Experts from IBM and Diagnostic Grifols describe how Diagnostic Grifols is using agile software development approaches in the tightly regulated field of medical devices. "Many organizations believe that they must use a waterfall approach in regulated environments," they say. "We show how this is not the case."

Introduction

This article describes how agile development practices can be adopted in the highly regulated medical device field. This approach can be easily adopted by organizations working in other regulated fields.

Although medical device regulations do not enforce a fixed lifecycle model, activities are presented in a sequential manner, thus hinting at a waterfall process. Meanwhile, for a decade or more, software teams have benefitted from agile development methods. Several medical device manufacturers have adopted agile practices while keeping development in compliance with regulations, but conflicts arise and decisions have to be taken in favor of agility or formality. The Association for the Advancement of Medical Instrumentation (AAMI) has produced a new Technical Information Report, TIR45, that gives guidance on the use of agile practices in medical device development (see the Related topics section for a link). This "Medical device software - Software life cycle processes" report adds new insights on the subject by mapping IEC 62304 activities into the agile development process (also cited in Related topics).

In this article, you will learn:

- How Diagnostic Grifols evolved its practices with this new report in mind
- How Diagnostic Grifols attained significant gains in productivity with a simultaneously agile and compliant approach
- How this type of approach could be used in other regulated industries, as well

Diagnostic Grifols recently underwent a major United States Food and Drug Administration (FDA) audit and passed with no major issues raised. The audit covered software that Diagnostic Grifols developed using agile approaches.
Regulatory principles, activities, and deliverables

There are two main approaches to regulatory control: prescriptive and descriptive.

- In the **prescriptive** approach, the regulations define what must be done and, in many cases, how it is to be done.
- In the **descriptive** approach, the regulations define what must be achieved and largely leave the question of how to achieve it to the regulated organization.

This does not mean that enforcement is any less strict. Indeed, in most cases it means that regulatory inspections are more searching, because the organization under inspection has to explain to the inspectors exactly how they achieved the results, rather than just checking a box on a form.

Regulators are moving increasingly toward the descriptive approach. This is the case, for example, for the FDA Quality System Regulation 21 CFR Part 820 (see Related topics. Most medical device manufacturers worldwide need to adhere to this regulation, as it is mandatory for devices manufactured or sold in the United States.

Differences in waterfall, agile, and V-model development approaches

Many different development approaches have been proposed over time. In this section, we describe three of the most popular approaches.

**Waterfall development**

The fundamental principle of a waterfall approach is that development proceeds in stages and does not move from one stage to the next until each stage is considered complete. As Figure 1 shows, it starts with user and system requirements, and then progresses through design, development, integration and verification, to installation and validation. Thus, at the end of each stage, you either have a complete description of the system, at some level, or a complete system. This characteristic is often seen as making it easier to be compliant with regulations, because each complete description can be checked against its predecessors.

The waterfall approach is often characterized as a process that permits flow in only a single direction. However, it does allow for feedback between stages as issues are discovered.
There are two main advantages of the waterfall approach:

- It works well in a contractual environment, because the contract can be let against a well-defined set of requirements.
- It avoids the problems of developers working without sufficient context to know whether they are creating something that is not maintainable or will prevent other needed functions being implemented.

However, in practice, it is rare if not impossible to be able to define requirements completely before starting development work, and if that is done, they are often obsolete before the project is completed. Typically, in waterfall approaches, companies actually iterate in each area, and the next area starts before the first is finished so development can prove that the requirements can actually be met. This means that you are actually doing some agile style development without the rigors of an agile process.

**Agile development**

Agile development is a radically different approach. It recognizes that requirements will change as more is learned about the way that the proposed system will be used. It places the emphasis on working systems (typically software), rather than enforcing a process with strict deliverables.

If this approach is not carefully managed, it could degenerate into a process with all of the problems that the waterfall was designed to avoid, so various means are used to avoid this. In fact, agile approaches often require more discipline than waterfall methods.
The main control mechanism used in agile processes is to break the development up into a number of sprints. Each sprint aims either to deliver a well-defined subset of the overall required functionality or to perform well-defined restructuring of the architecture of the system. This is known as refactoring in agile terminology.

The approach described in the rest of this article is how Diagnostic Grifols has successfully applied agile development methods. In addition, they integrated the scrum project management method that is often combined with agile development practices (scrum is derived from "scrimmage," the full-team huddles in rugby). Figure 2 shows an outline of this approach.

Figure 2. Outline of the scrum project management lifecycle

The V-model process

We should also mention the V-model here, as it is often seen as being just another way of looking at a waterfall approach. The main distinction from the waterfall method is that each of the activities at each level of development (the left side of the V) is related to the relevant activity for verifying and validating the system (the right side of the V). Although viewing it as an extension of the waterfall was the original intent, most modern interpretations treat the V-model as representing relationships between activities rather than a set of stages. In many ways, it does not matter whether we consider the items as activities that produce assets or as the assets themselves.
If we regard the V-model as representing relationships, it is clear that it is equally applicable to waterfall and agile development. The only difference is that in agile development, you do not have to complete one activity (with its related assets) before starting another. In fact, you do not even have to create any content in the assets, because they become containers that are populated during development.

**Being agile in a regulated environment**

Given the success of agile development, an inevitable question for companies in medical and other regulated industries is: Can we adopt agile approaches in our environment? In this section, we describe how one company, Diagnostic Grifols, has done this.

**Where agile and regulatory principles reinforce each other even when they seem to collide**

As we stated earlier, regulations currently follow mainly a descriptive approach. This means that regulators expect manufacturers to establish their own development processes and document them, while showing that their processes are robust enough to render safe and effective products when followed. Guidance documents and standards define a set of activities that are expected to be found in these processes, while leaving manufacturers ample room for deciding how to organize these activities in time.

This principle of quality assurance by establishment of robust processes seems to collide with the Agile Manifesto statement that "we have come to value individuals and interactions over processes and tools." In fact, following the agile principle of gathering skilled people and providing them with the means to work well just makes a robust process even more robust. This is because it ensures that teams continuously ask themselves how these are working and whether there are any improvements needed. That's something that regulators explicitly ask manufacturers to do.
Regulatory inspections seek to assess whether established processes were followed or not in the development of the inspected product. From a regulatory standpoint, only objective evidence is valid to support the fact that the process was followed. That leads to certain kinds of documentation that seems to collide with the agile value of "working software over comprehensive documentation." In fact, both agile and regulated principles serve the same purpose if we equate working software with safe and effective software. Documentation is not an end in itself. It is merely a means of showing that the product is going to fulfill its intended use in a safe manner, because it has been developed by following a robust process. Agile principles can help to challenge the documentation that is produced during development to ensure that only valuable materials are produced and wasteful ones are eliminated.

One of the main activities that guidance documents and standards ask for is to define design inputs. These definitions will be the basis for subsequent activities, such as architecture, design, coding, and testing. Regulations also expect a final validation activity that shows that the product is adequate to fulfill its intended use, as defined in those design inputs. Again, this seems to collide with the agile value of "customer collaboration over contract negotiation," although it's actually the opposite. That's because continuous customer involvement during all of the development helps developers fully understand the intended use and user needs. It also means that those needs are translated into a set of design inputs that define the system so that other activities can create the software to fulfill those needs.

Finally, development planning is another of the cornerstone activities that guidance documents and standards ask for to show how the processes that manufacturers establish are tailored to a given product and followed. Although this seems to collide with agile value of "responding to change over following a plan," the fact is that agile puts tremendous emphasis on planning at various levels. Also, regulators admit the need to update plans as development evolves.

**Mapping regulated activities in an agile development process**

The diagram in Figure 4 shows how the following regulated activities map into an agile development process:

- Planning
- Requirements analysis (software requirements specification, or SRS)
- Architectural design
- Detailed design
- Coding
- Unit, integration, system, and regression testing
- Software release
These activities are mapped into an agile cycle, more specifically a scrum process with its time-boxed sequence of sprints. A first iteration establishes the scope of the project, defines a broad plan of the initial releases, and lays the architectural foundations. After that, a sequence of releases that extend throughout the life of the product follows. Every release is planned for at the beginning and consists of a succession of sprints. Before release of the software to the field, a "hardening" sprint is devoted to ensuring that the software meets all regulatory requirements. Optionally, when a release includes a significant number of sprints, hardening sprints are inserted at certain points during development as synchronization points for regulated artifacts. This prevents accumulating either technical or regulatory debts.

The following diagram shows the scrum lifecycle in the context of the entire life of a product.
This mapping of activities tries to make up the best of both worlds by trading off its conflicts, as we discuss in the next section.

Resolving conflicts and making trade-offs

One of the main conflicts that arise in practical life when developing software is the degree of detail and formality of the requirements to implement. It is hard to argue against a complete set of requirements that has been reviewed and approved by all stakeholders as the basis to start the development work. Unfortunately, real life shows that complex systems are impossible to fully anticipate and describe in a set of complete and precise requirements that have no flaws. Add to this the fact that changes will occur through a variety of sources, such as customer feedback, and it is easy to see that a significant amount of effort is often spent in rework.

On the other hand, there are system features that are defined and decided outside of the development team. Every system has its business logic derived from its end use. Customers and other stakeholders (for example, service engineers and manufacturing engineers) are going to ask for a set of features that is adapted to the way that they will operate with the product. Moreover, when products evolve, a big part of the development has to build on existing functionalities. Therefore, it would be naïve to leave all this to a self-organized agile team and start the project without enough effort spent gathering all of the needs. These needs, if successfully addressed, can ensure commercial success for the product.

In addition, timing puts pressure on the development job, making concurrent work attractive as a way to shorten time to market. However, the dark side of parallelism is augmented communication complexity and a bigger impact of rework. Going back to our initial set of requirements, it makes sense to start writing test procedures while developers focus on implementation. In regulated environments, those can take a long time to write and are subject of formal review and approval cycles that delay their availability. Hopefully, this ends with a set of test procedures that are readily available to run after implementation is finished. The problem with concurrency is that when changes arise, they can affect requirements, implementation, and tests, thus multiplying the amount of lost work.
Thinking about the worst scenario, we can imagine a case where requirements are gathered, formally reviewed and approved, and given to developers and testers so that implementation and writing test procedures can start in parallel. If the developer later discovers contradictory requirements, lack of detail, or technical issues that render the desired functionalities impossible or impractical, the work must stop and new requirements must be developed. A significant amount of time devoted to the initial requirements will be lost, and obsolete test cases will need to be rewritten.

In our experience at Diagnostic Grifols, and as TIR45 has also pointed out very clearly, there is a trade-off to make between the conflicting forces that can be summarized in three alternatives:

- Just enough design input rather than complete design input
- Finish-to-finish relationships rather than finish-to-start ones (that is, activities that would take place sequentially in a waterfall approach can take place in parallel)
- Synchronization points between design inputs and design outputs

As the mapping diagram shows, there is a first set of high-level requirements that is formally approved. It must be comprehensive in scope but not complete in detail. This way, the features that the system will implement are recognized and approved by all of the stakeholders. For instance, decisions will be described in this document, but often with a "to be defined (TBD)" note so that details are added as the development evolves. The decisions answer these questions, for example:

- Will the system be able to be connected with the hospital's network to exchange data with the centralized information system?
- Will several systems be connected to enhance throughput?
- Will the system include an automatic self-adjustment algorithm to speed its manufacturing operations?

During the sprints that build functionality into the system, a set of features are selected and an appropriate amount of time is spent in their definition so that they are "clear enough." This means that development can start without spending an excessive amount of work on details that practical issues can challenge later. In addition, a limited amount of time is invested in designing a set of high-level tests that guide an informal exploratory test after the implementation is finished. This way, by taking advantage of the extensive communication of agile teams, the final functionality "emerges" and the proper functioning of code is assessed by exploratory tests.

When the functionality is considered "done," a synchronization task delivers the final set of requirements. These requirements are both detailed and consistent with the implementation of the system, in a finish-to-finish approach that renders regulated deliverables in an efficient manner.

At this point, the other main conflict between the regulated and agile worlds arises: Auditors expect a set of formal documents that are easy to inspect, but the iterative definition of requirements and informal testing that we have just described seems to point to a different end. Here is where hardening sprints are needed to complete these tasks:
• Finish all synchronization activities so that design inputs (SRS, architectural design, and detailed design) are consistent with design outputs (code and testing procedures)
• Export this information to documents that can go through a formal review and approval cycle
• Run the test procedures in a formal manner so that verification records are created

When a release consists of a limited number of sprints (up to about six), this hardening sprint needs to take place only once, at the end. When a totally new product is launched or a major release is involved, we find it necessary to insert a hardening sprint after every six to eight regular sprints. This helps to avoid accumulating technical and regulatory debts that are hard to deal with at the end of the development.

Where tools can help

Continuous iterations where all artifacts change and evolve put enormous pressure on the development team and challenge its capacity to produce working software consistently. Other agile practices, such as continuous integration or automated testing, also benefit from the use of tools.

In regulated environments, the traceability chain that links requirements, architecture, design, implementation, and testing and the control of changes to any of the former is a tough expectation to fulfill. It becomes a daunting task when they evolve and change at the rhythm of every iteration, making it very hard to produce up-to-date documents that make sense when inspected by an auditor. The use of tools that can accommodate these changes and keep the traceability chain consistent becomes invaluable.

From the previous sections, it's clear that requirements, architecture, detailed design, and test procedures are evolving throughout development, but static formal documents are also needed to satisfy the auditors' expectation. That is where tools become relevant.

Real-life examples with Rational DOORS requirements management tools

Diagnostic Grifols selected IBM® Rational® DOORS® requirements management software for systems and advanced IT applications to assist them in generation of regulated artifacts has been. They create and maintain the following items in that tool:

• Software requirements
• Architectural design
• Detailed design
• List of source files that implement the detailed design (but not the files themselves, because those are managed in a configuration management tool)
• System test procedures and records
• Traceability information for all of the above

With the use of the version control and baselining capabilities of DOORS, changes are accommodated and tracked as all of the previous artifacts evolve. When it's time to generate regulated documents, they take advantage of DOORS reporting capabilities by using templates to generate up-to-date information. That information is later introduced in a validated document control system.
Summary and conclusions

This article has outlined development approaches and their applicability to software development in a highly regulated environments. It has also shown how a well-established medical device manufacturer has adopted agile development practices in its highly regulated field. Organizations that work in other regulated fields could use a similar approach.

We have shown that agile practices are not only compatible with regulated industries but can be implemented in ways that provide strong support for compliance with regulations. For Diagnostic Grifols, this approach has produced benefits in productivity, as well as compliance.

Source: Diagrams courtesy of Scott Ambler, Scott Ambler + Associates
Related topics

- For more information related to this article, check these sources:
  - IBM Rational systems and software solutions for the medical device industry: Improve processes, manage IEC 61508 and IEC 62304 standards, develop quality products (IBM solution brief, requires registration, no charge)
  - Three steps to accelerating medical device product development (IBM brochure, PDF format)
  - IBM Rational solutions for electronic devices: Challenges and opportunities for the medical device industry
  - IEC 62304, Medical device software - Software life cycle processes (International Electrotechnical Commission, 2006)
  - Agile Development in the Medical Device Industry by Martin Bakal (DZone Agile Zone, August 2012)
  - Scrum project management with IBM Rational Team Concert Version 4, a two-part article by Millard Ellingsworth (developerWorks, March 2013)
  - The Agile Manifesto and the Principles behind the Agile Manifesto (agilemanifesto.org)
  - Scrum.org website
- To find out more about IBM Rational DOORS requirements management software:
  - Check the Rational DOORS Next Generation page on Jazz.net.
  - For detailed instructions, explore the Rational DOORS Information Center.
- Get the free trial download for Rational DOORS Web Access or try the free trial download of Rational DOORS Next Generation.
- Evaluate IBM software

© Copyright IBM Corporation 2013
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)