life cycle meet all of the specified requirements for that phase. Software verification looks for consistency, completeness, and correctness of the software and its supporting documentation, as it is being developed, and provides support for a subsequent conclusion that software is validated. Software testing is one of many verification activities intended to confirm that software development output meets its input requirements. Other verification activities include various static and dynamic analyses, code and document inspections, walkthroughs, and other techniques.
- Software validation is a part of the design validation for a finished device, but is not separately defined in the Quality System regulation. The FDA considers software validation to be "confirmation by examination and provision of objective evidence that software specifications conform to user needs and intended uses, and that the particular requirements implemented through software can be consistently fulfilled." In practice, software validation activities may occur both during, as well as at the end of the software development life cycle to ensure that all requirements have been fulfilled. Since software is usually part of a larger hardware system, the validation of software typically includes evidence that all software requirements have been implemented correctly and completely, and are traceable to system requirements. A conclusion that software is validated is highly dependent upon comprehensive software testing, inspections, analyses, and other verification tasks performed at each stage of the software development life cycle. Testing of device software functionality in a simulated use environment, and user site testing are typically included as components of an overall design validation program for a software automated device.

An interesting point here is the statement "that software specifications conform to user needs and intended uses". This assumes that the Software Requirements Specifications (SRS) in and of itself is correct, and Verification and Validation (V&V) assures the device software matches the specifications. What if the SRS itself is flawed in one or more areas? The very best we can say is that we've assured (through the V&V process) that the software perform well to the requirements (SRS) without highlighting the underlying flaw in the SRS.

So we have two areas to examine, that are not included in the FDA requirement:

- Are the requirements indeed accurate and complete?
- Has the system been designed appropriately to achieve the requirements?

The software quality assurance perspective is to "drive quality upstream" by verifying the correctness of the requirements, in this instance the SRS and Software Design Specification (SDS). It is especially in answering the second question that FMEA can be very helpful. Then the Verification and Validation process can assure that the system meets the requirements.

Failure Mode Effect Analysis (FMEA)

To find design and conceptual software faults, a Quality Engineering methodology called Failure Mode Effect Analysis (FMEA) can be utilized. We can answer the question: Has the system been appropriately designed to achieve the requirements? Conceptual software faults, or the inherent defects in the logic of the software models, are usually not components of standard testing methodologies. That is why FMEA is so beneficial.

FMEA proactively sets out to find out what exactly can go wrong, the cost of such failures, and the cost to mitigate these failures. At the heart of FMEA is the process of re-evaluating a preliminary design with the conscious mind that the design is flawed.

FMEA is a very simple tool, but what makes this tool very powerful is the fact that it incorporates multiple viewpoints into system design. Fred Brooks in his best-seller, The Mythical Man-Month: Essays on Software Engineering, describes an integrity goal as "[dictating] that the design must proceed from one mind or from a very small number of agreeing resonant minds." (Brooks, 1982). FMEA provides a framework for more people with different backgrounds, a design committee, to do exactly that. The goal is that those who are involved in this analysis are not new to the field – they have probably designed similar systems in the past. Consequently, they should be able to point out what components of the system are most likely to fail and what components are most likely to inflict the most damage if they fail. FMEA is successful only if the design committee is composed of experts from different functional groups that are committed in the design process and have enough authority to make the decision or outcome stick.

Unlike other risk management tools that often require statistical savvy, FMEA is relatively intuitive. It is often described as a design tool, because the idea is to correct the conceptual faults of the system or product – or in other words, to correct the defects before they are created.

According to Peter L. Goddard there can be two types of software FMEA: system-level software FMEA and detailed software FMEA. He describes the system-level software FMEA as being "based on the top level software design: the functional partitioning of the software design into … modules." (Goddard, 2000) This system-level software FMEA is an ongoing guideline that controls the risk of defects through product or process design requirements.

The detailed software FMEA is introduced later in the design process, "once at least pseudo code for the software modules is available." (Goddard, 2000) Detailed software FMEA is a testing tool that checks the validity of the risk mitigation mechanisms introduced earlier in the process by the system-level software FMEA.

Goddard recommends a process for developing a FMEA. First software hazards or problem areas must be identified. Being logical entities (as opposed to physical entities), hazards must be identified and translated into software terms prior to the analysis. After a list of all potential hazards is created, Goddard suggests performing a fault tree analysis to investigate the potential causes of hazards. The end results of the fault tree analysis should be a value set associated with each hazard cause, which is then identified as a software hazard.

The next step in FMEA is to assign a numerical rating of 1 through 10 for the following variables, which are listed in descending order of importance, to each of the software hazards:

1. Severity. A rating of 1 in severity indicates an insignificant damage and a rating of 10 indicates a catastrophic outcome resulting from the hazard.
2. Occurrence. A rating of 1 in occurrence indicates a hazard that is unlikely to happen and a rating of 10 indicates a hazard that inevitably will happen.
3. Detection. A rating of 1 in detection indicates that occurrence of a particular hazard is detectable (i.e. the hazard will be visible before the damage is done) and a rating of 10 indicates that no control exists to detect such occurrence (i.e. the damage will be visible once damage is done).

These variables are then multiplied together to produce a Risk Priority Number (RPN). Based on these assigned RPN's, the FMEA committee is then aided in the decision to select what hazards to focus their design fixes/mitigation planning on.

Part of what makes FMEA a successful risk mitigation tool is that it is a structured process for identifying areas of concern. It helps document, plan and track risk reduction activities proactively – thus reducing costs of fixing defects later. More importantly, FMEA is a group effort, which means the action plans that result from FMEA reflects the commitment and ownership of a cross-functional group of people (i.e. "all necks in the noose").

FMEA ultimately provides the documentation that the design committee has done their due diligence in re-evaluating the conceptual integrity of their design. Additionally, it can also serve as a proof of evidence-based decision making. A well-constructed FMEA worksheet can show the logical reasoning behind why mitigation plans are focused on a particular area of the software system.

FMEA has been recommended for evaluating critical systems in various industries and standards, including: SAE J1739, ARP 5580, ISO 9001:2000, MIL STD 1629A, MIL STD 882, IEC 61508 (Goddard, 2000). FMEA is not a substitute for rigorous testing mechanisms that check for completeness and accuracy of the systems, but it can be a powerful tool to improve quality, reliability and safety of a process or product.

Therefore the recommendation that FMEA should be a part of the FDA testing requirements for medical devices that impact lives. The use of a cross-functional group to evaluate the design is especially appropriate. Medical devices put a heavy responsibility on the design team responsible for the development and implementation of medical devices. The medical device design teams is not are not licensed care givers but computer hardware, and software developers. Theirs is the job of both discerning not only the requirements and anticipated uses of medical devices, but the potential for boundary, outlier, and simply misuse of these devices. It is worthwhile to repeat Marc Green's comment: "Faulty design could be construed as negligence on the part of the designers."

The goal of FMEA is to verify at the start of the design phase the components of the system that are most likely to fail and what components are most likely to inflict the most damage. Concentrating on what, and how, a critical heath care application is intended to respond is certainly a factor in the design, development and delivery of functionality. FMEA and its associated testing of medical device design and performance can assure that the device delivers its critical care under the anticipated circumstances and continues to delivers critical care when circumstances are less than ideal. Any of us can suggest that if events had only gone as we expected them to unfold, everything would "be ok". FMEA provides for an evaluation of the most serious failures and hazards.

Conclusion and Future Research

FMEA is being successfully used in many commercial settings to develop better products. Similar to critical software components in the aviation and automotive industry, FMEA can help in evaluating critical components of medical devices. The FDA should require medical devices manufacturers to use FMEA during the development of medical devices.

A challenge for the FDA as well as a medical device manufacturer is the question: What is appropriate and sufficient testing? How much testing is enough? Often it is impossible to test all possible conditions and situations. Certainly the testing should relate to the FDA's definition of Class I, II and III devices. Perhaps there should also be some factor as to the complexity of the software in a particular device, with more complex software functions calling for more testing. We would also suggest that an independent organization verify the testing performed by the manufacturer and provide some level of certification as to the testing that has been performed. Further research is necessary to determine certification levels and their relationship to the criticality levels of the medical devices.

References:

Berman, A. (2004). Reducing Medication Errors through Naming, Labeling, and Packaging, Journal of Medical Systems, Volume 28, Number 1, pp. 9-29.

Brooks Jr, F. (1982). The Mythical Man-Month: Essays on Software Engineering. Chapel Hill: Addison-Wesley.

Denger, C., Trapp, M., & Liggesmeyer, P. (2008). SafeSpection: A Systematic Customization Approach for Software Hazard Identification in Computer Safety, Reliability and Security, Lecture Notes in Computer Science, Springer.

Domis, D., & Trapp, M., (2008) Integrating Safety Analyses and Component-Based Design in Computer Safety, Reliability and Security, Lecture Notes in Computer Science, Springer.

FDA (2002). General Principles of Software Validation; Final Guidance for Industry and FDA Staff, http://www.fda.gov/downloads/MedicalDevices/DeviceRegulationandGuidance/GuidanceDocuments/ucm085371.pdf

FDA (2005). Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices, May. http://www.fda.govhttp://www.fda.gov/downloads/MedicalDevices/DeviceRegulationandGuidance/GuidanceDocuments/ucm089593.pdf

FDA (2006). FDA Regulatory Requirements and Approvals, http://www.cdaservices.com/fdaapproval.htm

FDA (2006a) Device Classification Panels, http://www.fda.gov/MedicalDevices/DeviceRegulationandGuidance/Overview/ClassifyYourDevice/ucm051530.htm#regs

Functional Safety and IEC 61508 (2005). http://www.iec.ch/zone/fsafety/pdf_safe/hld.pdf

Goddard, P. L. (2000). Software FMEA Techniques, Proceedings Annual Reliability and Maintainability Symposium, pp. 118-123, New York.

Gosbee, J. and Ritchie, E. (2007). Human–Computer Interaction and Medical Software, *Interactions*, July/August.

Green, Marc (n.d.) - Error and Injury in Computers and Medication Devices, http://www.visualexpert.com/Resources/compneg.html

Infusion Pump Recall (2009). http://www.baxter.com/about_baxter/press_room/documents/2009/03_11_09_colleague.pdf

Koppel, R., Metlay, J., Cohen, A., Abaluck, B., A. Localio, R., Stephen E. Kimmel, S., et al. (2005). Role of Computerized Physician Order Entry Systems in Facilitating Medication Errors (March) - http://jama.ama-assn.org/cgi/content/full/293/10/1197

Lee, I. and Pappas, G. (2006). High-Confidence Medicate Devices Software and Systems, *IEEE Computer*, 39, no. 4), pp 33 – 38.

Leveson, N. (1993). An Investigation of the Therac-25 Accidents. *IEEE Computer*, 26, No. 7 (July):18-41. http://courses.cs.vt.edu/~cs3604/lib/Therac_25/Therac_1.html

McQuaid, P. (2009). Software Disasters: What Have We Learned? *California Polytechnic State University SQP* VOL. 11, NO. 3, ASQ.

Nindel-Edwards, Jim & Steinke, Gerhard, Integrating Human Computer Interaction Testing into the Medical Device Approval Process, IIMA Conference, Houston, Texas, October 2009.

Quality System Regulation (2006). 21 CFR Part 820 - http://www.gmp1st.com/mdreg.htm

Sawyer, D. (1996). Do It By Design - An Introduction to Human Factors in Medical Devices, http://www.fda.gov/downloads/MedicalDevices/DeviceRegulationandGuidance/GuidanceDocuments/ucm095061.pdf