The Report Committee for Pang Lithisay Vadysirisack
Certifies that this is the approved version of the following report:

Practical Software Testing for an FDA-Regulated Environment

APPROVED BY

SUPERVISING COMMITTEE:

Sarfraz Khurshid, Supervisor

Dewayne E. Perry

Practical Software Testing for an FDA-Regulated Environment

by

Pang Lithisay Vadysirisack, B.S.E.E.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2011

Practical Software Testing for an FDA-Regulated Environment

Pang Lithisay Vadysirisack, M.S.E.

The University of Texas at Austin, 2011

Supervisor: Sarfraz Khurshid

Unlike hardware, software does not degrade over time or frequency use. This is good for software. Also unlike hardware, software can be easily changed. This unique characteristic gives software much of its power, but is also responsible for possible failures in software applications. When software is used within medical devices, software failures may result in bodily injury or death. As a result, regulations have been imposed on the makers of medical devices to ensure their safety, which includes the safety of the devices' software. The U.S. Food and Drug Administration requires establishment of systems and control processes to ensure quality devices. A principal part of the quality assurance effort is testing. This paper explores the unique role of software testing in the design, development, and release of software used for medical devices and applications. It also provides practical, industry-driven guidance on medical device software testing techniques and strategies.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Due to its widespread adoption, software has become an integral part of the modern world. It can be found in many devices we use every day. From computers and smart phones to microwave ovens and automobiles, software provides much of the functionality that millions of people depend on daily.

Due to Moore's Law, hardware has become smaller, cheaper and faster over the years. At the same time, software has evolved to allow users of computing hardware to rapidly create applications for the needs of their own and their customers. In a global market where the winners are the ones who are able to introduce products into the market first, the effective use of software is often considered a vital activity. Software provides a quick, powerful, and cost-effective way to provide the services that customers demand. It is not hard, therefore, to conclude that producers of high tech products have embraced the use of software in their devices and applications.

Unsurprisingly, software has taken root in the healthcare industry. Older generations of mechanical and analog electromechanical devices used in patient diagnosis, monitoring, and treatment have largely been replaced by devices and systems based on information technology [2]. Software can now be found in all sorts of products targeted specifically for the healthcare industry. X-ray computed tomography (CT) machines create three-dimensional images of patients with the use of computer processing [48]. Small implantable cardiac rhythm management (CRM) devices contain embedded software to allow monitoring and correction of slow heart rates [9]. In recent years, the United States government has created incentives and invested billions of dollars to stimulate the widespread adoption of electronic medical record systems in a push to improve care and lower cost [43].

In spite of the fact that the public depends on software to function correctly all the time, software development professionals generally admit to the view that there is simply no way to assure that software is completely free of flaws [23]. This is a disturbing outlook, especially given the possibility of physical injury or even death in an event of medical device failure. One of the main ways to reduce the risk of failure is testing. However, testing, by itself, has its own limitations and, when applied to complex software applications of today, often breaks down in its effectiveness. This paper discusses the role of software testing in producing quality medical devices and applications. In addition, it describes the regulations imposed on the medical device manufacturing industry. Finally, it provides practical, industry-driven guidance for persons involved in the design, development, and release of these products.

This paper is organized as follows: Chapter 2 provides background and motivation. Chapter 3 provides a review of the literature available on software testing for medical devices. Chapter 4 explores the regulatory viewpoint on software testing. Chapter 5 describes the existing landscape of software testing. Chapter 6 provides guidance for the practitioner of medical device software testing. Chapter 7 presents the conclusion.

Chapter 2: Background

A Software Developer's Story

It is Calvin's first month as a software developer at LittleTech, a small software contracting firm specializing in providing turnkey software solutions to many of the industries found in the southwest United States. Calvin, recruited heavily from the in-town university, had passed all his software engineering courses with flying colors and is enthusiastic in delivering positive results in his first real-world projects.

LittleTech soon lands a contract with a local manufacturer of portable music players, whose Research and Development department requires an application to be developed to test its new touch screen design. Due to their less than spectacular market share (most likely due to the overwhelming success of Apple's iPod), the manufacturer requests a junior engineer billing rate to reduce costs. LittleTech assigns Calvin to the task.

Due to his company's relatively small size, Calvin is forced to work mostly independently and learn as he goes. Because of his positive attitude and attention to detail, he is able to single-handedly design, develop, debug, and install the application, which the manufacturer accepts. In order to meet tight schedules (which was also driven by the manufacturer's reluctance to spend), Calvin does not follow a prescribed development process nor implement any quality checks, such as tests and reviews. These oversights most likely contribute to failures in the application; the most significant is its inability to read screen coordinates near its edges, which falsely fails some product designs. Fortunately, these failures do not stop production of any existing products, and the application is largely considered a success given the project's constraints. A few weeks into using the new test application, financial troubles force the discontinuation of

all efforts to include a touch screen into the manufacturer's newest model, shelving Calvin's software indefinitely.

For Calvin's next project, LittleTech secures a project to create a web-based inventory system for lab equipment used within a large semiconductor company. This project piques Calvin's interest since his school senior project involved creating a web-based database search engine. He is also determined to drastically reduce the number of "bugs" and failures, so he makes a point to include software testing activities to the project schedule that he submits to his customer. He subsequently designs the system based on mostly verbal requirements, writes his software, and presents his progress during development milestone meetings as mandated by the semiconductor company's corporate policy. He runs code modules through unit tests, performs integration tests to verify the interfaces between software components, and, in order to meet predefined deadlines, executes an abbreviated set of final system tests. After release of the web application, the customer is impressed by the improvement it provides over its previous method of recording items in spreadsheets distributed among several computers. However, after a few days of use, the customer discovers that the application fails to accept dashes in serial number strings. Calvin comes to find out that this was a verbal requirement and simply forgot to add this to the code. He submits an engineering change order and performs the fix, which is difficult since his code base had not been originally written to accommodate this requirement. To Calvin's dismay, further errors appear in the application, despite the fact that he ran a comprehensive suite of tests. A fatal flaw crashes the software if items are added to a list and then removed completely, causing an invalid arithmetic operation with zero. Calvin comes to the conclusion that an unexpected combination of input values had caused the error, which his testing never took into account.

At this point, Calvin is losing the confidence he had on his first days on the job. He was considered a "superstar" programmer throughout his college career and had been recognized as one of the best at LittleTech. He had been careful in his coding, followed good programming standards, and even performed, based on his knowledge, exhaustive tests. Nonetheless, his software deliverables still exhibited flaws.

Without giving Calvin any further time to sulk, LittleTech finds him a third assignment, this time with a medical device company. He is tasked to build a final shipment test system for the company's handheld programming devices. Now, the margin for error immediately became extremely small. Since Calvin's software will be used in the manufacture of a medical device, its fit and function will be governed by federal regulations. In addition, his software will be used to qualify product for shipment, making it a vital component for the economic health of the company. If his software does not perform as intended at all times, shipments may stop, orders lost, and, at worst, the company shut down due to regulatory noncompliance.

Calvin feels that the odds are stacked against him. Based on his past efforts, what are the chances that he will be able to create a high-quality application in a reasonable time and budget? Will these regulations hurt his chances for success or will they possibly help him build a near flawless system? How will this reliability and safety critical environment change his development strategy in terms of documentation, validation, and testing?

MEDICAL DEVICE REGULATORY BACKGROUND

The regulatory body that is responsible for protecting and promoting public health in the United State is the Food and Drug Administration (FDA). Established in 1862, it is

the oldest consumer protection agency in American history. Its mission statement is stated in its "What We Do" web page [46]:

> FDA is responsible for protecting the public health by assuring the safety, efficacy and security of human and veterinary drugs, biological products, medical devices, our nation's food supply, cosmetics, and products that emit radiation.

Before rising to achieve the far-reaching authority it has today, the landscape of drugs, therapies, and medical devices was one filled with deceit, unethical behavior, and danger. In the early 1900s, labeling and packaging of so-called medicines were either completely ineffective or caused harmful effects such as blindness and even death. One of the most devastating cases involved a widely consumed elixir that was found to have the same properties as antifreeze. With the increasing prevalence of electricity in the late 19th century, several devices appeared on the market that attempted to employ electricity to cure all sorts of ailments, which included back pain, insomnia, depression, sexual dysfunction, and heart disease. Products such as electric belts, electric brushes, and even devices that emitted radiation sought to exploit people's fascination with the new technology [34]. Regrettably, these "quack" devices never alleviated any of the problems that these device inventors had advertised they would.

Several high-profile cases sparked key pieces of legislation over the years. With several worthless and potentially harmful drugs openly available on the market, the first piece of legislation, the 1906 Pure Food and Drugs Act, attempted to curb the interstate buying and selling of adulterated and misbranded food and drugs. The events of the consumption of Elixir Sulfanilamide, which was the cause of more than 100 deaths, led to the landmark Food, Drug, and Cosmetic Act (FD&C) of 1938. This act's primary contribution was the requirement that all drugs undergo an approval process demonstrating their safety and efficacy before release to the public.

This act effectively gave the FDA complete authority to oversee the safety of food and drugs, but was required to be amended several times in response to several high profile incidents. The 1962 Kefauver Harris Amendment, requiring drug advertising to disclose side effects, was introduced in response to the Thalidomide tragedy, where children of the mothers taking this morning sickness pill were born with birth defects. The damage due to the use of the Dalkon Shield contraceptive medical device lead to the 1976 Medical Device Amendments (MDAs) that required pre-market testing and approval, known as Pre-Market Approvals (PMAs), of all medical devices. Finally, the most often referenced incident involving medical device software causing injury and death was the Therac-25 X-Ray episode where, in the mid-1980s, a software flaw led patients to receive a fatal overdose of radiation. The 1990 Safe Medical Device Act (SMDA) gave the FDA more regulatory powers to remove unsafe medical devices from the market. It also revised Chapter 820 of Title 21 of the Code of Federal Regulations (21 CFR 820) to mandate device manufacturers to implement quality controls, such as following Good Manufacturing Practices (GMPs) and maintaining Design History Files (DHFs).

SOFTWARE TESTING BACKGROUND

A test refers to taking measures to check the quality, performance, or reliability of (something), especially before putting it into widespread use or practice [11]. When applying the term in the context of the software field, the definition of a test becomes more complicated. Software testing may mean that it is intended to identify potential malfunctions, sometimes referred to as “bugs”. Software testing may describe validation of intended behavior. Yet other definitions specify that tests mainly reduce risk. Others also claim that software tests must produce a “correctness proof”. This evident difficulty

in defining software test provides insight into how software testing has often been described as hard, complex, and among the "dark arts" [35] of software development. However, software testing remains the de facto validation paradigm used in practice.

Since the nature of software development affords it to be quickly checked (or "debugged") while creating code, software testing most likely existed since the inception of software programs in the mid 1940s. By 1950, Alan Turing, one of the most influential computer scientists of all time, publishes his famous Turing test, which aims to confirm if a program exhibits intelligence. The Turing test could be considered the earliest software test case meant to validate a program's behavior, which, for this test, was to convince a human interrogator that he or she was communicating with another human being.

Gelperin and Hetzel [18] divide the history of software testing into several distinct periods based on the prevalent model used at the time:

- Until 1956 – The Debugging-Oriented Period
- 1957–1978 – The Demonstration-Oriented Period
- 1979–1982 – The Destruction-Oriented Period
- 1983–1987 – The Evaluation-Oriented Period
- 1988–Now – The Prevention-Oriented Period

The Demonstration-Oriented Period is distinguished from the Debugging-Oriented Period in that the former focuses on determining whether a program solves a problem rather than making sure a program runs. The Destruction-Oriented Period viewed testing as an approach to actively discover as many faults as possible in a program. As software grew into more of a professional engineering discipline in the 1980s, testing moved from an activity performed at the end of development to one integrated throughout the entire software development lifecycle. The contemporary view

of testing places it squarely within the development process as an effective preventer of faults and unintended behavior.

From the time when testing simply meant debugging, the topic of software testing has now swelled to a point where it could fill numerous books. In the same manner as software development, software testing has evolved to into an activity resembling a science rather than something that seemed unstructured and undisciplined. It has made major strides over the years, but, at the same time, the scale of the software testing task has become increasingly more difficult to manage. Choosing from fundamental tasks such as component-level testing to more advanced techniques like correctness proofs, software developers and testers must make informed decisions as to the testing strategy to implement. More often than not, the answers to the "how", "why", and "what" questions posed when deciding on a testing strategy are seldom supported by hard evidence. In reality, other factors, such as time, money, and people's reputations, often end up disproportionally steering the testing effort.

CONSEQUENCES OF MEDICAL DEVICE SOFTWARE FAILURES

Section 201(h) of the FD&C Act defines a medical device as [12]:

> …an instrument, apparatus, implement, machine, contrivance, implant, in vitro reagent, or other similar or related article, including a component part, or accessory which is:
>
> - recognized in the official National Formulary, or the United States Pharmacopoeia, or any supplement to them,
> - intended for use in the diagnosis of disease or other conditions, or in the cure, mitigation, treatment, or prevention of disease, in man or other animals, or
> - intended to affect the structure or any function of the body of man or other animals, and which does not achieve any of its primary intended purposes through chemical action within or on the body of man or other animals

> and which is not dependent upon being metabolized for the achievement of any of its primary intended purposes.

Between 1938, when the FD&C Act was passed, until the 1980s, the regulations were fairly effective in ensuring the safety of medical devices. During this period, medical devices were as low tech as tongue depressors, and ultrasound machines were considered technologically advanced. The rapid advancement of microprocessors in the 1970s enabled functionality to migrate from hardware to software. These new high tech devices were able to gain approval under the regulations at the time because their software was simply regarded as a "black box" that was a part of an overall system to be inspected by the FDA. The incidents involving the Therac-25 machine changed everybody's fundamental view of software used in medical devices.

Mirroring the trends seen by product manufacturers at the time, software had become more prevalent in the Therac-25 than the Therac-20; Therac models belonged to a family of devices that was meant to treat cancer. The predecessor model had relied heavily on electromechanical safeguards to prevent overdoses of radiation, whereas the newer model leveraged software. In at least six incidents between 1985 and 1987, the machine had administered high-energy electron therapy when a safer, low-energy type was requested. The patients involved in these accidents either experienced severe tissue damage or died. Subsequent investigations generally point to the poor design, development, and test of the software as the root causes of these accidents. Investigators cite the manufacturer's overconfidence in the software's abilities as a major factor as well. The once optimistic view of software being equivalent to the safety mechanisms achieved through hardware had now been challenged. It was now time for a new way of thinking on software quality control, and testing was to play a significant role in this transformation.

Chapter 3: Literature Review

Research material on design, development, and quality of software for medical devices have focused primarily on providing guidance in response to the regulations that the FDA has released as well as the various guidance documents that the FDA recognizes. Since the early 1900s, the FDA has evolved and expanded to keep pace with advancing technologies over the years. With every addition or adjustment in the regulations, researchers with medical device software involvement must occasionally rethink, shift, or append to their current approaches to regulatory compliance. Luckily for medical device professionals and researchers, the FDA, much like other federal agencies, does not move very quickly. There is usually a sufficient amount of time for thoughtful discussions to begin and end before the next round of changes comes into effect. These discussions include topics such as the interpretation (and intent) of the regulations, recommendations on design and development strategies, or how to fit cutting-edge technologies within the limits of the regulations. With a plethora of resources available on the subject of software testing, the medical device software developer is able to choose from a sizeable toolkit when building the case to present to regulatory bodies on the safety, efficacy, and reliability of his or her product.

Collectively, these resources provide a solid base for the understanding of the work ahead for the stakeholders in the design, development, and test of medical device software. However, these same resources provide little in guidance and advice that can be practically followed in the imperfect world found within most organizations. This paper attempts to provide “a light at the end of the tunnel” for the aforementioned stakeholders, who may feel fear and frustration on what may seem to be an impossible task of not only producing a complex system that works but also complying with the law. A recurring

statement found in most resources is that software testing is expensive, incomplete, and still – to this day – considered an art rather than science. This paper does not dispute this claim; instead, it provides direction on how best to apply testing strategies to not only meet regulatory compliance but also deliver the highest chance of success.

REGULATIONS AND GUIDANCE DOCUMENTS

The seminal piece of legislation that all medical companies in the United States must follow is FDA's Title 21 of the Code of Federal Regulations (21 CFR), which governs food and drugs within the United States. A majority of the regulations enforced by the FDA with regard to medical devices are listed in parts 800 to 1299 (21 CFR 800 to 21 CFR 1299). These sections include the following:

- Medical device definition – a description of what is and what is not considered a medical device
- Device classification – based on the risk that the medical device presents to the patient, it may be designated as Class I, Class II, or Class III
- 510(k) Clearance to Market – 510(k) pre-market notification submission allows new or modified medical devices to more quickly enter the market if they are deemed similar to preexisting approved devices
- Establishment Registration – manufacturers and other establishments are required to register with the FDA
- Device Listing – the FDA requires registered establishments to publicly list their commercially available devices
- Medical Device Reporting (MDR) – all firms must report any complaints, serious injuries, or deaths associated with their medical device to the FDA

- Good Manufacturing Practice (GMP) – Part 820, also referred to as the Quality System Regulations (QSRs), regulate the planning, design, development, review, documentation, implementation, production, and test of a medical device; it also contains language specific to software validation

The QSRs are the most important to a medical device software engineer since it directly addresses the need for a medical device, and thus any of its software, to have been controlled under a comprehensive quality control system. A crucial part of the quality control system is validation. When the FDA published the General Principles of Software Validation (GPSV) in 2002, it became evident to medical device manufacturers that a spotlight indeed does shine brightly on software. This "Guidance for Industry" document provides specific guidance on the activities needed to validate software used in (a) a medical device and (b) the development, manufacture, or control of quality of a medical device. Beyond testing, it details the need for planning, lifecycle models, requirement generation, verification activities, and other aspects of known good software engineering that the agency endorses. Thanks to the passage of The FDA Modernization Act of 1997, these following publications are also standards fully recognized by the FDA:

- ISO 13845 – Echoing the same themes as the QSRs, the International Organization for Standardization (ISO) 13485 standard entitled "Medical Devices – Quality Management Systems-Requirements for Regulatory Purposes" states that its primary objective is "to facilitate harmonized medical device regulatory requirements for quality management systems" [21].
- ANSI/AAMI/IEC 62304 – The International Electrotechnical Commission (IEC) 62304 Standard "provides a framework of life cycle PROCESSES with ACTIVITIES and TASKS necessary for the safe design and maintenance of MEDICAL DEVICE SOFTWARE" [4].

- AAMI TIR36 – The Association for the Advancement of Medical Instrumentation "Validation of Software for Regulated Processes" Technical Information Report (AAMI TIR36) discusses how the general provisions of the QSR apply specifically to non-device software, which is used in the production of a medical device or in the implementation of the medical device manufacturer's quality control system, an example being software that handles patient complaints.
- GAMP 5 – The International Society for Pharmaceutical Engineering (ISPE) "Good Automated Manufacturing Practice" (GAMP) 5 guidance document provides technical details on a flexible risk-based approach to compliant computerized systems used for regulated manufacturing environments, based on "scalable specification and verification" [17].

Another policy that could apply to a medical device software application is 21 CFR Part 11. Recognizing the trend of enterprises moving to a more paperless environment, these rules provide a means to generate electronic records and electronic signatures that are considered legally equivalent to paper records. An example where this legislation would apply could be storage of pass/fail results into a database that is generated from an end-of-production-line custom test software application.

RESOURCES ON SOFTWARE TESTING AND DEPENDABILITY

The current landscape of software testing is expansive and growing every day. Always trying to keep up with the continual changes in software technologies, software testing strives to reach and maintain one of its main goals: ensuring dependable software. Arguably the most influential book on software testing is Glenford Myers' *The Art of Software Testing* [35]. Since its release in 1979, it was the only notable publication for many years until the interest in this field spiked in the Internet-inspired 1990s. Popular

books include Brian Marick's *The Craft of Software Testing* [33], *Software Testing* by Gregory M. Kapfhammer [28], and Cem Kamer's *Testing Computer Software* [27]. A simple Internet search on "software testing" produces thousands of resources speaking to the extreme difficulties in achieving reliable software and the methodologies put forward by the software testing community in an attempt to solve these problems. The following list is a small sample of particularly interesting works:

- "Software Testing Research: Achievements, Challenges, Dreams" [7] – this paper by Antonia Bertolino provides a comprehensive look at the history as well as the current and future desires of the software testing field
- "What is Software Testing? And Why is It So Hard?" [47] – J.A. Whittaker tackles the challenges of testing today's software products head on with an in-depth investigation into the complex problems testers face and the methods to address these problems
- "A Direct Path to Dependable Software" [22] – in this article, Daniel Jackson puts current mainstream approaches to software dependability into question by proposing more "straightforward" and justifiable methods.

RESOURCES ON SOFTWARE QUALITY IN A REGULATED ENVIRONMENT

When one narrows the research space to software quality in a regulated medical environment, the focus of the literature usually shifts from the goal of ensuring dependable software to the goal of producing software with a high confidence of meeting its intended behavior. This phenomenon is most likely a direct result of the way the regulations are stated, which overwhelmingly indicates that software validation is *not* equal to testing. This theme is reiterated in the David A. Vogel's book *Medical Device Software Verification, Validation, and Compliance*. Falling short of actually reading 21

CFR or the GPSV, Vogel's book is as close to a complete reference on the verification and validation, also known as V&V, of software for the medical device industry. In other resources, topics span the medical device software knowledge spectrum. Examinations on the implications of the use of unconventional technologies such as formal methods [25] and static analysis, surveys on real-world software development techniques [13], studies on device recalls due to software [45], and analyses of risk management principles [26] all aim to guide readers through the jungle-like environment shaped by the regulations.

Chapter 4: FDA's View on Software Testing

Undeniably, the FDA does recognize the fact that software has been able to improve the lives of hundreds of thousands of people by providing functionality in all sorts of medical devices that were never thought possible; however, due to the exponential rise of complexity in software and hardware systems, testing alone has become impractical and ineffective. The following passage from Bruce Sterling's "The Hacker Crackdown" [42] sheds some light into the FDA's ambivalence toward the use of software in medical devices:

> The stuff we call "software" is not like anything that human society is used to thinking about. Software is something like a machine, and something like mathematics, and something like language, and something like thought, and art, and information.... but software is not in fact any of those other things. The protean quality of software is one of the great sources of its fascination. It also makes software very powerful, very subtle, very unpredictable, and very risky.
>
> Some software is bad and buggy. Some is "robust," even "bulletproof." The best software is that which has been tested by thousands of users under thousands of different conditions, over years. It is then known as "stable." This does not mean that the software is now flawless, free of bugs. It generally means that there are plenty of bugs in it, but the bugs are well-identified and fairly well understood.
>
> There is simply no way to assure that software is free of flaws. Though software is mathematical in nature, it cannot by "proven" like a mathematical theorem; software is more like language, with inherent ambiguities, with different definitions, different assumptions, [and] different levels of meaning that can conflict.

By all means, the FDA does want software to undergo extensive testing. In fact, testing is one of the only ways to directly gauge the quality of a piece of software. Gauging quality is a step toward *validation*, which based on the QSRs, is defined as "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." The QSRs

expand on this statement by defining *verification* as providing "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…and provides support for a subsequent conclusion that software is validated." By specifically defining these terms, validation and verification gain special importance in the regulations. With the strategic use of testing, verification can be achieved, which in turn, contributes to the validation effort. The relationship between testing, verification, and validation can be visualized in the following diagram [44]:

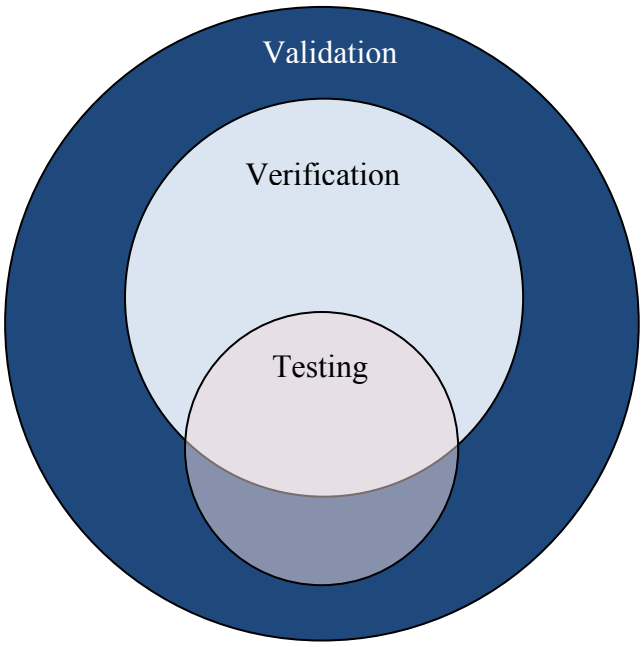

Figure 1: Relationship between testing, verification, and validation.

As seen in Figure 1, there are more activities to validation than testing. This point is reinforced in the following (bold-faced) sentences found in the GPSV: "Software testing is a necessary activity. However, in most cases software testing by itself is not sufficient to establish confidence that the software is fit for its intended use." A majority

of these extra activities are a result of the need for implementation of a Software Development Life Cycle (SDLC) model, as referenced in FDA's definition of verification. Although the FDA does not recommend any particular life cycle model, such as waterfall or spiral, it does view the application of an SDLC as an effective detector and preventer of software errors. Furthermore, an SDLC provides three items in addition to safety assurance, as illustrated in Figure 2, which are of utmost importance to the FDA:

- Control – the FDA demands that a medical device company show that it is in control; an SDLC contains "control points" dispersed among its phases where an assessment of the health of the project may be made
- Organization – the FDA considers organization, a by-product of control, a key characteristic to have; an SDLC identifies elements, such as roles, documents, stages, and tasks, and defines a structure for and relationships between these elements
- Documentation – a common saying heard in the FDA-regulated world is "if it is not documented, it did not happen" [20]; most SDLCs are well suited for the creation and maintenance of plans, requirements, design specification, code reviews, test reports, risk analyses, software metrics, and other documents to provide evidence supporting a claim of regulatory compliance

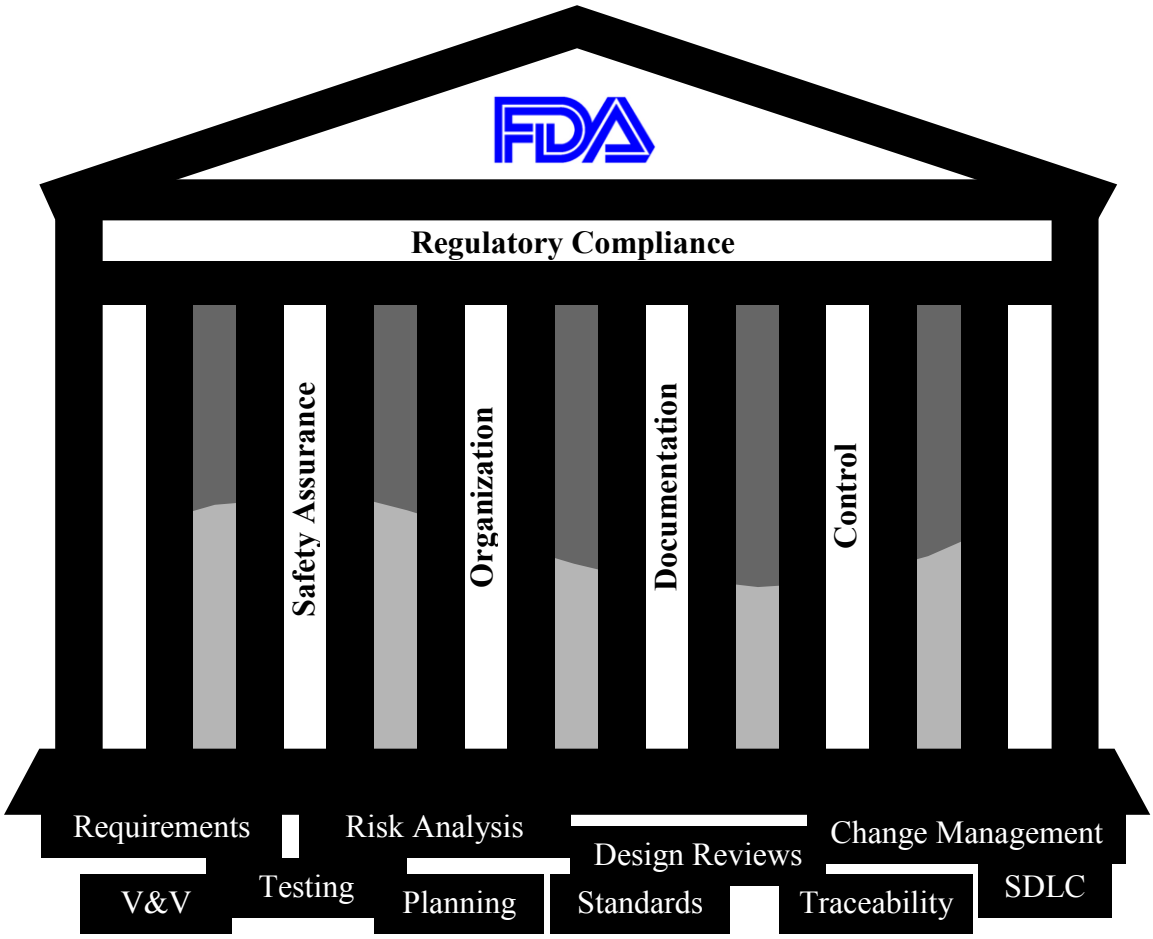

Figure 2: Pillars and foundations toward regulatory compliance.

Chapter 5: Software Testing State of the Art

Much like the way the animal kingdom has been carefully grouped into specific categories through years of research, the subject of software testing has been thoughtfully decomposed and classified based on the specific characteristics of each testing methodology. A glimpse into the discipline's vast and complex landscape can be seen in the following tables from the "Guide to the Software Engineering Body of Knowledge" (SWEBOK) [6], which provides a comprehensive breakdown of the knowledge area of software testing:

Testing Basic Concepts and Definitions	Testing-related terminology	Definitions of testing and related terminology
		Faults vs. Failures
	Theoretical foundations	Test selection criteria/Test adequacy criteria (or stopping rules)
		Testing effectiveness/Objectives for testing
		Testing for defect removal
		The oracle problem
		Theoretical and practical limitations of testing
		The problem of infeasible paths
	Relationships of testing to other activities	Testing vs. Static Analysis Techniques
		Testing vs. Correctness Proofs and Formal Verification
		Testing vs. Debugging
		Testing vs. Programming
		Testing within Software Quality Assurance (SQA)
		Testing within Cleanroom
		Testing and Certification

Table 1: Decomposition for Testing Basic Concepts and Definitions.

Test Levels	The target of the test	Unit testing
		Integration testing
		System testing
	Objectives of testing	Acceptance/qualification testing
		Installation testing
		Alpha and Beta testing
		Conformance testing/Functional testing/Correctness testing

		Reliability achievement and evaluation by testing
		Performance testing
		Stress testing
		Back-to-back testing
		Recovery testing
		Configuration testing
		Usability testing

Table 2: Decomposition for Test Levels.

Test Techniques	**Based on which tests are generated**	Based on tester's experience	Ad hoc
		Specification-based	Equivalence partitioning
			Boundary-value analysis
			Decision table
			Finite-state machine-based
			Testing from formal specifications
			Random testing
		Code-based	Reference models for code-based testing (flow graph, call graph)
			Control flow-based criteria
			Data flow-based criteria
		Fault-based	Error guessing
			Mutation testing
		Usage-based	Operational profile
			Software Reliability Engineered Testing (SRET)
		Based on nature of application	Object-oriented testing
			Component-based testing
			Web-based testing
			Testing of concurrent programs
			Protocol conformance testing
			Testing of distributed systems
			Testing of real-time systems
			Testing of scientific software
	Ignorance or knowledge of implementation	Black -box techniques	Equivalence partitioning
			Boundary-value analysis
			Decision table
			Finite-state machine-based
			Testing from formal specifications
			Error guessing
			Random testing
			Operational profile
			SRET
		White-box techniques	Reference models for code-based testing
			Control flow-based criteria
			Data flow-based criteria
			Mutation testing
	Selecting/combining techniques		Functional and structural
			Coverage and operational/Saturation effect

Table 3: Decomposition for Test Techniques.

Test Related Measures	Evaluation of the program under test	Program measurements to aid in planning and designing testing
		Types, classification and statistics of faults
		Remaining number of defects/Fault density
		Life test, reliability evaluation
		Reliability growth models
	Evaluation of the tests performed	Coverage/thoroughness measures
		Fault seeding
		Mutation score
		Comparison and relative effectiveness of different techniques

Table 4: Decomposition for Test Related Measures.

Managing the Test Process	Management concerns	Attitudes/Egoless programming
		Test process
		Test documentation and workproducts
		Internal vs. independent test team
		Cost/effort estimation and other process measures
		Termination
		Test reuse and test patterns
	Test activities	Planning
		Test case generation
		Test environment development
		Execution
		Test results evaluation
		Problem reporting/Test log
		Defect tracking

Table 5: Decomposition for Managing the Test Process.

In addition to well-established software testing concepts, leading-edge technologies aim to compensate for the shortfalls of software testing. To addresses input state explosion problems, static analysis is a fast-growing field that checks the correctness of all executions in a program (without actually executing the program) through the use of specialized languages and solver engines. Symbolic execution methods run programs, but instead of using actual values, *symbolic* values are tracked and

analyzed. Byzantine fault tolerance [10] is a much-researched redundancy measure that defends against Byzantine failures, which normally occur in distributed software systems where its components arbitrarily fail, presenting different symptoms to different components. Self-healing software [29] is a reactive fault protection technique that allows software to monitor, diagnose, adapt, fix, and test itself. As long as software keeps heading toward greater complexity, as exemplified in trends toward high mobility, parallel/multi-core execution, and always-on connectivity, the subject of software testing will continue to evolve and grow.

There are several books and papers that go into detail on each software testing method and theory, and it is beyond the scope of this paper to dive into all of them. The software tester should take solace in the availability of a large variety of tools to pick from. At the same time, the task of choosing the "right" tools that will make the FDA "happy" may seem highly intimidating. The remainder of the paper explores the mindset to have when developing software for a medical device.

Chapter 6: Practical Guidance

KNOW THE REGULATIONS (AND THE REGULATORS)

Before playing any sport, one must know the rules of the game. This also applies to the environments overseen by the FDA. A clear understanding of the contents of the regulations is a prerequisite to complying with the regulations. The rules are established and readily available for anyone who wishes to develop medical device software. There is a plethora of official records, written mostly in legalese, as printed documents, in electronic form, and available through the Internet. Alternatively, there are several support websites, articles, books, seminars, and other resources available to guide those who need to sift through the meanings behind these legal documents.

While examining the regulations, it is important to understand the motivations behind the policies that have been put in place. One way to do this is to put oneself in the shoes of the regulatory body and its staff. A history of what has and can go wrong is well documented, as indicated in the prior sections of this paper. One can assume that every auditor, investigator, engineer, or other employee of the administration has been educated on the storied and turbulent history of medicine and medical devices. It is also safe to assume that regulatory staff members recognize the great responsibility placed upon them. The FDA, its laws, and its people are the last line of defense for the public against harm from possibly dangerous devices. As evidenced by the tone and extent of the literature, software risks deserve special attention. As a result, investigators are more likely to be particularly alert when medical devices contain software or have been affected by software in some significant manner.

Although the regulations are strict in their objectives in protecting the public, the FDA does recognize the realities facing the medical device community. As seen in many of its guidance documents, the FDA fully acknowledges the burden that federal

regulations place on technological progress of the devices it regulates. In fact, the other goal of the administration is to "speed innovations" as stated in the second paragraph of its "What We Do" web page [46]:

> FDA is also responsible for advancing the public health by helping to speed innovations that make medicines more effective, safer, and more affordable and by helping the public get the accurate, science-based information they need to use medicines and foods to maintain and improve their health.

The fact that the regulations do not specifically state that device manufacturers follow a specific software engineering technology, technique, or model is intentional. Because of the complexities, intricacies, and dynamic nature of the software industry, the FDA realizes that it is not and never will be the subject matter expert in this field. *No* particular programming language is mandated, *no* particular software development lifecycle model is preferred, and there is *not* an authorized list of test tools to use. The regulations require the end result to be safe and effective devices, but their flexibility allows manufacturers to continually improve their devices based on what the industry and research community find most appropriate.

What does this mean for testing activities of medical device software? Admittedly, the FDA has focused their regulations on ensuring that vendors follow accepted design processes rather than encouraging software to be extensively tested. However, the regulations do require that vendors provide "objective evidence" that their medical devices meet prescribed intended uses. Testing is one of the only techniques that allow vendors to meet this requirement. It is fully expected that those involved in medical software validate their systems for intended use, but other test methods and software engineering tools that go beyond this minimum requirement must also be explored. As discussed in the prior sections of this paper, testing can only go so far and software failures pose serious consequences to the well-being of end users. Not only do those

involved in medical devices have a legal obligation, but there is also an unwritten ethical obligation to protect people from harm at all costs.

It is also important to note that stiff punishment awaits those who fail to comply with the federal statutes. The FDA may inspect a medical device manufacturer at any time. Similar to the way that regulations on software quality are published, the objectives of inspections and audits are also publicly accessible. Investigations are categorized by level and have specific protocols governing findings, reports, follow-ups, conclusions, and enforcement. In the event of noncompliance, the FDA has these enforcement tools available at its disposal [8]:

- Form FDA 483 – form used by the FDA to document and communicate concerns discovered during inspections
- Warning Letter – voluntary compliance
- Seizure of Product – administrative detention
- Injunction – stops business operation; in extreme cases, firearm-carrying special agents may execute arrest and search warrants
- Import Alerts – detention that stops devices from entering the United States
- Criminal Prosecution – jail and/or fines
- Civil Penalties – fines or other monetary relief for affected individuals
- FDA-requested recall – a recall initiated by a firm in response to a formal request
- FDA-mandated recall – a recall initiated by the FDA when a firm fails to voluntarily comply to a formal recall request

ASSESS THE ENVIRONMENT

The FDA regulations apply exactly the way they are written whether it is a large, well-established medical device company with decades of experience or a small upstart

company consisting of a few dozen engineers with little experience in the medical field. As a result, an honest assessment of the organization's personnel and established organizational support structure must be made. The level of competence of the development team plays a major role in the successful execution of tasks toward regulatory compliance. Additionally, having managerial and executive support for a well-defined corporate quality system adds to the quality of the products as much as having highly skilled team members. The knowledge level that is needed does not just include items technical in nature but also a healthy comprehension of the demands for quality as expressed in the FDA statutes and guidelines.

The environmental conditions highly influence software testing activities. When dealing with stakeholders from different fields, not all may understand the complex demands involved in the planning, design, implementation, verification, and validation of software systems. For example, the project manager may be overly concerned with meeting schedules and staying within budget and thus downplays the value of testing. As a result, tests are not placed throughout the development plan as described in the GPSV. To make matters worse, the company has not established a quality system to include Standard Operating Procedure (SOP) or Work Instruction (WI) documents that clearly define the company's recommended SDLC model and what tests are required for it. In cases like the one described, it is the responsibility of the ones most involved with the software to remind everyone of regulatory responsibilities and that appropriate planning, review, execution, and documentation of tests help meet these obligations. Supporting material such as the GPSV, AAMI TIR36, and IEC 62304 should be referenced as much as possible.

TEST EARLY AND OFTEN

It is a generally accepted idea that the earlier a defect is detected in the SDLC, the less costly it will be to correct it [23], [44], [39]. Not only is this good software engineering practice, regulatory guidance documents support the practice of proactively and frequently testing software. In Section 5.2.5 of the GPSV, it states that "test cases should be created as early in the software development process as feasible." This theme is reiterated in GAMP 5, where it defines an approach that contains phases for specification and corresponding verification testing as shown in Figure 3 [17].

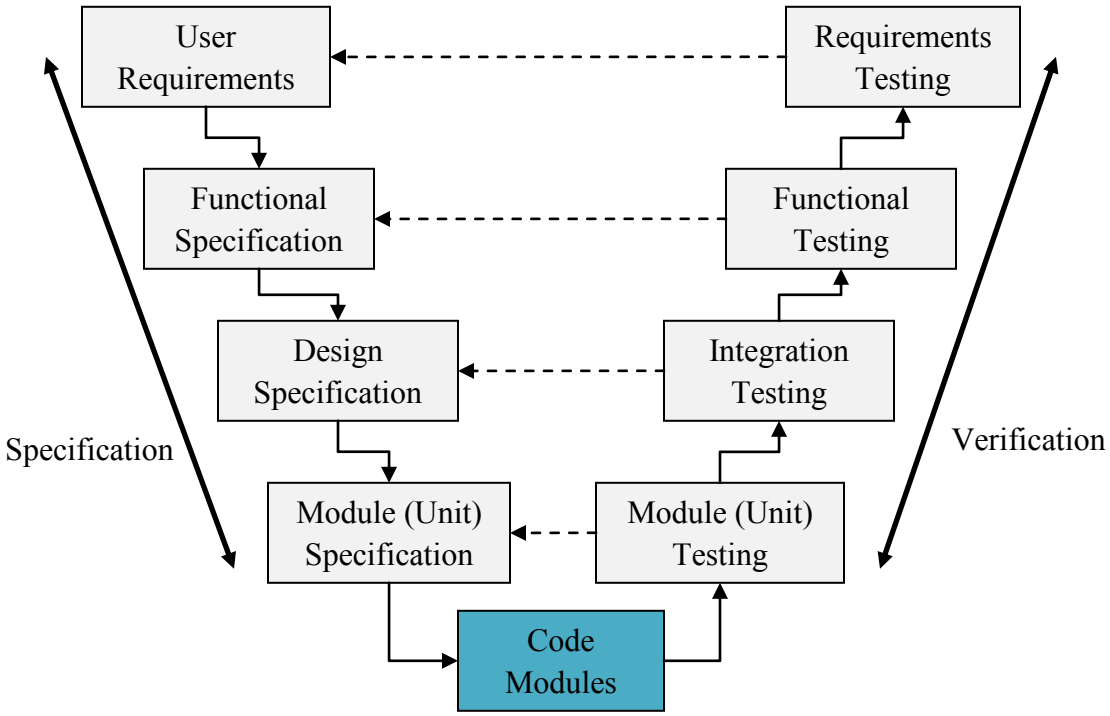

Figure 3: GAMP 5 Approach for a Custom Application.

TEST ACCORDING TO RISK

The FDA ranks safety as its number one concern. As a result, a coordinated effort must be made to reduce the risk of harm to a patient, user, caregiver, or other bystander. Echoing software testing's stance that the earlier a fault (source of harm) is detected, the easier and less expensive it can be to fix, risk analysis should also occur early and often. Coupled with risk analysis is risk management. Risk management is a systematic approach consisting of hazard identification, measure of hazard severity, risk evaluation, risk control, and residual risk control. As guidance, AAMI TIR36 presents a risk model flow chart, shown in Figure 4, that identifies the process and process elements needed to properly perform risk management steps [3]:

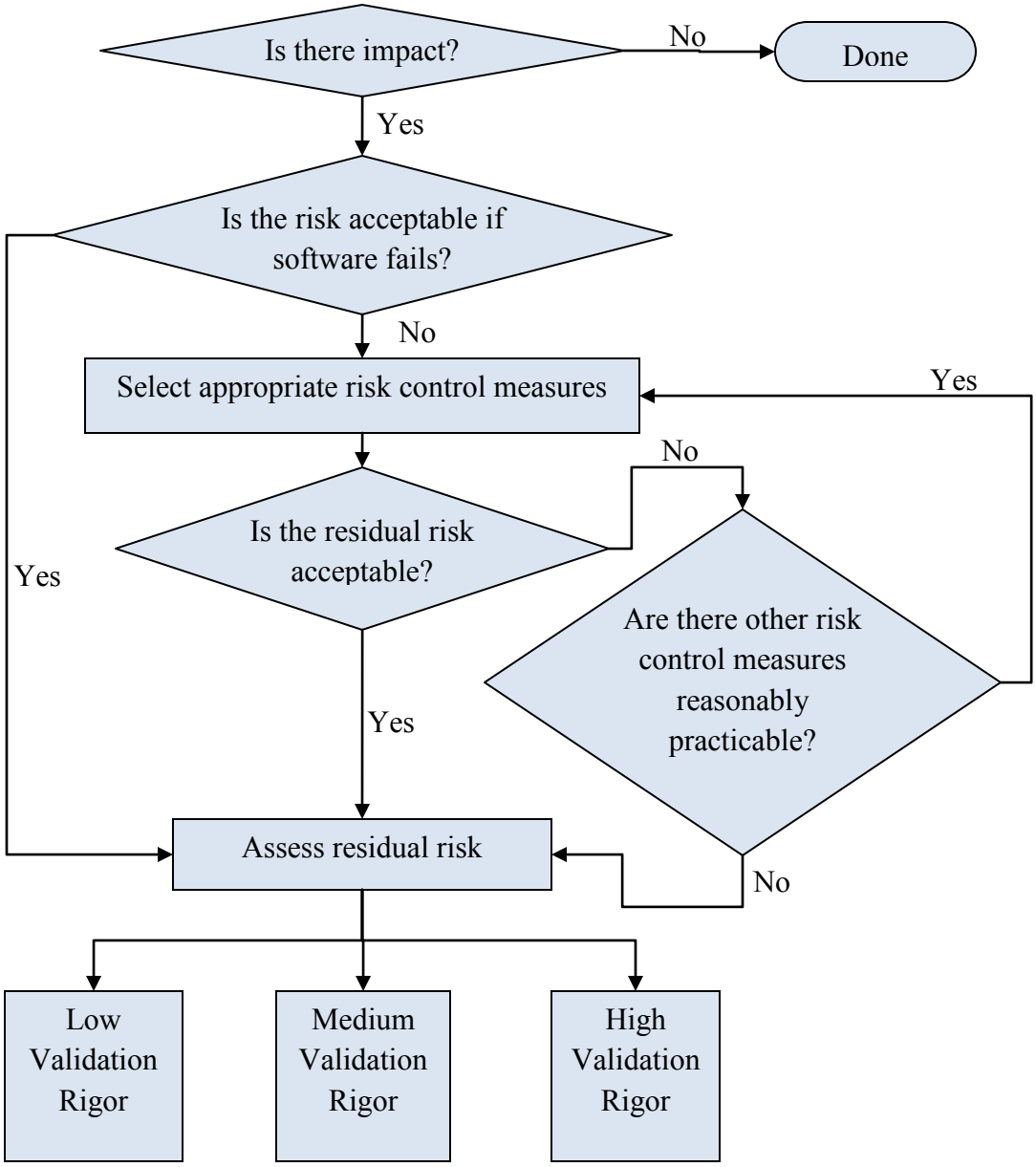

Figure 4: AAMI TIR36 Risk Management Model.

Armed with a complete picture of the risks, a testing strategy may be formulated to more vigorously attack the areas of the software that pose the greatest risk. This approach is endorsed by the GPSV in its recommendation that the "level of validation

effort should be commensurate with the risk." Any testing technique may be used to increase the confidence that risks have been minimized, and it is left to the owners of the software to provide justification for the test selection. Rather than asserting that a testing strategy reduces the *probability* of a failure, test selection should focus on the goal of reducing those failures that can cause the most severe harm. Because software errors are *systematic* in nature, rather than random or fatigue-related as experienced by hardware, a failure can be attributed only to the software's design.

PLAN, EXECUTE, DOCUMENT

The high level of documentation required in medical device company is a major differentiator between it and a company that does not deal with safety-critical applications. Non-regulated companies may implement quality systems following standards such as ISO 9001, the Capability Maturity Model Integration (CMMI) approach, or Institute of Electrical and Electronic Engineers (IEEE) standards, but none of these standards are compulsory and the companies' products may certainly reach the market without tight adherence to them. Furthermore, these standards address risks related to project schedules and budgets, whereas FDA regulations and guidance focus on safety risks and tend to ignore overall project objectives. Because of its importance, medical device company documentation needs to be tightly controlled, complete, consistent, and auditable. Documentation becomes even more critical if a medical device company were to ever face civil or criminal prosecution stemming from device failures.

The first axiom for any woodworker is "measure twice and cut once." For the development and testing of software systems, this principle is also appropriate. Having good test plans demonstrate that careful and thorough thought had been put in before any potentially expensive test activity had been performed. Moreover, early planning reduces

the risk of tests suffering low quality due to testers facing uncertainties or pressures when it comes time to execute the "cut". In a convincing endorsement, Section 5.2.5 of the GPSV states that testing "is a time consuming, difficult, and imperfect activity. As such, it requires early planning in order to be effective and efficient." When devising a test plan specifically for medical device software, these should be the main questions to ask:

- What requirements will be tested?
- How does the testing reduce identified risks?
- How will the testing be performed?
- How will the software be configured during tests?
- Who will perform the tests?
- How long will testing take?
- What will the test coverage be? I.e., what is the required quality level (based on risk)?

Recall that the QSRs place special emphasis on the *validation* of software systems and that validation involves showing *objective evidence* that the software meets *user needs* and *intended uses*. The first direct effect of this policy is the need for an established set of requirements, preferably finalized in the early stages of the SDLC. In turn, the generation of requirements creates several pieces of documentation that begin with plans and end with documents that show evidence that the product has met its pre-established requirements. An excellent framework to refer to is the IEEE Software Standards Collection [41], which lists 22 standards that address professional software engineering practices. The IEEE standards, although not specifically required by the FDA, lists the documents that have potential to be generated at the intermediate stages of a typical SDLC. For software testing activities, the most applicable standard is IEEE 829, also known as the "829 Standard for Software Test Documentation". In addition to example

documentation, this standard provides templates, proposed outlines, and explanations of purpose for a test plan, test design specification, test case specification, test procedure specification, test item transmittal report, test log, test incident report, and test summary report. Lastly, the requirements traceability matrix, a key component for validation, is specified in IEEE 1016, entitled "Recommended Practice for Software Design Descriptions".

One may ask the obvious question: "Exactly how much documentation should be produced?" Documentation is an inherent component of good software engineering, so it should *always* be an important consideration. As stated previously, the level of testing directly depends on the level of risk. Similarly, the level of documentation should be determined by the complexity level of the system. It may be argued that risks increase in a more complex system, which suggests that the level of documentation depends on risk as well. The absolute minimum expectation is to show documented evidence that all requirements have been formally tested, but this is rarely a successful strategy in meeting regulatory compliance. It is left to the software team to determine the appropriate level of documentation by analyzing tradeoffs and providing justifications for including certain documents and omitting others. The level of documentation imposed on the team should not be too extreme as to cause an unnecessary burden, and it should not be so low that it fails to comply with the law. The wise use of testing methods, e.g. scripts, automated testing tools, testing frameworks, and self-documenting test code, can help lessen the labor of writing (and oftentimes rewriting) documents.

LEVERAGE THE TOOLKIT

The FDA realizes that developers cannot test forever. To exhaustively check all aspects of a software program has been shown to be expensive, impractical, and often

impossible [35], [27]. Therefore, regulatory policy concedes to the idea that software should, instead, be shown that it meets an "acceptable level of confidence". Beyond showing software has been validated for established requirements, the software professional can take advantage of the entire testing toolkit in the development of a risk-based "acceptable level". Moreover, the implied flexibility of the regulations permits the use of any software test methodology that has shown to increase the likelihood of a safe and effective shippable product.

The level of sophistication of the testing effort is limited by cost, schedule, and technology. Take the verification model shown in Figure 3 as an example. This model specifies test methods that target different areas of requirements specification. Test cases could be written as protocols containing specific steps, input values, expected results, and acceptance criteria. Execution of these test cases could be as simple as a person following each instruction and hand writing output data and results. On the other hand, an "xUnit" testing framework could be used to automatically input values, execute the software under test, and output a report. This type of test automation considerably decreases the testing time as compared to manual testing. As an added benefit, automated test program generation provides testers the opportunity to tap into their creativity and analytical skills. However, inclusion of methods such as test automation can only be possible if sufficient resources and time are available to plan, develop, and validate the *software performing the tests*. This rationale may also be applied when determining the feasibility of using advanced techniques such as static analysis and model checking.

Chapter 7: Conclusion

This paper has presented an overview of medical device software testing in the context of FDA regulations. Software testing is an imperfect but necessary activity, and the FDA recognizes this fact. The FDA also recognizes the grave dangers in faulty software. Laws and guidance material stress the importance of device safety and effectiveness yet limit implementation details to allow for flexibility in formulating software testing strategies. Six guidance points have been presented in this paper to enable medical device software stakeholders, such as Calvin the software developer, to draw nearer to achieving regulatory compliance. This guidance is not meant to be definitive; it instead seeks to reflect the spirit of the regulations by pulling from real-world medical device industry practices. This fact still remains: it is ultimately left to the professional software teams and individuals to adapt recommendations to the project at hand.

Glossary

21 CFR 820 – Chapter 820 of Title 21 of the Code of Federal Regulations

AAMI – Association for the Advancement of Medical Instrumentation

ANSI – American National Standards Institute

CMMI – Capability Maturity Model Integration

DHF – Design History File, record containing or referencing the records necessary to demonstrate that the design was developed in accordance with the approved design plan and requirements

FDA – Food and Drug Administration

FD&C Act – Federal Food, Drug, and Cosmetic Act

GAMP – Good Automated Manufacturing Practice

GMP – Good Manufacturing Practice, regulations promulgated by the FDA to require that proactive steps are taken to ensure safe, pure, and effective products

GPSV – General Principles of Software Validation; Final Guidance for Industry and FDA Staff

IEC – International Electrotechnical Commission

IEEE – Institute of Electrical and Electronics Engineers

ISO – International Organization for Standardization

ISPE – International Society for Pharmaceutical Engineering

MDA – Medical Device Amendment

Moore's Law – A prediction by Gordon Moore stating that the number of transistors per square inch doubles every year

MDR – Medical Device Reporting

PMA – Pre-Market Approval, a private license granted to the applicant for marketing a particular medical device

QSR – Quality System Regulation

SDLC – Software Development Lifecycle

SOP – Standard Operating Procedure, a written document or instruction detailing all steps and activities of a process or procedure

SMDA – Safe Medical Device Act

TIR – Technical Information Report

WI – Work Instruction, a written document or instruction detailing how someone should perform the job

xUnit – Unit testing frameworks that include such frameworks as jUnit, qUnit, cppUnit, and nUnit.

Bibliography

[1] 21 CFR Part 11; Electronic Records; Electronic Signatures Validation, Guidance for Industry, Center for Devices and Radiological Health, Food and Drug Administration, U.S. Department of Health and Services, 2001.

[2] A Research and Development Needs Report by NITRD. High-Confidence Medical Devices: Cyber-Physical Systems for 21st Century Health Care, http://www.nitrd.gov/About/MedDevice-FINAL1-web.pdf.

[3] AAMI TIR36:2007, Technical Information Report, Validation of software for regulated processes, Association for the Advancement of Medical Instrumentation, 2008. http://www.aami.org.

[4] ANSI/AAMI/IEC 62304:2006, American National Standard, Medical device software – Software life cycle processes, International Electrotechnical Commission, 2006.

[5] Benet, A. F. A Risk Driven Approach to testing Medical Device Software, *Advances in Systems Safety,* Springer, Dordrecht, 2011.

[6] Bertolino, A. Software Testing (Chapter 5). in *Guide to the Software Engineering Body of Knowledge SWEBOK, 2004 Version*, (Los Alamitos, CA, 2004), IEEE Computer Society, 69-98. http://www.swebok.org.

[7] Bertolino, A. Software Testing Research: Achievements, Challenges, Dreams. in *Future of Software Engineering, 2007*, (Minneapolis, MN, 2007), IEEE Computer Society, 85-103.

[8] Coleman, K. A. FDA's Medical Device Inspection Process. in *11th Conference of the Global Harmonization Task Force*, (Washington, DC, 2007).

[9] Cardiac Rhythm Management, *Altera*. Retrieved October 1, 2011: http://www.altera.com/end-markets/medical/crm/med-crm.html.

[10] Castro, M. and Liskov, B. Practical Byzantine Fault Tolerance. in *Proceedings of the Third Symposium on Operating Systems Design and Implementation* (New Orleans, LA, 1999).

[11] Define Test – Google Search, Google, 2011. Retrieved October 29, 2011: http://www.google.com/search?q=define+test.

[12] Definitions. 21 CFR, § 201 (2004).

[13] Feldmann, R.L., Shull, F., Denger, C., Host, M., and Lindholm, C. A Survey of Software Engineering Techniques in Medical Device Development. in *Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability*, (Boston, MA 2007), 25-27.

[14] Fowler, K. Mission-critical and safety-critical development. *Instrumentation & Measurement Magazine, IEEE*, 7 (4), 52-59.

[15] Fu, K. Reducing Risks of Implantable Medical Devices. *Communications of the ACM*, 52 (6), 25-27.

[16] Fu, K. Trustworthy Medical Device Software, *Institute of Medicine Workshop on Public Health Effectiveness of the FDA 510(k) Clearance Process*, The National Academies Press, Washington, D.C., 2011.

[17] GAMP 5, A Risk-Based Approach to Compliant GxP Computerized Systems, International Society for Pharmaceutical Engineering, 2008. http://www.ispe.org.

[18] Gelperin, D. and Hetzel, B. The Growth of Software Testing. *Communications of the ACM*, 31 (6), 687-695.

[19] General Principles of Software Validation; Final Guidance for Industry and Staff, Center for Devices and Radiological Health, Food and Drug Administration, U.S. Department of Health and Services, 2002. http://www.fda.gov.

[20] Headlee, D. The Paper Trail: CRFs, Source Documents and Data Collection Tools. *SoCRA SOURCE*, May, 2004, 30-33.

[21] ISO 13485:2003, Medical devices – Quality management systems – Requirements for regulatory purposes, International Organization for Standardization, 2003. http://www.iso.org.

[22] Jackson, D. A Direct Path to Dependable Software. *Communications of the ACM*, 52 (4), 78-88.

[23] Jenkins, N. *A Software Testing Primer, An Introduction to Software Testing*. Creative Commons, 2008.

[24] Jetley, R. and Chelf, B. Diagnosing Medical Device Software Defects Using Static Analysis, 2009. Retrieved September 28, 2011: http://www.mddionline.com/article/diagnosing-medical-device-software-defects-using-static-analysis.

[25] Jetley, R., Purushothaman I. S., and Jones, P. A Formal Methods Approach to Medical Device Review. *Computer*, 39 (4), 61-67.

[26] Jones, P. L. et al. Risk Management in the Design of Medical Device Software Systems. *Biomedical Instrumentation & Technology*, 36 (4), 237-266.

[27] Kaner, C. *Testing Computer Software, 2nd Edition*, Wiley, New York, 1999.

[28] Kapfhammer, G. *Software Testing*, CRC Press, 2004.

[29] Keromytis, A. The Case for Self-Healing Software, *Columbia University*, 2003.

[30] Lee, I. and Pappas, G. J. High-Confidence Medical Device Software and Systems. *IEEE Computer*, 39 (4), 33-38.

[31] Levenson, N. G. and Turner, C. S. An Investigation of the Therac-25 Accidents. *IEEE Computer* 26 (7), 18-41.

[32] Meerts, J. The History of Software Testing, 2010. Retrieved October 22, 2011: http://www.testingreferences.com/testinghistory.php.

[33] Marick, B. *Craft of Software Testing: Subsystems Testing Including Object-Based and Object-Oriented Testing*, Prentice Hall, PTR, 1994.

[34] The Museum of Questionable Medical Devices, 2011. Retrieved September 1, 2011: http://www.museumofquackery.com.

[35] Myers, Glenford J. *The Art of Software Testing, Second Edition*. John Wiley & Sons, Inc., Hoboken, NJ, 2004.

[36] Nadler, R. Software Verification vs. Validation, Bob on Medical Device Software, 2009. Retrieved March 29, 2011: http://rdn-consulting.com/blog/2009/03/26/software-verification-vs-validation.

[37] Oliver, D. P. A Comparison of Software Development Methods in a Regulated and Commercial Environment. *Software Quality Professional*, 8 (2), 23-29.

[38] Rakitin, R. Coping with defective software in medical devices. *Computer*, 39 (4), 40-45.

[39] Ray, A., Jetley, R. and Jones, P. Engineering High Confidence Medical Device Software. in *The 2nd Joint Workshop On High Confidence Medical Devices, Software, and Systems (HCMDSS) and Medical Device Plug-and-Play (MD PnP) Interoperability*, (San Francisco, CA, 2009).

[40] Rockoff, J. D. Flaws in medical coding can kill, *Baltimore Sun*, 2008. Retrieved October 9, 2011: http://articles.baltimoresun.com/2008-06-30/news/0806290232_1_software-fda-officials-medical-device.

[41] Software Engineering Standards Collection. *Institute of Electrical and Electronics Engineers*, IEEE Press, Piscataway, NJ, 1999.

[42] Sterling, B. *The Hacker Crackdown*. Bantam Books, 1992, 31-32.

[43] Vanac, M. *MedCity News*, 2010. Retrieved August 27, 2011: http://www.medcitynews.com/2010/04/billions-of-stimulus-dollars-flow-for-electronic-medical-records/.

[44] Vogel, David A. *Medical Device Software Verification, Validation and Compliance*. Artech House, Incorporated, Norwood, MA, 2010.

[45] Wallace, D. and Kuhn, R. Failure Modes in Medical Device Software: An Analysis of 15 Years of Recall Data, *ACS/ IEEE International Conference on Computer Systems and Applications*, 2001, 301-311.

[46] What We Do, *FDA*, 2010. Retrieved September 13, 2011: http://www.fda.gov/aboutfda/whatwedo/default.htm.

[47] Whittaker, J. A. What Is Software Testing? And Why Is It So Hard? *IEEE Software*, January/February 2000, 70-79.

[48] X-ray computed tomography, *Wikipedia*, 2011. Retrieved September 22, 2011: http://en.wikipedia.org/wiki/X-ray_computed_tomography.