Agile Methods:

Most are not ready for prime time in medical device software design and development

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

as published in DesignFax Online, July 2006

The agile methods community has been gaining strength for almost a decade. A number of medical device manufacturers have started to experiment with these methods to see if, in fact, greater efficiencies can be gained in medical device software development.

Software managers who are more familiar with the traditional waterfall development lifecycle (a sequence of concept, requirements, design, implement, test phases) and its derivatives are often skeptical of, and almost fearful of, agile methodology. Some even believe that the agile crowd resembles more of a "software cult" because of the way in which these special-interest groups first began organizing themselves around the "Agile Manifesto".[1] However, software giants like Microsoft have begun to use agile methods successfully and are encouraging their further use. Can the agile techniques, successfully used to develop user-interface-intensive web-based software, be adapted to develop real-time embedded software common in medical devices? Do the techniques promoted by the agile crowd produce the design and requirements artifacts – the objective evidence – that the medical device regulatory agencies such as the FDA seek? While there are some basic problems reconciling some of the agile methods with regulatory requirements, the medical device community could learn much by experimenting with agile methods. But adopting them wholesale is unwise and premature.

What are agile methods?

The agile movement is still very young. Agile methods are largely a collection of loosely associated methodologies covering the design, development,

COMPANY PROFILE

Intertech Engineering Associates, Inc.

Address:	100 Lowder Brook Avenue Suite 2500 Westwood, MA 02090 www.inea.com - (781) 801-1100
Industry:	(Electro)Medical Devices
Services:	Assessments Training Consulting Hands-on Engineering
Skills:	Product Design Risk Management Requirements Engineering Electronics Development Software Development Software Verification and Validation Production/Quality System Software Validation

management, and testing of software. Among the most widely touted agile methods are the following: Extreme Programming, or XP, is perhaps the most visible and controversial of the agile methods. It is often referred to as "lightweight" software development because it sidesteps the creation and maintenance of detailed software requirements or detailed design documentation. Programmers work in small groups that are further broken down into pairs. Each pair shares a computer. One person writes code while the other audits to cross check each other's work. The software evolves through definitions of requirements, called "stories", that are short enough to be described on index cards. The process is intended to be rapid, with software examples quickly being written and presented to the customer for either acceptance or incremental change to the "story" requirements. Traditional techniques try to discourage change in the late phases of the development lifecycle due to increasing costs in maintenance of the software and its interrelated requirements, design and test documents. The XP methodology, on the other hand, encourages constant change and theoretically responds to change faster because its lightweight nature carries no supporting documentation to update with changing requirements. Scrum is a project-management method whose name is borrowed from rugby. The basic concept of scrum is to organize around short-term and long-term goals. The long-term goals, called "sprints", are expected to be accomplished in 30 days of development activity. Short, or "stand up," meeting, or scrums, of 15 to 30 minutes are held daily to review the progress of the previous day and to set goals for the upcoming day. Test Driven Development, or TDD. This sounds like an appealing methodology, given the software verification and validation requirements for medical device software. TDD focuses on developing tests for the software items before the software is written. The tests are automated so that they can be rerun in their entirety on a build-by-build basis. The TDD sequence is: develop tests for the software functionality, write the software, run the tests, fix the software and "refactor". Refactoring is an agile concept that requires redesigning or rewriting software to correct for poorly partitioned or redundant code modules.

What has prompted the development of agile techniques?

Software engineers universally have recognized the need to deal with change in the software development lifecycle more expeditiously. Traditional lifecycle models, like the waterfall model, ignore the iterative nature of software development. Other iterative models, like the spiral model and the iterative waterfall, do account for changing requirements, but do so by updating requirements and designs in each iteration before the code is changed … at least that's the theory. Software engineers who try to follow traditional lifecycle models often complain that too little of their time is spent engineering software and too much is spent on documentation. Software engineering teams often follow traditional lifecycle models on paper, but ignore the model's proper sequencing of tasks. The engineers jump into modifying code first and document their changes (or what they remember of them) later. They feel they have accelerated the process by getting software released faster. They almost always feel that the documentation is largely a waste of time. And for them it is. The benefit of documenting requirements is to facilitate the development of the software, not to chronicle the design history. Avoiding the "heavyweight" lifecycle models that require documenting first, implementing last, is the primary motivation behind the experimentation with agile methods, particularly XP. The underlying cause for the burdensome documentation is the perception of constantly changing requirements. The goal of XP in particular is to encourage change, to respond to it with rapid software iterations, and to tie the reality of the implementation to the "customer's" expectations through tests that are developed ahead of the software. The scale of modern software projects also has driven the software industry to examine alternative methods. It has long been recognized that the efficiency of work groups declines as the group size increases[2], especially where complex interactions among group members are required. This is mostly due to the increased amount (and lower efficiency) of communications required as group size increases. With many medical device software projects ranging from hundreds of thousands to millions of lines of code, it is impractical to expect a small group to develop all the code in a reasonable period of time. Traditional development lifecycle models deal with this by communicating via written requirements, specifications, and designs. Agile methods, on the other hand, try to break a project down to subcomponents that can be implemented by smaller teams that are more efficient. In particular, the agile management methods of Scrum attempt to make large projects look like a series of small projects. The process of breaking large projects into smaller subprojects is well known outside Scrum. The difficulties lie with integrating the small subcomponents into one cohesive product. The integration difficulties can only be magnified if the underlying technical methodologies challenge communications by being light on documentation.

Are agile methods appropriate for medical device software?

Much software developed today is for information technology or business process automation, and often, the work process being automated by software is murky because it was not well understood when it was done manually, or it was done in many different ways by many different people. It's no easy task to write solid software requirements for an ill-defined underlying process, especially when user opinions and preferences are involved. Usually, the development of this kind of software is a discovery process that involves early storyboarding, rapid prototyping, etc., to elicit user opinion about what parts of the design work and what parts do not work. If the software team commits to very detailed requirement and design documents prior to implementing and communicating the vision to the customer or user, the process is, indeed, burdensome. In contrast, software-driven medical devices are generally developed to automate medical diagnostic or therapeutic procedures that are very well defined. In fact, it may be seen as quite appropriate for a medical device to restrict the creativity of the user. Many devices now sport graphic user interfaces (GUIs). In these cases, some of the same issues with user opinion and preference do exist, and at least the GUI may benefit from the accelerated evolution of XP. However, other than for the GUI functionality of a medical device, the evolutionary trial-and-error methods that are formalized by XP are simply inappropriate. It is not reasonable or even legal to put a series of intermediate versions of medical device software in the hands of users for "intended use" applications with intended users. Finally, the XP methodology in particular may be difficult to reconcile with the FDA's regulations and guidance documents that relate to medical device software.

Regulatory requirements

The FDA has given medical device software special attention since the 1980s when the Therac 25 incident resulted in several patient deaths and injuries that were traced to software. The investigation of this incident implicated specific defects in the software. Perhaps more revealing were the software design and development processes and procedures found to be contributing factors to the catastrophe. Unlike electrical and mechanical failures that may be random (such as component failure), software failures are always systematic design failures. Although electrical and mechanical subsystems too may have systematic flaws, with software all defects are design defects. This, coupled with the fact that software is very complex, easy to change, and that changes can be largely invisible, led the FDA to the conclusion that it was the design of software that needed to be controlled. The Quality System Regulations have specific requirements for design control (21 CFR 820.30) that require the documentation of specifications and designs. Furthermore, they require controlled technical and management review of the documentation. The FDA's guidance document, the General Principles of Software Validation[3] (GPSV) is very specific in its expectations that requirements be explicitly documented prior to implementation and prior to the development of test procedures. Although the GPSV is a guidance document, and is therefore not mandatory, the burden is upon any medical device manufacturer that uses techniques that are counter to this guidance to establish that its methodologies are at least as good as those in the GPSV. This creates an added burden for the regulatory pre-market notification submissions for a medical device. Furthermore, any unusual practices will likely be a target of special attention from the FDA on any quality system inspections. Part of the regulatory requirement is to leave a trail of design and development artifacts (objective evidence, in FDA terms) that can be audited by the agency to prove compliance with the design control regulations. The "lightweight" methodologies in XP, in particular, may lead to some "heavyweight" problems with the FDA's search for objective evidence. XP was developed to encourage late changes in software requirements and to facilitate code re-use in its refactoring activities. Unfortunately for XP, some analyses[4] have singled out code re-use and late requirements changes as contributing factors in the Therac 25 catastrophe. It may be an uphill battle to convince regulators to get behind a development methodology that encourages late change and code re-use with little supporting documentation.

Should the regulations change?

Regulations should definitely change with the state of the art. It certainly was never our legislators' intention to stifle creativity or productivity. The legal machinery moves slowly, and seldom is based on taking risks. If agile methods are to be recognized by medical device regulators, there will need to be a large body of evidence that they provide all the same good design control practices and crosschecks that their predecessors do. Unfortunately for agile enthusiasts, the FDA is not moved by development speed or efficiency; its mission is to ensure the safety and effectiveness of devices. Agile devotees will need to prove agile can produce devices that are just as safe and effective and that their design practices can be monitored by regulators just as effectively.

Do any of the agile methods fit into the current regulatory environment?

As a project-management method, Scrum is somewhat independent of engineering methodologies and probably can be wrapped around more traditional engineering techniques and development lifecycles to make product development more efficient in a way that is well within the bounds of the current FDA regulations. The Scrum method manages outputs (releases, prototypes etc.) in blocks of work that can be accomplished in roughly 30-day sprints.

Requirements implementation and change backlog are managed within each sprint. Short-term, daily management activity is organized in daily huddles or scrums in which engineers report on progress since the last meeting, planned progress before the next meeting, and perceived obstacles to achieving that progress. Although Scrum methods are usually thought of as being part of an XP development environment, there is nothing about Scrum that requires XP. In fact, the methodology looks like it would apply equally well to requirements development, software design, software development, test design and development, and test execution in more traditional lifecycle models. However, other agile methods get so specific in their implementations that they may not be practical for medical devices without some radical changes.

Conclusions

The agile methods interest groups are to be applauded for their work in experimenting with new software development techniques. There is no question that software is so broadly defined that different types of software may require different development methodologies, just as they benefit from different development languages, tools, operating systems, and so on.

It may be too fanciful to ever expect that a regulatory agency charged with responsibility for inspection of design activity would agree to less (let alone no) documentation of software requirements and designs. However, the methodologies are intriguing, and some methodology is better than chaos. Perhaps some of the methodology defined for extreme programming (XP) could be modified to create novel methodologies for extreme specification (XS), extreme design (XD), and extreme verification and validation (XVV). The key to widespread use of agile methods in the medical device industry will be to reconcile the activities and outputs with regulatory expectations.

References

[1] www.agilemanifesto.org

[2] Brooks, Frederick P., Jr. The Mythical Man-Month Essays on Software Engineering, Addison Wesley Publishing Company, 1975

[3] General Principles of Software Validation; Final Guidance for Industry and FDA Staff, Food and Drug Administration, Center for Devices and Radiological Health, January 11, 2002.

[4] Nancy Leveson, Safeware: System Safety and Computers, Addison-Wesley, 1995

Acknowledgements

Kristen Barnico and Robert Demilia, both of Intertech Engineering Associates, Inc. provided background assistance in the background research and editing of this article.

ABOUT THE AUTHOR:

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

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

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

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

Leverage INTERTECH's expertise to:

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

Intertech Service Offerings:

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

ABOUT INTERTECH:

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

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

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

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

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