Software Safety for Every Phase of Software Development

by David A. Vogel, Ph.D
Intertech Engineering Associates, Inc.

as published in Biomedical Instrumentation & Technology, July/August, 2006

Where does risk management belong, at the beginning of a project or at the end? Many managers choose one or the other, or both. The correct answer is that risk management needs to be considered throughout the software development life cycle. When the risks include the safety of patients or the users of the software, the methods need to be more formal. This article will explain why, and then will provide the how and the how much.

Never heard of software safety? Well, get comfortable with it, because it is here to stay. *Software safety* is a term that originated in the aviation industry to refer to the collection of definition, design, implementation, analysis, and validation activities that are associated with software development for safety-critical software. The medical-device industry is an additional player now leading the charge toward adopting such terminology, as well as the techniques used to develop software that is safe for its intended use.

The medical-device industry's interest in software safety has evolved consistently since a number of cancer patients received excessive doses of radiation in the Therac 25 incident of the late 1980s. Until then, software was treated much like any other component of a device. The Therac tragedy and the subsequent analysis of what led to the defect that caused the overdoses made it clear to regulators that software errors were very much to blame. Furthermore, it became increasingly clear that the way software defects and flaws in the software development process lead to medical device failure is very different from the way hardware defects lead to device failure. This brought a heightened level of scrutiny to medical-device software. Certain controls have been suggested that are special to software to assure the safety of the associated medical device.

COMPANY PROFILE

Intertech Engineering Associates, Inc.

Address: 100 Lowder Brook Avenue
Suite 2500
Westwood, MA 02090
www.inea.com - (781) 801-1100

Industry: (Electro)Medical Devices

Services: Assessments
Training
Consulting
Hands-on Engineering

Skills: Product Design
Risk Management
Requirements Engineering
Electronics Development
Software Development
Software Verification and Validation
Production/Quality System Software Validation

Software safety involves nothing more than good software engineering practices. Risks need to be assessed and managed, control measures need to be engineered into the product, and the efficacy of the control measures needs to be validated. When safety is involved, the risks of concern are the risks of harm to users, patients, bystanders, or the environment.

The methodologies described here apply to any "high risk" software. Business can benefit from these methodologies by considering the risks of their confidential data being compromised, the risk of insecure electronic financial transactions, the risk of loss of privacy of clients' medical records, or other identity data, or simply the risk to the business's reputation if a software product fails.

Background

What's Different About Software?

The Food and Drug Administration's *General Principles of Software Validation; Final Guidance for Industry and FDA Staff* (January 11, 2002) addresses this question in detail. Briefly paraphrasing from this guide:

- Software does not wear out. When software fails, it is due to a design or implementation defect that has always existed. The defect was designed in!
- Software fails without warning. There are no such things as intermittent failures, brownouts, and so forth. Software fails in the field because it is subjected to inputs or combinations of inputs that were not anticipated and/or were not tested during development. Latent defects exist before release of the product and may only be triggered or recognized once they are in broad use.
- Software can be more complex than hardware. It is common for device software to be hundreds of thousands or millions of lines long. Device software may be integrated with commercial off-the-shelf software such as operating systems that can easily reach similar sizes. It is difficult to test all the software in a device, and nearly impossible to test all combinations of inputs and branching.
- Software is changed too easily. Attempts to make last-minute corrections can lead to disastrous results (see next item). Well-intentioned developers, integrators or manufacturing personnel can make changes invisibly to software that is delivered as part of a device.
- Seemingly insignificant changes in one area of software functionality can lead to disastrous defects in unrelated areas of functionality.

Unlike hardware that can wear out and break down in the field, software's failures are created during the design and implementation of the software. Consequently, the regulators have concluded that tighter controls are necessary during the design and development of software. Because the regulators are concerned particularly with the safety and efficacy of medical devices, they have recommended that risk analysis and risk management should be a requisite part of the development and validation of medical devices.

Risk Management Standard Provides a Process

A standard was released in 2000 to guide the industry in the risk management process *(ANSI/AAMI/ISO 14971:2000 Medical Devices--Application of Risk Management to Medical Devices).* Although the standard did not address software explicitly, it did specify a process that could, in general terms, be applied to the management of risks that are introduced by software in medical devices.

Briefly, this standard defines a risk management process as:

- Risk Analysis - Identify harms that could be imposed upon the patient, user, or environment through the use of the device. Hazards that could possibly lead to the harms are itemized along with perceived causes of those hazards.
- Risk Evaluation - Risk is a combination of the severity of a hazard (and resulting harm) and the likelihood that hazard will occur.

Determinations are made as to whether the risk is great enough to need to be controlled.

- Risk Control - Design Controls for those risks that are severe enough and/or likely enough to happen that they require control.
- Residual Risk Analysis and Evaluation - Reevaluate the risk after the control measure is in place. If the residual risk is still not acceptable, design different or additional controls to further reduce the risk to acceptable levels.

How Does Risk Management Apply to Software?

Analyzing the risk of hazards that are initiated by software, or where software is one of several contributing factors, is different from hazards that can be caused by hardware that breaks or wears out in use. Assessment of the severity component of risk is much the same as it would be for hazards that are initiated by hardware or the operator. The severity of the resulting harm depends little, if at all, on the causes that initiated the chain of events that led up to the harm. However, the assessment of the probability contribution of risk is a problem when software is one of the factors contributing to a hazardous event.

Factors that could affect the likelihood of a potential harm occurring due to a failure of the device software include:

1. Detectability of the result of the failure
2. The existence of downstream risk control measures to correct the failure or mitigate the harm that could result from the failure
3. The number of subsequent or simultaneous events that must occur to ultimately result in harm (multiplied probabilities)
4. The likelihoods of the above mentioned subsequent or simultaneous events
5. The number of parallel paths of subsequent events and their respective probabilities that could lead to harm (additive probabilities).

In certain cases, harm may only result from a software failure if certain other events occur subsequent to or simultaneously with the software failure. The likelihood of these downstream events in the causal chain will affect the likelihood that a software failure will result in harm.

If a downstream event in the causal chain is itself a failure (of the process, device, care system, etc.) of low probability, then the overall probability of the failure resulting in harm is reduced. Further, as more low-likelihood events (such as failures and intentional misuse) are required in the causal chain for an initiating software failure to result in harm, the overall likelihood is reduced even further, because the probabilities of each required event multiply, thereby resulting in very low likelihoods.

Another approach for dealing with safety-critical software is to deal with the severity of the harm resulting from the software failure and to assume that the software will fail. Because harm from device software failure is always delivered via the medical device system, there will be a number of design opportunities to prevent the failure, to detect and correct the failure mode, and/or to reduce the severity of the risk (*i.e.* mitigation). These can be exploited as risk control measures that deal with an assumed failure of the software. The residual risk assessment then focuses on the residual severity of the hazard, rather than trying to estimate likelihood.

To put the above concepts into the context of a widely publicized (though nonmedical) disaster caused by a software failure, consider the Mars Polar Lander, which was lost when it crashed into the surface of Mars in December 1999. A software failure misinterpreted a signal from a sensor on the landing leg to mean that the lander had touched down when, in fact, it was still more than 100 feet from the surface. This led to a premature shutdown of the descent engines and the ultimate destruction of the spacecraft. This program and another failed Mars mission, the Mars Climate Orbiter, had a combined cost of $328 million.

If during the design of the Lander, a risk assessment had been conducted, it might have concluded that one hazard could be the premature shutdown of the descent engine, which would lead to the harm of destruction of the spacecraft. Could the failure have been detected? Yes, the flawed sensor reading could

have been cross-checked against sensors on the other legs, as well as against the altitude and velocity sensors. The result could have been a determination that the craft was still in motion above the surface of Mars and the shutdown command could have been overridden.

Downstream measures could have been put in place to ignore a faulty sensor reading or to attempt to reread the sensor. Either measure would have led to a successful landing, even though a software flaw existed.

Controlling Risk in Devices Controlled by Software

With the risk evaluation complete, attention turns to identifying and designing risk-control measures for those risks of intolerable severity and likelihoods that cannot be ignored.

It is useful to think of risk control measures (RCMs) in classes. Risk control can be accomplished by controlling a risk's severity, likelihood, or both. The classes of RCMs are ranked in order of their ability to control and/or to contain an identified risk. The medical-device industry often (and perhaps incorrectly) refers to all RCMs as *mitigations.* The following ranking of RCMs (in descending order) makes it clear that mitigations are but one class of RCM, and are not among the preferred classes of RCMs.

1. *Inherently safe design is the best RCM.* The design of the system is such that an identified harm and the hazards that might lead to that harm are made nearly impossible by system design. For example, a blood warmer has an identified harm of overheating the blood. Hazards that lead to that harm include improper timing of the heater, heater stuck on, failed termperature sensors that cause the software to continue heating, and so forth. An inherently safe design would be a heater that cannot reach a temperature over 98.6°F. Regardless of how long the software causes the heater to run, the blood never exceeds 98.6°F.

2. *Preventive measures are the next best risk control.* The events that lead up to a hazard may be well known. Controls that monitor for those conditions and alert or take controlling action to avert the imminent hazard are preventive in nature.

   Medical infusion pumps often are designed to prevent the hazards related to over-infusion or under-infusion of drugs. Consider just the specific hazard that the motor runs out of control due to a master processor failure. Often a slave (or safety) microprocessor is incorporated into the design that communicates with a master processor to monitor expected motor movements. The slave processor also monitors the existence of predictably timed communications from the master processor. If communications fails, it could be assumed that there is a likelihood of a master processor failure that may result in unpredictable behavior of the delivery motor. The slave processor detects the hazardous situation and puts the pump into a safe state (*i.e.* with the motor turned off). The potential for a runaway motor is detected, and the hazard is prevented.

3. *Corrective measures are closely related to preventive measures, but are activated when the hazardous event has already begun.* Take, for example, a defibrillator. An identified harm is failure to discharge when triggered. A hazard that could lead to this is a microprocessor that is stuck in a non-operative condition. A watchdog timer can monitor the microprocessor and can detect when the microprocessor is in a non-operative state. If the watchdog time reinitializes the microprocessor, it would have corrected the situation and would have averted the harm. The hazard (or hazardous situation) had already occurred when the microprocessor got stuck, although the problem had not yet resulted in harm.

4. *Mitigating measures or mitigations are RCMs that attack the severity of a hazard.* Once a hazardous event has occurred, mitigating measures are relied upon to reduce the severity of the resulting harm. For example,

an anticipated harm for an electrical stimulator is shock or burning from excessive current. If adequate controls from an inherently safe design, or preventive or corrective measures, are not possible, then one could design a fuse that would stop the current once the overcurrent condition is met. Presumably, the limit on the fuse would be slightly higher than the operating range for the stimulator. The fuse blows only when the stimulator operates outside the intended operating range. In other words, the hazard has begun already, but the severity is reduced because the fuse quickly stops all stimulation.

5. *Labeling and/or training is at or near the bottom of the list as RCMs for obvious reasons.* Labeling is often not read and is language-dependent. Training might or might not be effective to control risks, but it is difficult to keep training current, because turnover in the user environment might be high.

Notice that testing was not included in the above list of risk-control measures, although it is commonly used in hazard analyses as a mitigation. Testing does nothing to control the severity of a risk. It does reduce the likelihood somewhat, but without a way of quantifying the original or residual likelihood of software failure, who can tell if the risk has been sufficiently reduced? Unfortunately, the only definitive answer to this is that if the software fails in the field, one could be led to believe that it was not tested enough.

After the Mars Polar Lander disaster mentioned above, an investigation was conducted to determine the technical and process flaws that led to the disaster. The commission concluded that the faults were due to inadequate testing and oversight.

This type of logic is flawed, and indicates an overdependence on testing. The testing was inadequate *because* there was a failure of the software. If the Lander had not crashed, would the commission still have concluded that the testing was inadequate? Yet, if the test team had increased their test efforts a hundredfold and the Lander had crashed, would the conclusion still have been "inadequate testing"?

The point is simply that nobody knows how much testing is enough testing. If software is responsible for an application whose failure would result in severe consequences, then testing alone is not adequate to assure the proper operation of the application. Risk control measures must be designed into these systems.

How Does Software Safety Relate to Risk Management?

Without dwelling on semantics too much, it is important to define some terminology. The terms *software hazard analysis* and *software hazard management* are commonly used in the industry. These terms are obsolete and should be avoided. *Software is never itself a hazard because there is never a direct connection between software and harm.* Software can be a cause or contributing factor to a hazard, but it can never be the hazard itself.

Also to be avoided is the concept of *software risk management* or any other term that would imply that the risks imposed by software can be considered outside the total-system risk analysis. To be clearer, risk management is a system-level process. Obviously, in identifying harms and hazards one must be analyzing at the system level. Software introduces unique risks because of its unique properties and deserves special treatment within a system-level risk management process.

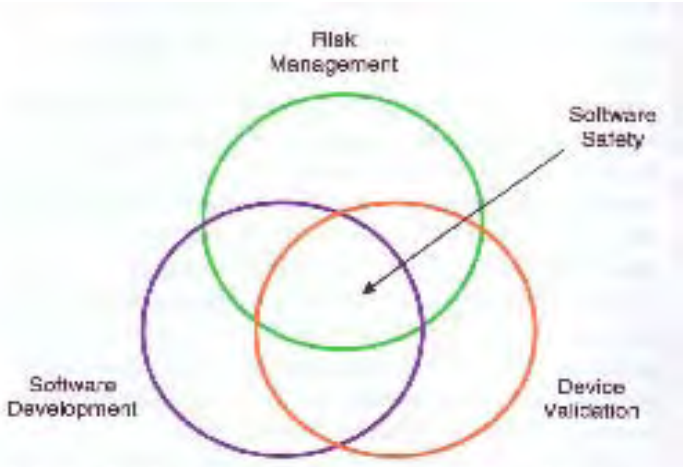

Figure 1. *Software safety both includes and excludes elements of risk management, software development, and device validation.*

Software is not just a source of risk. It is also often part of the solution. Risk-control measures can be implemented in software to control risks that are imposed by the system hardware, other software, or even the users.

Software Safety is the collection of activities that assure the safe operation of software in a device. The activities include risk management activities, software development activities and software validation activities. Figure 1 represents more clearly the relationships of software safety to these activities.

Software safety is the intersection of risk management, software development, and device validation. As the figure shows, there are risk-management activities that are not related to software safety, just as there are device-validation activities unrelated to software safety. All software is considered in the risk management process, but only safety-critical software is controlled by the software safety activities.

Software safety includes all the activities that build confidence that software is not a cause of or contributing factor to a device hazard that leads to harm. Furthermore, if the software itself is used as a risk-control measure, software safety activities provide documented, reasonable assurance that the software-implemented RCMs provide the level of control they were designed to provide.

Figure 1 also helps explain what software safety is *not*.

- Software safety is not just testing the device to come to the conclusion that it is safe, although testing is an important part of the verification and validation of risk-control measures implemented by or for software. It includes analysis, evaluation, and design of risk-control measures.

- Software safety is not a single analysis tool like failure modes, and effects analysis, fault tree analysis, or other similar analyses, although these analyses are an important part of the risk-assessment component of risk management, which is part of software safety.

Software safety is not a one-time event conducted at the beginning or end of a development project. It is a collection of activities that are performed throughout the life cycle of the product, and more specifically, throughout the life cycle of the device software.

Software Safety Throughout the Development Life Cycle...and Beyond

At the beginning of a device development project, nobody can predict all of the potential hazards and their causes that, ultimately, could lead to the harm of a patient, user or the environment. Part of practicing software safety is the iterative application of risk-management processes at each phase or activity of the software-development life cycle. It makes no sense to try to analyze failure modes of yet-to-be-designed components in the early concept phases of a development project. However, if nothing is done until the end of the project, retro-implementing risk control measures into a system that is almost complete can be very costly and can lead to unexpected defects.

Table 1, though far from being all-inclusive, suggests some activities that should be considered as the development of a device progresses through the life cycle. As more detail about the design and implementation of the device evolves, more sophisticated tools and methodologies are used to predict causes for hazards.

Table 1. Risk Control Over the Development Life Cycle of a Device

Concept

- Identify Intended Use for the Device
- Identify users, use environment
- Identify potential harms to patient, user, or environment
- Preliminary hazard and cause analysis
- Preliminary itemization of risk control measures (RCMs)

Requirements

- Identify software requirements for software implemented RCMs
- Identify non-software RCMs for possible software failures
- Review & revise activities from prior phase using new details from requirements activity

Design

- Designs for software RCM requirements
- Review software designs for robustness for unexpected situations (e.g. hardware failures, communication errors, unexpected inputs) to anticipate failure modes that could cause hazards.
- At the architectural design level and module design level review for failure modes that could cause hazards
- Review and revise activities from prior phase

Implementation

- Implement software RCM designs
- Use techniques like FMEA and FTA to uncover new failure modes that could lead to hazards
- Design and implement new RCMs for newly uncovered hazards or causes
- Review and revise activities from prior phase

Test

- Fully test all software implemented RCMs
- Test the effectiveness of non-sofrtware RCMs for software initiated hazards
- Use past experience to “error guess” failure modes that could be hazardous
- Review and revise activities from prior phase

Release & Maintenance

- Clearly document the configuration that has been “safety validated”
- Monitor field complaints and problems. Use this information to re-evaluate possible fault conditions, misuse, abuse, or misunderstandings that could lead to hazards.
- Fully analyze any changes in maintenance releases for new hazards they may introduce or for any RCMs they may have inadvertently broken

* FMEA = Failure Modes & Effects Analysis FTA = Fault Tree Analysis

ABOUT THE AUTHOR:

David Vogel is the founder and president of Intertech Engineering Associates, Inc.

Dr. Vogel was a participant in a joint AAMI/FDA workgroup to develop a standard for Critical Device Software Validation which was subsequently included in the IEC 62304 Software Lifecycle Standard. He was also a participant on the joint AAMI/FDA workgroup to develop a Technical Information Report (TIR) for Medical Device Software Risk Management. Currently, Dr. Vogel is a member of the AAMI/FDA workgroup developing a TIR on Quality System Software Validation.

A frequent lecturer for workshops and seminars on topics related to medical device development and validation, Dr. Vogel also is the author of numerous publications and holds several patents.

Dr. Vogel received a bachelor's degree in electrical engineering from Massachusetts Institute of Technology. He earned a master's degree in biomedical engineering, a master's degree in electrical and computer engineering, and a doctorate in biomedical engineering from the University of Michigan.

Intertech Service Offerings:

Risk Analysis and Management
Software Design and Development
Electronic Design and Development
Requirements Development and Management
Documentation and Traceability
Verification and Validation
Evaluations, Reviews, Inspections
Planning
Project Management
Compliance Consulting and Training
Manufacturing and Quality System Software Validation

Leverage INTERTECH's expertise to:

Reduce Project Risk
Shorten Time to Market
Cut Development and Test Cost
Assure Quality Products

ABOUT INTERTECH:

Intertech Engineering Associates has been helping medical device manufacturers bring their products to market since 1982. Through a distinct top-down service model, Intertech offers high-level consulting and hands-on engineering. By balancing technical expertise and practical business experience, we support clients through all phases of product development. While we do make your job easier, Intertech exists not to replace but to partner with clients to help balance the concerns of quality, time and cost.

With considerable experience in FDA regulatory compliance, our time-tested development process can anticipate and solve problems inexpensively on the planning board rather than through costly solutions later in the development, test, or post-deployment phases. By using deliberate processes, Intertech ensures an improvement in quality and can build client expertise.

Call us today for more information or a free consultation at 781.801.1100

INTERTECH Engineering Associates, Inc.
100 Lowder Brook Drive Suite 2500 Westwood, MA 02090 USA

www.inea.com - info@inea.com - Tel: (781) 801-1100 - Fax: (781) 801-1108