Trustworthy Medical Device Software

Kevin Fu
Assistant Professor
University of Massachusetts Amherst
Department of Computer Science
kevinfu@cs.umass.edu

April 11, 2011

Abstract

This report summarizes what the computing research community knows about the role of trustworthy software for safety and effectiveness of medical devices. Research shows that problems in medical device software result largely from a failure to apply well-known systems engineering techniques, especially during specification of requirements and analysis of human factors. Recommendations to increase the trustworthiness of medical device software include (1) regulatory policies that specify outcome measures rather than technology, (2) collection of statistics on the role of software in medical devices, (3) establishment of open-research platforms for innovation, (4) clearer roles and responsibility for the shared burden of software, (5) clarification of the meaning of substantial equivalence for software, and (6) an increase in FDA's access to outside experts in software. This report draws upon material from research in software engineering and trustworthy computing, public FDA data, and accident reports to provide a high-level understanding of the issues surrounding the risks and benefits of medical device software.

1 Introduction

Software plays a significant and increasing role in the critical functions of medical devices. From 2002–2010, software-based medical devices resulted in over 537 recalls affecting more than 1,527,311 devices[1] [49]. From 1999–2005, the number of recalls affecting devices containing software more than doubled from 118 to 273 [2]. During this period, 11.3% of all recalls were attributable to software failures. This recall rate is nearly double compared to the period of 1983–1997 where only 6% of recalls were attributable to computer software [54]. For pacemakers and implantable cardioverter defibrillators, the number of devices recalled due to software abnormalities more than doubled from 1991–2000 [34]. In 2006, Faris noted the milestone that over half of the medical devices on the U.S. market now involve software [11].

Yet, despite the lessons learned by tragic accidents such as the radiation injuries and deaths caused by the Therac-25 linear accelerator over twenty years ago [29], medical devices that depend on software continue to injure or kill patients in preventable ways. Problems in medical

[1]The total number of devices was computed by summing up the number of devices listed for each of the 537 FDA recalls attributed to software. A device subject to multiple recalls would be counted once per recall.

Sidebar 1: The many definitions of trustworthiness.

One definition of *trustworthy software* is "software that is dependable (including but not limited to reliability, safety, security, availability, and maintainability) and customer-responsive. It can fulfill customer trust and meet the customer's stated, unstated, and even unanticipated needs [23]." Another definition emphasizes the multi-dimensional, system-oriented nature that trustworthiness of a system implies that it is worthy of being trusted to satisfy its specified requirements (e.g., safety, effectiveness, dependability, reliability, security, privacy) with some [quantifiable] measures of assurance [37]. The National Science Foundation associates trustworthiness with properties of security, reliability, privacy, and usability—arguing that these "properties will lead to the levels of availability, dependability, confidentiality and manageability that our systems, software and services must achieve in order to overcome the lack of trust people currently feel about computing and what computing enables [40]."

device software result largely from a failure to apply well-known systems engineering techniques, especially during specification of requirements and analysis of human factors.

> "The ability of software to implement complex functionality that cannot be implemented at reasonable cost in hardware makes new kinds of medical devices possible.... [10]"

Software can help and hurt. Software can significantly affect patient safety in both positive and negative ways. Software helps to automatically detect dangerous glucose levels that could be fatal for a person using an insulin pump to treat diabetes. Medical linear accelerators use software to more precisely target the radiation dose. Remote monitoring of implanted devices may help to more quickly discover malfunctions and may lead to longer survival of patients [26]. However, medical device software contributes to the injury or death of patients. Problems ranging from poor user interfaces to overconfidence in software have led to accidents such as fatally incorrect dosages on infusion pumps [12, 16, 35] and radiation therapy [29, 4]. A common trait for adverse events in medical device software is that the problems are often set in place before any implementation begins (Table 1).

Medical devices ought to be trustworthy. In the context of software, trustworthiness is inextricably linked with safety and effectiveness. There are several definitions of *trustworthy software* (Sidebar 1) that vary by the specific contributions and terminology of various research subdisciplines. However, the fundamental idea is that software trustworthiness is a *system* property measuring how well a software system meets *requirements* such that *stakeholders* will *trust* in the operation of the system. The requirements include overlapping and sometimes competing notions of safety, effectiveness, usability, dependability, reliability, security, privacy, availability, and maintainability.

Failure to meaningfully specify requirements, complacency, and lack of care for human factors can erode trustworthiness. The lack of trustworthy medical device software leads to shortfalls in properties such as safety, effectiveness, usability, dependability, reliability, security, and privacy. Good systems engineering [46] and the adoption of modern software engineering techniques can mitigate many of the risks of medical device software. Such techniques include a technical and

Eng. Stage	Adverse Event	Contributing Factor
Requirements Specification	Linear accelerator: Patients died from massive overdoses of radiation.	An FDA memo regarding the Corrective Action Plan (CAP) notes that, "Unfortunately, the AECL response also seems to point out an apparent lack of documentation on software specifications and a software test plan [30, p. 539]."
Design	Pacemakers/Implantable defibrillators: Implant can be wirelessly tricked into inducing a fatal heart rhythm [21].	Security and privacy need to be part of the early design process.
Human Factors	Infusion pump: Patients injured or killed by drug overdoses.	Software that did not prevent key bounce misinterpreted key presses of 20 mL as 200 mL [17].
Implementation	Infusion pump: Underdosed patient experienced increased intracranial pressure followed by brain death.	Buffer overflow (programming error) shut down pump [14].
Testing	Ambulance dispatch: Lost emergency calls.	An earlier system for the London Ambulance Service failed two major tests and was scuttled [20]. Ambulance workers later accused the computer system of losing calls and that "the number of deaths in north London became so acute that the computer system was withdrawn [53]." The ambulance company attributed the problems to "teething troubles" with a new computer system [53].
Maintenance	Health Information Technology (HIT) devices: Computers systems globally rendered unavailable.	An anti-virus update misclassified a core Windows operating system component as malware and quarantined the file, causing a continuous reboot cycle for any system that accepted the software update [32]. Numerous hospitals were affected. At Upstate University Hospital in New York, 2,500 of the 6,000 computers were affected [52]. In Rhode Island, a third of the hospitals were forced "to postpone elective surgeries and stop treating patients without traumas in emergency rooms [50]."

Table 1: *Examples of adverse events where medical device software played a significant role.*

managerial mindset that focuses on "design and development of the overall system [30, p. 140]" as opposed to focusing on optimization of components; meaningful specification of requirements such as intent specifications [31]; application of systems safety [30], and static analysis [39].

Although it is possible to create trustworthy medical device software under somewhat artificial constraints to achieve safety and effectiveness without satisfying other properties, in practice it is difficult to find environments where the properties are not linked. A medical device that works effectively in isolation may lose the effectiveness property if another component engineered separately joins the system—causing unanticipated interactions. For example, a computer virus caused 300 patients to be turned away from radiation therapy because of shortfalls in security [38]. A security component can also reduce effectiveness if not designed in the context of system. For instance, a mammography imaging system may become ineffective if an automatic update of an anti-virus program designed to increase security causes the underlying operating system to instead fail [32].

Innovations that combine computer technology with medical devices could greatly improve the quality of health care [28, 39], but the same life-saving technology could reduce safety because of the challenges of creating trustworthy medical device software. For instance, an implantable medical device with no physical means to wirelessly communicate over long distances may work safely and effectively for years. However, adding remote monitoring of telemetry to the device introduces an interface that fundamentally changes the properties of the overall system. The new system must require not only that any component designed to interact with the device is trustworthy, but also that any component *capable of communicating with the device* is trustworthy.

2 Medical Devices, But with Software: What's the Difference?

Patients benefit from software-based medical devices because "computers provide a level of power, speed, and control not otherwise possible [30]." Without computer software, it would not be feasible to innovate a closed-loop, glucose-sensing insulin pump; a remotely monitored, implantable cardiac defibrillator; or a linear accelerator that calculates the radiation dose based on a patient's tissue density in each cross-section. However, the methodology used in practice to mitigate risks inherent to software have not kept pace with the deployment of software-based medical devices. For example, using techniques that work well to assure the safety and effectiveness of hardware or mechanical components will not mitigate the risks introduced by software. The following points use the writing of Pfleeger et al. [45] with permission. There are several reasons why software requires a different set of tools to assure safety and effectiveness.

- The discrete (as opposed to continuous) nature of software [43].

  Software is sensitive to small errors. Most engineered systems have large tolerances for error. For example, a 1 inch nail manufactured to be 1.0001 inches or 0.9999 inches can still be useful. Manufacturing is a continuous process, and small errors lead to results essentially the same as the exact, desired result. However, consider a slight error in entering a bolus dosage on an infusion pump. A single key press error in selecting hours versus minutes could result in a bolus drip at 60 times the desired rate of drug delivery [13]. With some exceptions, small changes in continuous systems lead to small effects; small changes to discrete systems lead to large and often disastrous effects. The discrete nature of software also leads to limited ability to interpolate between test results. A system that correctly provides a radiation dose of 20 centigray (cGy) and 40 cGy does not in its own allow interpolation that it would work correctly for 32 cGy. There is also seldom no direct equivalent to "over-engineering" safety margins for software systems in comparison to physical systems.

- The immaturity of software combined with rapid change.

  We keep running at an ever-faster pace to develop or use increasingly complex software systems that we do not fully understand, and we place such software in systems that are more and more critical. For example, a NITRD report from the High-Confidence Medical-Device Software and Systems (HCMDSS) Workshop [39] notes that:

  > "Many medical devices are, essentially, embedded systems. As such, software is often a fundamental, albeit not always obvious, part of a devices functionality. ...devices and systems are becoming increasingly complicated and interconnected. We may already have reached the point where testing as the primary means to gain confidence in a system is impractical or ineffective."

  The recent reporting of several radiation deaths stemming from medical linear accelerators [3] further highlights how complexity outpaces the maturity of present-day practices for creating trustworthy medical device software:

  > "'When it exceeds certain levels of complexity, there is not enough time and not enough resources to check the behavior of a complicated device to every possible, conceivable kind of input,' said Dr. Williamson, the medical physicist from Virginia."
  > ...
  > "But the technology introduces its own risks: it has created new avenues for error in software and operation, and those mistakes can be more difficult to detect. As a result, a single error that becomes embedded in a treatment plan can be repeated in multiple radiation sessions [3]."

Despite these challenges, software has improved the effectiveness of critical systems in contexts such as avionics. Modern airplanes would be difficult to fly without the assistance of software, but airplanes have also introduced safety risks of software by using fly-by-wire (electronic) controls instead of pneumatics. However, there is a substantial belief among software engineers that the medical device community (unlike the avionics community) does not take full advantage of well-known techniques for engineering software for critical systems. Many software engineers feel that that well-known technology not only lags, but is often ignored by medical device manufacturers. The safety culture of the avionics community does not appear to have a universal appreciation in the medical device community.

3 Techniques to Create Trustworthy Medical Device Software

While the role of software in medical devices continues to increase in significance, deployment lags for well-known techniques that can mitigate many of the risks introduced by software. The following discussion draws from several technical documents on software engineering for critical systems.

The reader is strongly encouraged to read the full text of reports from NITRD on *High-Confidence Medical Devices* [39] and the National Academy on *Software for Dependable Systems* [10]. Highly recommended reading on software engineering for critical systems include *Safeware: System Safety and Computers* [30] and *Solid Software* [45] as well as evidence-based certification strategies such as the British Ministry of Defence Standard 00-56 [41].

3.1 Adopt Modern Software Engineering Techniques

Medical device software lags in the adoption of modern software engineering techniques ranging from requirements specification to verification techniques. Fortunately, mature technology is already available to address common problems in medical device software, and that technology has been successful in other safety-critical industries such as avionics.

Programming languages that do not support software fault detection as comprehensively as possible should be avoided in medical device software. The C programming language, for example, has a very weak type system, and so the considerable benefits of strong type checking are lost. By contrast, the Ada programming language provides extensive support for software fault detection. Similarly, mechanical review of software using a technique known as static analysis is a mature technology that can identify possible faults quickly and efficiently. Static analysis supports the overall goal of developing trustworthy software and should be employed to the extent possible. Type checking and static analysis are two mature methods that guide software engineers toward safer and more effective medical device software by reducing or eliminating common sources of software errors.

Some programming systems permit a specification of software to be embedded into the software itself so that compliance of the code with the specification can be checked mechanically. A commercial system that provides this capability along with commercial support of both the language itself and the associated static analysis tools is SPARK Ada. Techniques such as these should be employed whenever possible to enable more effective testing and analysis of software.

A software specification is a statement of what the software has to do. Stating precisely what software has to do has proven extremely difficult, and specification is known to be a major source of software faults. Research over many years has yielded formal languages, i.e., languages with semantics defined in mathematics, that can help to avoid specification errors. Formal specification has been shown to be effective, and formal specifications for medical devices should be employed whenever possible.

3.2 Meaningfully Specify Requirements

Safety failures in software tend to stem from flaws during specification of requirements [30, Ch. 15]. The first example from Table 1 represents a failure of requirements specification in a 1980s linear accelerator that killed a number of patients, and some believe that the lack of meaningful systems-level specification of requirements contributed to the deaths in the recent radiation overdoses from a modern linear accelerator [3].

In critical systems, meaningful specification of requirements is crucial to properly anchor testing and analysis. Shortfalls in specification of requirements will lead to a false sense of safety and effectiveness during subsequent design, implementation, testing, etc. An example of meaningful specification of a requirement might be “stop delivery if dose exceeds patient’s prescription” or “patient’s received level of radiation must match level of radiation specified by operator.” Such specification of requirements go beyond purely functional descriptions such as “pressing start button begins primary infusion” or “delivered level of radiation adjusts to tissue density” that do not meaningfully capture the end-to-end system properties of a medical device.

Leading software engineers believe that many medical device manufacturers have an opportunity to significantly improve specification of requirements. In comparing medical devices to avionics systems, researchers wrote in the NITRD *High-Confidence Medical Devices: Cyber-Physical Systems for 21st century Health Care* report [39, p.59] that,

> “Perhaps the most striking [difference] is the almost complete lack of regard, in the medical-device software domain, for the specification of requirements.”

A National Academy NRC report [10] similarly noted that,

> “At least in comparison with other domains (such as medical devices), avionics software appears to have fared well inasmuch as major losses of life and severe injuries have been avoided.”

The NITRD report emphasizes that business models and incentives in the medical device sector lead to highly proprietary technologies that have two detrimental side effects: (1) companies are less likely to perceive value from specification of requirements, and (2) academic researchers have a much harder time participating in the innovation of medical device technology.

The NRC report recommended a direct path to dependable software [22] for critical systems such as found in medical devices. Under this philosophy, system designers focus on providing direct evidence to support claims about software dependability. The approach contrasts with prescriptive standards[2] that may otherwise dictate the specific claims. System designers are given flexibility to innovate by selecting the claims deemed necessary for the specific application at hand. Designers are forced to think carefully about proving the claims, but a difficulty remains in that the results are only as meaningful as the chosen claims.

3.3 Apply a System Engineering Approach

Software adds such complexity to the design of medical devices that the device must be treated as a system rather than an isolated component. The behavior of medical device software depends on its context within a system. Whereas biocompatibility of material may lend itself to conventional testing [27, Ch. 11], the complexity of software requires a systems engineering approach [46]. At a recent workshop on infusion pumps, it was pointed out that the 510(k) process is mostly a checklist, but this checklist approach provides less assurance as devices increase in complex system behavior [9]. Shuren provides an example of software-based medical devices that may operate safely and effectively in isolation, but not when integrated as a system [47]:

> “Images produced by a CT scanner from one vendor were presented as a mirror image by another vendor’s picture archiving and communication system (PACS) web software. The PACS software vendor stipulates that something in the interface between the two products causes some images to be randomly ’flipped’ when displayed.”

[2]IEC 62304 provides standards for development of medical device software, but not for the requirements themselves where many software flaws tend to begin [9].

The NITRD report from the High-Confidence Medical-Device Software and Systems (HCMDSS) Workshop [39] notes that:

> "Integrating technology into the clinical environment—which includes practitioners, workflows, and specific devices—often lacks a holistic, systems perspective. Many medical devices are designed, developed, and marketed largely as individual systems or gadgets. Device integration, interoperability, and safety features are not considered during development, acquisition, or deployment."

The rapid push toward device interoperability, wireless communication, and Internet connectivity will likely improve the effectiveness of care, but will also reinforce the notion of medical device software as systems rather than isolated devices. Because medical devices are no longer isolated devices, an effective strategy for increasing trustworthiness is to follow good system engineering methodology.

Evaluation of medical device software should require independent, third-party review by experts who are not connected with the manufacturer. Third party evaluation in combination with good systems engineering can mitigate many of the system-level risks of medical device software.

3.4 Mitigate Risks Due to Human Factors

Poor understanding of human factors can lead to the design of medical device software that reinforces risky behavior, which can result in injury or death. For instance, a software application card used in an implantable drug pump was recalled because of a user interface where the hours and minutes fields for a bolus rate were ambiguously labeled on the computer screen [12]. A patient with an implantable drug pump died from an overdose because the health care professional set the bolus interval to 20 minutes rather than 20 hours [13]. Thus, the drug was administered at 60 times the desired rate. The patient passed out while driving, experienced a motor vehicle accident, and later died after the family removed life support.

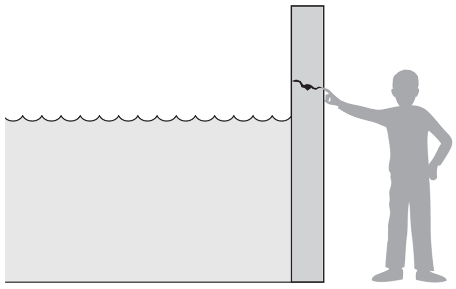
Unmitigated risks of human factors also contributed to the recent radiation overdoses of patients treated by linear accelerators. One report from the New York Times [5] quotes Dr. James Thrall, professor of radiology at Harvard Medical School and chairman of the American College of Radiology, saying, "There is nothing on the machine that tells the technologist that they've dialed in a badly incorrect radiation exposure."

Medical device software must accommodate inevitable human errors without affecting patient safety. Moreover, the specification of requirements should take into account all the key stakeholders. For instance, it is believed that some infusion pump manufacturers specify requirements based mostly on interactions with physicians rather than the primary operators of the pump: nurses. When nurses become disoriented and frustrated using infusion pumps, operational problems can result. Inadequate attention to human factors during specification of requirements will promote hazardous situations.

3.5 Mitigate Low-Probability, High-Consequence Risks

Manufacturers, health care professionals, and users often put too much confidence in medical device software. *It can't happen here. There are no reported problems.* Such statements have only a shallow basis in fact, but lead to a false sense of security. The manufacturer of the Therac-25

Sidebar 2: Threats, Controls, and Vulnerabilities from Pfleeger and Pfleeger [44].



A threat to a computing system is a set of circumstances that has the potential to cause loss or harm. To see the difference between a threat and a vulnerability, consider the illustration. Here, a wall is holding water back. The water to the left of the wall is a threat to the man on the right of the wall: The water could rise, overflowing onto the man, or it could stay beneath the height of the wall, causing the wall to collapse. So the threat of harm is the potential for the man to get wet, get hurt, or be drowned. For now, the wall is intact, so the threat to the man is unrealized. However, we can see a small crack in the wall—a vulnerability that threatens the man's security. If the water rises to or beyond the level of the crack, it will exploit the vulnerability and harm the man.

linear accelerator, which killed and injured a number of patients with radiation overdoses, initially responded to complaints from treatment facilities that, "the machine could not possibly over treat a patient and that no similar complaints were submitted to them [29, 30, 11]." It is very difficult to reproduce problems in software—often leading to denial rather than discovery of root causes. This difficulty derives in part from the complexity of a device's system-of-systems architecture and from the embedded nature of the system (See Section 2 for further discussion).

Security and privacy fall into the category of low-probability, high-consequence risks that could lead to widespread problems with little or no warning. Problems range from downtime to intentional harm to patients. Because devices can easily connect with physically insecure infrastructure such as the Internet and because software vulnerabilities (Sidebar 2) are often discovered with little or no warning before threats exploit the vulnerability [48], security and privacy outcome measures should play a central role for all major aspects of software development of medical device software (specification, design, human factors, implementation, testing, and maintenance).

> "Patients who receive treatment from a potentially lethal medical device should have access to information about its evaluation just as they have access to information about the side effects and risks of medications."— Report on *Software for Dependable Systems: Sufficient Evidence?* by the National Research Council of the National Academies [10, p. 91]

Specification of requirements should address low-probability, high-consequence risks. If a high-consequence risk proves too difficult or costly to mitigate, health care professionals deserve to know about the risks no matter how small.

Innovations in wireless communication and computer networking have led to great improvements in patient care ranging from remote, mobile monitoring of patients (e.g., at-home monitors

Sidebar 3: Medical devices are susceptible to malware.

Medical devices are no more immune to malware (e.g., viruses, botnets, keystroke loggers) than any other computer. Computer viruses can delete files, change values, expose data, and spread to other devices. A computer virus does not distinguish between a home computer and a hospital computer. Yet in the health care setting, the consequences of malicious software could lead to less effective care (e.g., corrupted electronic medical records that necessitate re-testing) and diminished safety (e.g., overdoses from infusion pumps, radiation therapy, or implantable medical devices).

For these reasons, vendors may advise health care providers to install anti-virus software with automated Internet-based updates. However, these products introduce risks that can themselves reduce the trustworthiness of the medical device software. When McAfee released an automated update of its virus definition files, the anti-virus product incorrectly flagged a critical piece of Windows software as malicious—and quarantined the software [32]. This disruption of a critical file caused a number of hospitals to suffer downtime. Medical systems were rendered unavailable.

for cardiac arrhythmias or diabetes) to reduced risks of infection as a result of removing computer equipment from the sterile zone of an operating room (e.g., wireless wands for pacemaker reprogramming). However, the increased interconnectedness of medical devices leads to security and privacy risks for both medical devices in the hospital and in the home [25, 19, 33]. For instance, there is no public record of a specification that requires a home monitoring device *not to be physically capable* of reprogramming an implanted cardiac device. Thus, a malicious piece of software could change the behavior of a home monitor to quietly disable therapies or even induce a fatal heart rhythm—without violating the public specification.

4 Policy Recommendations for Trustworthy Medical Device Software

Regulatory and economic policies should promote innovation while incentivizing trustworthiness in a least burdensome manner. Although one study of medical device recalls concludes that the economic impact of poor quality does not in general have severe financial penalties on the affected company [51], policy recommendations below focus on technical and managerial issues rather than financial penalties or incentives.

4.1 Specify Outcome Measures, Not Technology

The safety and effectiveness of software-based medical devices could be better regulated in terms of outcome measures rather than in terms of specific technologies[3]. The regulatory infrastructure should aim at making industry meet meaningful goals and show proof of achieving such goals.

The push toward prescriptive standards leads to an oversimplification in that the trustworthiness of a device depends on context. For example, one FDA notice advises to "update your operating system and medical device software [15]." However, software updates themselves can carry risks that should be either accepted or mitigated depending on the situation specific to each

[3]In Europe, the legal definition of a medical device explicitly mentions software [18]. In the United States, the legal definition of a medical device is less specific.

medical device. On a desktop computer used to update a portable automated external defibrillator (AED), it might be reasonable to routinely update the operating system even if there is a risk that the update may fail in a manner that makes the desktop machine inoperable. However, updating the operating system on the defibrillator itself carries a risk that failure could render the AED inoperable. A hospital that updates all its devices simultaneously is vulnerable to system-wide inability to provide care.

Rather than prescribe specific technologies, regulatory policies should incentivize manufacturers to specify meaningful outcome measures in the context of the given device and be required to prove such claims. Lessons from evidence based medicine [42] could assist in creating outcome measures for trustworthy medical device software.

4.2 Collect Better Statistics on the Role of Software in Medical Devices

Many questions about the trustworthiness of medical device software are difficult to answer because of lack of data and inadequate record keeping. Questions include:

- To what degree are critical device functions being performed by software (vs. hardware)? Is the amount increasing? Decreasing?
- What effect does software have on reliability? Availability? Maintainability? Ease of use?
- How do these software characteristics compare with similar implementations in hardware? Does the software make the device safer or more effective?
- What does data from the predicate device reveal about the new device? Does predicate data save time in specification of the new device? Does predicate data save time in testing of the new device?

Many record keeping tools are already in place (e.g., the MAUDE adverse events database and the recalls database at FDA). However, these tools are severely underutilized. Databases suffer from severe underreporting. For example, in the same time period there are only 372 adverse event reports in MAUDE that cite “computer software issues” despite there being well over 500 entries in the recall database that cite software as a reason for the recall. At the VA, “over 122 medical devices have been compromised by malware over the last 14 months” according to House testimony [1]. But there are no records in MAUDE citing a “computer system security issue[4]”.

Scott Bolte of GE Healthcare emphasizes that for security problems, formal reporting is especially lacking [6]:

> “Although there is a lot of anecdotal evidence that malicious software has compromised medical devices, there is a notable lack of formal evidence. So without this formal reporting, the FDA is limited in its ability to act or intervene. Reporting is something providers and arguably the manufacturers themselves can and should start doing immediately.”

Policies should encourage better reporting of adverse events and recalls. Otherwise it will only be possible to point out anecdotal failures rather than confidently point out trends for successful products that epitomize innovation of trustworthy medical device software.

[4]On July 12, 2010, it was determined that two bogus MAUDE records classified under a “computer system security issue” were not actually related to computer security.

4.3 Enable Open Research in Software-based Medical Devices

The highly proprietary nature of the medical device software industry makes it difficult for innovators to build upon techniques of properly built systems. Some information may become public after an accident, but this information teaches about failure rather than success. More open access to success stories of engineering medical device software would lead to innovation of safer and more effective devices. The NITRD report [39] explains that:

> "Today we have open-research platforms that provide highly effective support for the widespread dissemination of new technologies and even the development of classified applications. The platforms also provide test beds for collaborations involving both researchers and practitioners. One spectacular example is the Berkeley Motes system with the TinyOS operating system."
>
> "The medical-device community could benefit from the existence of such open-research platforms. They would enable academic researchers to become engaged in directly relevant problems while preserving the need for proprietary development by the industry. (TinyOS facilitates academic input even on government-classified technology, which is an example of what is possible.)"
>
> "An open research community needs to be established comprising academics and medical device manufacturers to create strategies for the development of end-to-end, principled, engineering-based design and development tools."

4.4 Clearly Specify Roles and Responsibility

In complex systems of systems that rely on software, it is difficult to pinpoint a single party responsible for ensuring trustworthiness of software because the property is of the system of systems rather than of individual components. A modern linear accelerator is an example of a complex system of systems because commercial off-the-shelf (COTS) software such as Windows may serve as the underlying operating system for a separately engineered software application for planning and calculation of dose distribution. An embedded software system then uses the treatment plan to controls mechanical components that deliver radiation therapy to a patient. When different entities separately manage software components in complex systems of systems, system-level properties such as safety are more difficult to ensure because no single entity is responsible for overall safety.

The FDA notes that a key challenge is a shared responsibility for failures in software [15]. If the user updates the software on a medical device, then is the manufacturer truly at fault? If a medical device relies on third party software such as operating systems, then who is responsible for maintaining the software?

Technology alone is unlikely to mitigate risks that stem from system-level interactions of complex software designed by different organizations with different agendas and outcome measures. The problem is likely intractable without a single authority responsible for the trustworthiness of interfaces between interacting systems. The interface between medical device application software COTS software is a common battleground for disclaimers of responsibility (See Sidebar 4).

Leveson [30, Sec. 4.2.1] points out that diffusion of responsibility and authority is an ineffective organizational structure that can have disastrous effects when safety is involved. The British

Sidebar 4: Take Service Pack 3 and see me in the morning.

Medical devices can outlast the underlying operating system software. Many medical devices rely on commercial off-the-shelf (COTS) software, but COTS software tends to have a shorter lifetime for expected use than a typical medical device. For instance, Microsoft mainstream support for Windows XP lasted for less than 8 years (December 2001–April 2009) [36], whereas an MR scanner may have an operational life of ten to twenty years [6].

It is not uncommon for a newly announced medical device to rely on operating system no longer supported by its manufacturer. Microsoft ended support for security patches for Windows XP Service Pack 2 and advises vendors to upgrade products to Service Pack 3. But hospitals often receive conflicting advice on whether to update software. House testimony [24] mentions that,

> "As a sobering side-note, over the last three weeks, in collaboration with a researcher from Georgia Tech in Atlanta who is involved with the Conficker Working group, I have identified at least 300 critical medical devices from a single manufacturer that have been infected with Conficker. These devices are used in large hospitals, and allow doctors to view and manipulate high-intensity scans (MRI, CT Scans etc), and are often found in or near ICU facilities, connected to local area networks that include other critical medical devices.
>
> Worse, after we notified the manufacturer and identified and contacted the hospitals involved, both in the US and abroad, we were told that because of regulatory requirements, 90 days notice was required before these systems could be modified to remove the infections and vulnerabilities."

Users of medical PACS (picture archiving and communication system) struggle to meet conflicting requirements: medical device manufacturers who require health care facilities to use old, insecure operating systems and FDA guidelines that advise keeping operating systems up-to-date with security patches. One anonymous posting on a technical support web site [8] reads:

> "I am setting up a medical imaging facility and I am trying to do the same thing as well. The PACS system we are integrating with is only compatible with SP2. I order 6 new Dell workstations and they came preloaded with SP3. There are 'actual versions of programs out there that require SP2. For instance, the $250,000 Kodak suite I am installing. Plus a $30,000/yr service contract. This holds true for the majority of the hospitals which have PACS systems.
>
> However, if what you are saying is true then I found something useful within your post. You stated 'if you installed XP with integrated sp3, it is not possible to downgrade sp3 to sp2,' is this true? Do you have any supporting documentation as this would be very helpful so that I can provide Dell with a reason why I need to order downgraded XP discs."

The plaintive quality of this call for help provides insight into how helpless some users feel because of the diffusion of responsibility for maintaining COTS software contained within medical devices.

Ministry of Defence [41] provides a good example of clear roles and responsibilities for safety management of military systems. The ideas apply broadly to critical systems and may work well for medical systems.

4.5 Clarify the Meaning of Substantial Equivalence for Software

In the context of the 510(k) pre-market notification process, demonstration of "substantial equivalence" to a previously approved "predicate" medical device allows a manufacturer to more quickly seek approval to market a medical device.

Imagine if the predicate device has a function implemented in hardware, and the manufacturer claims that the new version is substantially equivalent because the only difference is that the new version is implemented in software. Because hardware and software exhibit significantly different behavior (see Section 2), it is important that the design, implementation, testing, human factors analysis, and maintenance of the new device mitigate the risks inherent to software. However, this difference casts doubt on substantial equivalence because of the different technological characteristics that raise different risks to safety and effectiveness. Furthermore, when does a software-related flaw in a recalled predicate device imply that the same flaw exists in the new device?

As was noted at the Institute of Medicine Meeting #2 on Public Health Effectiveness of the FDA 510(k) Clearance Process in June 2010, there is doubt on whether hardware can act as a predicate for functions implemented in software:

> "One of the interesting classes is radiation equipment...even the software, which I wonder where they got the first predicate for software."
>
> –Dr. David Feigel
> Former Director
> FDA Center for Devices and Radiological Health (CDRH)

The interpretation of substantial equivalence needs clarification for software-based medical devices.

4.6 Increase FDA Access to Outside Experts in Software Engineering

The FDA should increase its ability to maintain safety and effectiveness of medical devices by developing a steady pipeline of human resources with expertise in software engineering for critical systems.

Various offices within FDA's Center for Devices and Radiological Health employ a small number of software experts. FDA also has a number of successful fellowship programs including the Commissioner's Fellowship Program, the Medical Device Fellowship Program, and the Device Evaluation Intern Program to attract students and experienced experts from medical and scientific communities. However, software experts are notably underrepresented in these programs. The Web page for the Medical Device Fellowship Program[5] targets health professionals, and other

[5] http://www.fda.gov/AboutFDA/WorkingatFDA/FellowshipInternshipGraduateFacultyPrograms/MedicalDeviceFellowshipProgramCDRH/default.htm

Sidebar 5: Substantial Equivalence: Paper or plastic?

An interesting thought experiment is to ask how the trustworthiness of electronic health records differ from traditional paper records. FDA generally does not consider a paper medical record as a medical device. However, FDA may consider an electronic health record as a medical device. Adding automated algorithms to prioritize display of data from an electronic medical record would shift the system toward regulation as a medical device.

Paper records are subject to threats such as fire, flood, misplacement, incorrect entry, and theft. Paper records are cumbersome to backup and require large storage rooms. But electronic records introduce risks qualitatively different from paper records. Making changes to a paper record tends to leave behind physical evidence that is auditable, but making electronic records auditable requires intentional design. A single coding error or errant key press could lead to destruction of an entire collection of electronic records—especially for encrypted data. The speed of technology can make electronic record keeping easier, but can encourage bad habits that to lead to difficult to detect mistakes. For instance, a computer display that clears the screen following the completion of an operation makes it difficult to trace back a sequence of changes.

Overconfidence in software for electronic medical records could lead to financially-motivated decisions to discontinue paper-based backup systems. One full-scale failure of a clinical computing system at the Beth Israel Deaconess Medical Center lasted four days—forcing the hospital to revert to manual processing of paper records [25]. While paper-based backup procedures allowed care to continue, few of the medical interns had any experience with writing orders on paper. When health care professionals struggle with technology, patients are at risk.

Heated debates about paper versus electronic recording appears in other contexts such as voting. A National Academies report [7] provides context for the electronic voting debate with arguments applicable to the safety and effectiveness of electronic medical records.

existing programs primarily target biomedical engineers rather than software engineers. Of the fifty Commissioner's Fellows selected in 2009, none had formal training in computer science[6]. In 2008, one of the fifty fellows had a computer science degree, but did not work in the Center for Devices and Radiological Health. A former FDA manager indicated that software experts rarely participate in these fellowship programs. Another person familiar with FDA processes noted that seldom does an FDA inspector assigned to review a 510(k) application have experience in software engineering—even though the majority of medical devices today rely on software.

The FDA should expand its access to outside experts for medical device software by creating fellowship programs that target software engineers. For instance, FDA could more aggressively recruit students and faculty from computer science and engineering—especially individuals with advanced training in software engineering topics such as system and software safety, dependable computing, formal methods, formal verification, and trustworthy computing.

[6] `http://www.fda.gov/downloads/AboutFDA/WorkingatFDA/FellowshipInternshipGraduateFacultyPrograms/CommissionersFellowshipProgram/UCM216921.pdf`

5 Summary

The lack of trustworthy medical device software leads to shortfalls in safety and effectiveness, which are inextricably linked with properties such as usability, dependability, reliability, security, privacy, availability, and maintainability. Many risks of medical device software could be mitigated by applying well-known systems engineering techniques, especially during specification of requirements and analysis of human factors. Today, the frequency of news reports on tragic, preventable accidents involving software-based medical devices falls somewhere between that of automobile accidents and airplane accidents. Event reporting on tragic medical device accidents is likely headed toward the frequency of the former given the continued increase in system complexity of medical device software and present-day regulatory policies that do not adequately encourage use of modern software engineering and system engineering practices.

6 Acknowledgments

Several individuals deserve thanks for assistance with this report. Shari Lawrence Pfleeger provided extensive feedback and contributed material on threats and on why software behaves differently from hardware. John Knight provided feedback and contributed material on modern software engineering techniques. Dr. Philip Henneman, Daniel Jackson, Leon Osterweil, and Jerry Saltzer provided feedback on early drafts of this report. Paul L. Jones, Nancy Leveson, John F. Murray Jr., and Gideon Yuval provided further technical input. Quinn Stewart collected data pertaining to FDA medical device recalls. Hong Zhang assisted with formatting of the report. Heather Colvin and Abigail Mitchell at the Institute of Medicine deserve special thanks for organizing a successful series of workshops on the Public Health Effectiveness of the FDA 510(k) Clearance Process.

References

[1] Roger W. Baker. Statement of The Honorable Roger W. Baker. House Committee on Veterans' Affairs, Subcommittee on Oversight and Investigations, Hearing on Assessing Information Security at the U.S. Department of Veterans Affairs, May 2010.

[2] Z Bliznakov, G Mitalas, and N Pallikarakis. Analysis and classification of medical device recalls. *World Congress on Medical Physics and Biomedical Engineering 2006*, 14(6):3782–3785, 2006.

[3] Walt Bogdanich. As technology surges, radiation safeguards lag. The New York Times, `http://www.nytimes.com/2010/01/27/us/27radiation.html`, January 26 2010.

[4] Walt Bogdanich. Radiation offers new cures, and ways to do harm. The New York Times, `http://www.nytimes.com/2010/01/24/health/24radiation.html`, January 23 2010.

[5] Walt Bogdanich and Rebecca R. Ruiz. F.D.A. to increase oversight of medical radiation. The New York Times, `http://www.nytimes.com/2010/02/10/health/policy/10radiation.html`, February 9 2010.

[6] Scott Bolte. Cybersecurity of medical devices. http://www.fda.gov/MedicalDevices/Safety/MedSunMedicalProductSafetyNetwork/ucm127816.htm, April 2005.

[7] Richard Celeste, Dick Thornburgh, and Herbert Lin, editors. *Asking the Right Questions about Electronic Voting*. National Academies Press, 2005. Computer Science and Telecommunications Board (CSTB).

[8] Windows Client Tech Center. Downgrade from SP3 to SP2. http://social.technet.microsoft.com/forums/en-US/itproxpsp/thread/5a63cf6d-f249-4cbe-8cd2-620f301ec6fc/, December 2008.

[9] Richard Chapman. Assurance cases for external infusion pumps. FDA Infusion Pump Workshop, http://www.fda.gov/downloads/MedicalDevices/NewsEvents/WorkshopsConferences/UCM217456.pdf, May 2010.

[10] Lynette I. Millett Daniel Jackson, Martyn Thomas, editor. *Software for Dependable Systems: Sufficient Evidence?* National Academies Press, 2007. Committee on Certifiably Dependable Software Systems, National Research Council.

[11] Thomas H. Faris. *Safe and Sound Software: Creating an Efficient and Effective Quality System for Software Medical Device Organizations*. ASQ Quality Press, 2006.

[12] FDA. Medtronic announces nationwide, voluntary recall of model 8870 software application card. http://www.fda.gov/MedicalDevices/Safety/RecallsCorrectionsRemovals/ListofRecalls/ucm133126.htm, September 2004.

[13] FDA. Neuro n'vision programmer. MAUDE Adverse Event Report #2182207-2004-00681, http://www.accessdata.fda.gov/SCRIPTs/cdrh/cfdocs/cfMAUDE/Detail.cfm?MDRFOI__ID=527622, May 2004.

[14] FDA. Baxter healthcare pte. ltd. colleague 3 cxe volumetric infusion pump 80frn. MAUDE Adverse Event Report #6000001-2007-09468, http://www.accessdata.fda.gov/SCRIPTs/cdrh/cfdocs/cfMAUDE/Detail.cfm?MDRFOI__ID=914443, September 2007.

[15] FDA. Reminder from FDA: Cybersecurity for networked medical devices is a shared responsibility. http://www.fda.gov/MedicalDevices/Safety/AlertsandNotices/ucm189111.htm, November 2009.

[16] FDA. Infusion pump improvement initiative. http://www.fda.gov/medicaldevices/productsandmedicalprocedures/GeneralHospitalDevicesandSupplies/InfusionPumps/ucm205424.htm, April 2010.

[17] Valerie A. Flournoy. Medical device recalls. http://www.fda.gov/downloads/MedicalDevices/NewsEvents/WorkshopsConferences/UCM216992.pdf, May 2010.

[18] Richard C. Fries. *Reliable Design of Medical Devices, Second Edition*. CRC Press, 2005.

[19] Kevin Fu. Inside risks, reducing risks of implantable medical devices. *Communications of the ACM*, 52(6), June 2009.

[20] Dorothy R. Graham. London ambulance service computer system problems. Risks Digest, Volume 13, Issue 38, `http://catless.ncl.ac.uk/Risks/13.38.html`, April 10 1992.

[21] Daniel Halperin, Thomas S. Heydt-Benjamin, Benjamin Ransford, Shane S. Clark, Benessa Defend, Will Morgan, Kevin Fu, Tadayoshi Kohno, and William H. Maisel. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *Proceedings of the 29th Annual IEEE Symposium on Security and Privacy*, May 2008.

[22] Daniel Jackson. A direct path to dependable software. *Communications of the ACM*, 52(4):78–88, 2009.

[23] Bijay J. Jayaswal and Peter C. Patton. *Design for Trustworthy Software*. Prentice Hall, Pearson Education, 2007.

[24] Rodney Joffe. Testimony of Rodney Joffe. House Committee on Energy and Commerce Subcommittee on Communications, Technology and the Internet, `http://energycommerce.house.gov/Press_111/20090501/testimony_joffe.pdf`, May 1 2009.

[25] Peter Kilbridge. Computer crash–lessons from a system failure. *N Engl J Med*, 348(10):881–2, Mar 2003.

[26] Gina Kolata. New tools for helping heart patients. The New York Times, `http://www.nytimes.com/2010/06/22/health/22heart.html`, June 21 2010.

[27] Theodore R. Kucklick, editor. *The Medical Device R&D Handbook*. Taylor & Francis, 2006.

[28] I Lee, G Pappas, R Cleaveland, J Hatcliff, B Krogh, P Lee, H Rubin, and L Sha. High-confidence medical device software and systems. *Computer*, 39(4):33 – 38, 2006.

[29] N Leveson and C Turner. An investigation of the Therac-25 accidents. *IEEE Computer*, 26(7):18 – 41, 1993.

[30] Nancy G. Leveson. *Safeware: System Safety and Computers*. Addison-Wesley, 1995.

[31] Nancy G. Leveson. Intent specifications: An approach to building human-centered specifications. *IEEE Transactions on Software Engineering*, 26:15–35, 2000.

[32] John Leyden. Rogue McAfee update strikes police, hospitals and Intel. `http://www.theregister.co.uk/2010/04/22/mcafee_false_positive_analysis/`, April 22 2010.

[33] William H Maisel and Tadayoshi Kohno. Improving the security and privacy of implantable medical devices. *N Engl J Med*, 362(13):1164–6, Apr 2010.

[34] William H Maisel, William G Stevenson, and Laurence M Epstein. Changing trends in pacemaker and implantable cardioverter defibrillator generator advisories. *Pacing Clin Electrophysiol*, 25(12):1670–8, Dec 2002.

[35] Barry Meier. F.D.A. steps up oversight of infusion pumps. The New York Times, `http://www.nytimes.com/2010/04/24/business/24pump.html`, April 23 2010.

[36] Microsoft. Windows XP history. `http://www.microsoft.com/windows/WinHistoryProGraphic.mspx`, June 2003.

[37] Peter G. Neumann. Risks of untrustworthiness. In *22nd Annual Computer Security Applications Conference (ACSAC)*, 2006.

[38] BBC News. Hospital struck by computer virus. `http://news.bbc.co.uk/2/hi/uk_news/england/merseyside/4174204.stm`, August 22 2005.

[39] NITRD. High-confidence medical devices: Cyber-physical systems for 21st century health care. Networking and Information Technology Research and Development Program, February 2009.

[40] NSF. Trustworthy computing. http://www.nsf.gov/funding/pgm_summ.jsp?pims_id=503326, 2010.

[41] Ministry of Defence. Ministry of defence standard 00-56: Safety management requirements for defence systems, June 2007. Issue 4.

[42] LeighAnne Olsen, Dara Aisner, and J. Michael McGinnis, editors. *The Learning Healthcare System: Workshop Summary (IOM Roundtable on Evidence-Based Medicine)*. National Academies Press, 2007. Institute of Medicine.

[43] David Lorge Parnas. Software aspects of strategic defense systems. *Communications of the ACM*, 28(12):1326–1335, 1985.

[44] Charles P. Pfleeger and Shari Lawrence Pfleeger. *Security in Computing*. Prentice Hall, 4e edition, 2007.

[45] Shari Lawrence Pfleeger, Les Hatton, and Charles Howell. *Solid Software*. Prentice Hall, 2001.

[46] Michael Ryschkewitsch, Dawn Schaible, and Wiley Larson. The art and science of systems engineering. NASA Monograph, `http://spacese.spacegrant.org/uploads/images/Art_and_Sci_of_SE.pdf`, January 2009.

[47] Jeffrey Shuren. Testimony of Jeffrey Shuren, Director of FDA's Center for Devices and Radiological Health. Health Information Technology (HIT) Policy Committee Adoption/Certification Workgroup, February 25 2010.

[48] Stuart Staniford, Vern Paxson, and Nicholas Weaver. How to own the internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium*, pages 149–167, 2002.

[49] Quinn Stewart and Kevin Fu. A second look: Analysis of FDA recalls of software-based medical devices. Manuscript, July 2010.

[50] Peter Svensson. McAfee antivirus program goes berserk, freezes pcs. Associated Press, `http://abcnews.go.com/Technology/wireStory?id=10437730`, April 21 2010.

[51] Sriram Thirumalai and Kingshuk K. Sinha. Product recalls in the medical device industry: An empirical exploration of the sources and financial consequences. Accepted to Management Science, May 2010.

[52] Dave Tobin. University hospital computers plagued by anti-virus glitch. The Post-Standard, `http://www.syracuse.com/news/index.ssf/2010/04/university_hospital_plagued_by.html`, April 21 2010.

[53] Brian Tompsett. Long call wait for London ambulances. Risks Digest, Volume 13, Issue 42, `http://catless.ncl.ac.uk/Risks/13.42.html#subj8`, April 16 1992.

[54] Dolores R. Wallace and D. Richard Kuhn. Failure modes in medical device software: an analysis of 15 years of recall data. *International Journal of Reliability Quality and Safety Engineering*, 8:351–372, 2001.