12-2009

Agile Software Development and its Compatibility with a Document-Driven Approach? A Case Study

Lise Tordrup Heeager
Department of Computer Science, Aalborg University, liseh@cs.aau.dk

Peter Axel Nielsen
Department of Computer Science, Aalborg University, pan@cs.aau.dk

Agile Software Development and its Compatibility with a Document-Driven Approach? A Case Study

Lise Tordrup Heeager & Peter Axel Nielsen
Department of Computer Science
Aalborg University
Denmark
Email: liseh@cs.aau.dk, pan@cs.aau.dk

Abstract

It is generally assumed among software developers and managers that a document-driven development process is incompatible with an agile development process. There are few reports on this issue and even less empirical research documenting the assumed incompatibility. More and more software companies however have a desire to adopt agile development processes while maintaining compliance with a quality assurance standard or even a process standard. We have studied the software development process of a Danish pharmaceutical company. For market reasons the company has to comply the US Food and Drug Administration's standards for software development. The company has also successfully implemented significant parts of the agile methodology Scrum. We describe this case and we analyse using Soft Systems Methodology how well the development process combines these diverging sets of process requirements. We find that despite much effort the agile development process suffers from its adaptation to the FDA standard, but at the same time many of the genuine qualities of an agile process have been maintained. From this case study we discuss the implications for development companies facing the challenge of implementing a mixed development approach.

Keywords: Agile software development, document-driven software development, process standards, FDA.

INTRODUCTION

Embedded software is a growing industry and a significant part of embedded software is safety critical (e.g., Knight 2002). Whether this safety-critical embedded software is used to control potentially dangerous devices, machinery or infrastructures there is very often a set of requirements compiled into standards, which govern or should govern how software is developed. The European Space Agency has a number of such process standards (Jones et al. 2002). The CMMI (e.g., Chrissis et al. 2003) was invented and promoted for the US Department of Defence as a maturity and process standard for all its software subcontractors to be compliant with. The ISO9000 family of standards for quality management has been suggested and used to standardize also how software companies manage their quality assurance when developing software (Mutafelija & Stromberg 2003; McMichael & Lombardi 2007). For clinical software and embedded software in clinical devices there are several process standards. A medical device for the US market has to be approved by the FDA (Food and Drug Administration) and as part of this approval the software development processes have to be compliant with their process standards (FDA 2002). Medical device approval is becoming increasingly expensive and the desire to be compliant from an early stage with FDA's process standards is thus increasing (Lee et al. 2006; Abdeen et al. 2007). There are many reports suggesting that these process requirements are indeed necessary as safety-critical software would otherwise be too error-prone and too risky (Rakitin 2006; Schrenker 2006). These process standards all require that requirements are specified prior to their design and implementation, that documentation is a cornerstone to achieve high quality, and that documents are only changed through controlled procedures, i.e., what is called a document-driven approach to systems development.

Agile development, on the other hand, is based on the assumptions that programs and programming is primary – not documentation. The agile manifesto for software development has this as one of four major principles. Agile development seeks to increase software quality by adhering to a very flexible process (e.g., Cockburn 2002; Beck & Andres 2004) that relies on the competence of the software developers and not on defined processes (Aaen 2003). Agile development is increasingly popular among software companies as it provides a better explanation of how requirements change over time and how the resulting software therefore fits the final requirements.

There has already been a discussion of whether these two sets of assumptions, agile development and document-driven development, can co-exist and even be combined into a mixed approach. A notable debate has been between Beck and Boehm on whether an agile approach is paradigmatically different from a plan-driven approach (Beck & Boehm 2003). Some conclude from a theoretical study of the approaches that they are very

alike and can co-exist (Turner & Jain 2002; Theunissen et al. 2003; Fritzsche & Keil 2007). Experience reports on a mixed approach are few, but some report how they have taken a rigorous process standard like CMMI as the primary framework and apply the agile approach Scrum within that (Sutherland et al. 2007; Jakobsen & Johnson 2008). Others report from experience with adding quality assurance processes to an agile development approach (Opelt & Beeson 2008). (Rottier & Rodrigues 2008) report from their experience with introducing Scrum in Cochlear, a medical device company. The research on these issues has so far been either theoretical (and also detached from real-world experience) or it has been mere reporting on experience (without any connection to theory). We want with this paper to contribute to a better understanding of how a mixed approach may be practiced and how well it is practiced.

In the following section we explain how we have performed our research as an interpretive case study. This is followed by a section where we describe the case we have studied. Then we present an analysis of the case, together with an evaluation of the mixed approach in practice. After this follows a section where we discuss the findings and then finally we conclude the paper.

THE RESEARCH APPROACH

We studied a pharmaceutical company in Denmark. In particular we studied how a clinical product with embedded software is developed. The product development involves pharmaceutical and clinical researchers responsible for the drugs, process engineers responsible for production facilities, mechanical engineers responsible for the products mechanical functions dispensing the drugs, embedded hardware engineers responsible for the computer chips and communication controlling the mechanics, and software developers responsible for the software controlling the hardware. The project is large, involving more than a hundred engineers and developers and lasting several years. Within this project we have specifically studied the software team and how they have implemented a software development approach that is a mix of a document-driven FDA-compliant process and an agile process.

The study was organized as an single case study. Our research interest was in understanding in an interpretive way (Walsham 1995) how they develop software with this mixed approach and how well it supported their intentions of a reasonably effective and efficient development practice. In particular we wanted to understand this in detail and in the outset we did not know exactly which details we were looking for. We also wanted to understand the developers different views on the mixed approach. For these reasons an interpretive case study approach is appropriate.

Phase	Data collection	Data documentation	Data analysis
1	7 qualitative interviews based on an interview guide	Interviews audio recorded and transcribed	Using Soft Systems Methodology to explore the problem situation
2	8 (4+4) qualitative interviews based on 2 interview guides	Interviews audio recorded and transcribed	Using Soft Systems Methodology with a particular focus on the mixed approach
3	Seminar presentation and validation with 11 members of the software team	Meeting audio recorded and transcribed	Validation of findings about the mixed approach

Table 1: Phases in the interpretive research approach

We collected and analysed the empirical data iteratively through three phases, see Table 1. The first phase was based on a semi-structured interview guide that included the subjects: FDA requirements, agile software development, the requirement specification, handling of errors and general strengths and weaknesses. Seven interviews were conducted with: the project manager, the software architect, a software tester and four developers. The interviews were recorded and transcribed. The analysis was performed using Soft Systems Methodology (SMM) (Checkland & Scholes 1990) by (i) drawing rich pictures of the situation and problems encountered; (ii) creating root definitions and conceptual models of activity systems; and (iii) a comparison between the models and the interviews.

The second phase included eight interviews with: the project manager, the software architect, a tester and five developers. Four of these interviewees participated in the first phase, whereas the last four participated for the first time. The interviews were again audio recorded and transcribed. This second round of analysis also

followed SSM, but now with a single human activity systems and its conceptual model. This model was used first to identify sub activities and then further to compare the intended mixed approach with the development practice of the software team.

In the third phase the findings from the second analysis were written into a report and presented to the software team in a seminar. The seminar included a presentation of the results and a discussion of these. The interviewees from the previous two phases participated.

CASE DESCRIPTION

As outlined in the previous section this is a case study of a pharmaceutical company currently developing a clinical device for dispensing drugs and information processing. The controlling of the drug dispensing is performed by embedded software. In total over 100 managers, researchers, engineers and developers are working on the device project. The project has been running for several years and is soon to enter the final stage of refining the product.

The software development team consists of 17 developers and their project manager working in close cooperation with the engineers developing the hardware. The software development team is divided into two sub teams; one sub team develops software for the remote control to device while the other sub team develops software for the part that actually controls the drug dispensing. Furthermore, two testers, a software architect and a project manager are assigned to the software project. Some years ago it was decided by the project manager that the software team should utilise parts of an agile methodology in their daily practice by introducing iterations of four-week periods each supposed to produce a tested increment of software. The software team chose Scrum (Schwaber & Beedle 2001) as their primary agile method. The overall device project still has to be compliant with a large number of standards, some which apply directly to the software development process (FDA 2002). Maintaining adherence to an agile approach and the FDA standards at the same time has been a difficult management task for the software team. The software project manager has subsequently formulated a mixed software process to solve this.

researchers have developed the model in Figure 1 based on the first round of interviews as a model of the mixed development approach. SSM (Checkland & Scholes 1990) was applied to the problem situation after the first round of interviewing and several human activity systems with accompanying root definitions and conceptual models were tried and then compared with the interview data. The status of the conceptual model in Figure 1 is that it expresses a consolidated view of the mixed approach. It is consolidated only to the extent that the interviewees agree that it expresses well the intentions and assumptions behind the mixed approach.

The input to the mixed development approach is a set of requirements stated at the higher levels of the project organization in overall requirements specifications. The requirements are placed in the project documents such as the product specification and human-computer interaction specification. These documents are developed outside of the software team.

The project is committed to a strict document structure with formal handovers between phases and between different sub projects. Changing the requirements in the product specification is extremely difficult and time-consuming because of the many stakeholders involved in approvals and handovers. The software requirement specification (SRS) is based on the overall specifications.

The SRS serves as the software backlog and is written by the software architect. The SRS constitute the foundation on which the developers must understand and develop their current development tasks (Activity 1). A hazard analysis enlightens the risks associated with the device and its operation, which has to be considered and mitigated. In order to develop a safe and user friendly device the software developers are required to have an understanding of the context of the device, e.g., patients' behaviour and physical space.

The developers will plan their iterations, i.e., iterating activities (Activity 2). Due to the size of the software team they are divided into sub teams during this process. The tasks will be written on post-it notes on a large board on the wall in the team's open office space in order to visually display: which tasks are to be solved, which currently are being solved and which have been solved.

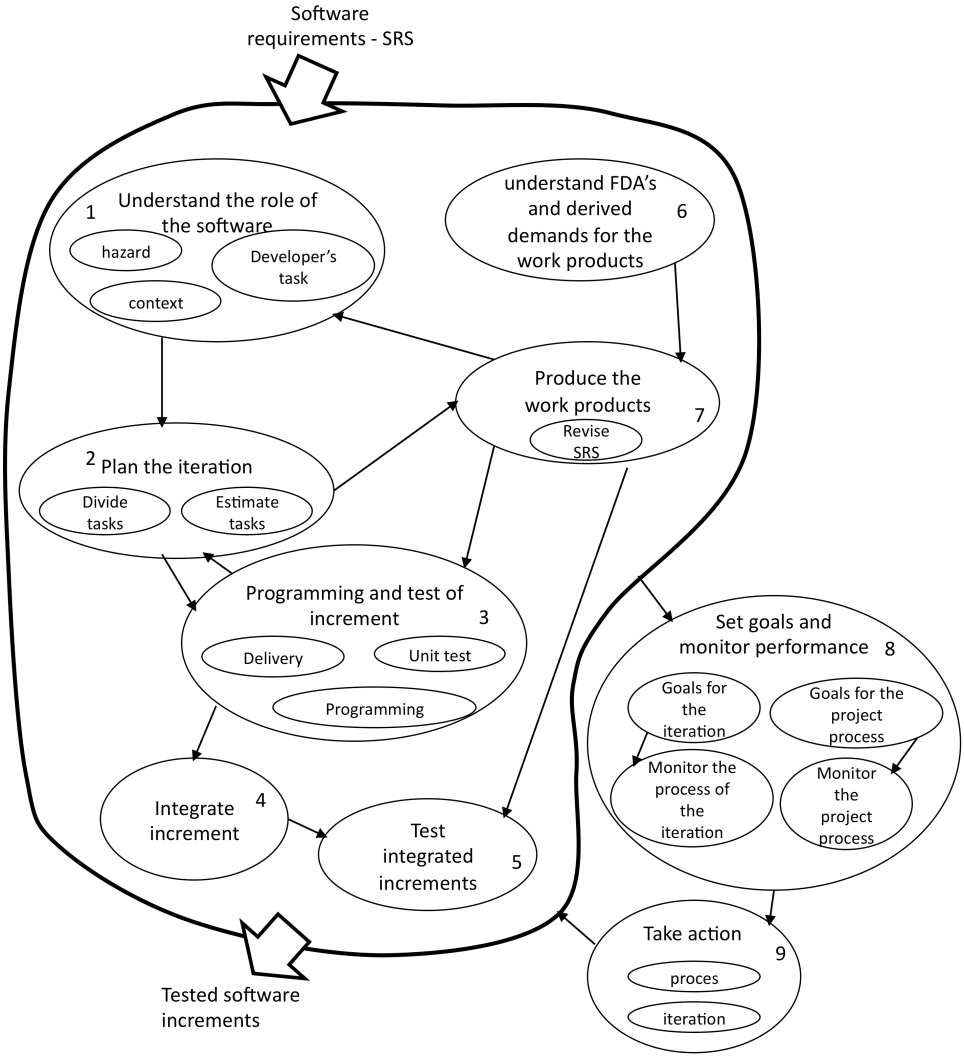

Figure 1: The software team's mixed development approach

The iterations contain programming, delivery and unit testing of the increments (Activity 3). The developers choose tasks from the board and then implement this piece of the software. All tasks are to be peer reviewed by a colleague before being delivered. The coordination during an iteration is done at stand-up meetings every morning lasting 15 minutes. The purpose of the stand-up meeting is for the developers to state their past, present and future (next 24 hours) working tasks in order to give the group an overview of what everybody is working on and make sure no one is getting stuck in a problem. The developers create the unit test cases when programming a task. The developers decide whether they write the test cases before or after programming. The unit tests are run automatically every 24 hours to ensure that new software has not implemented a new error. The software team's development manual states that they must have one hundred percent condition-decision coverage on unit tests.

The new increment, i.e., the result of an iteration, is continuously integrated with the existing system (Activity 4). After integration test (Activity 5) the increment is moved to the system engineering group which runs a device test. Errors are reported back to the software team trough an error reporting system. When an error is reported by the system engineering group several formal procedures must be followed.

The developers must understand the demands from FDA as well as the derived demands for the work products in order to produce these (Activity 6). The SRS is the most important work document in the software development team (Activity 7) and that will be gradually revised and detailed as the project progresses.

The development activities are monitored relative to the goals set for each iteration and for the entire project process (Activity 8). Progress for each iteration will be visualized in burn-down charts next to the task board on

the wall in the open office space. Each iteration ends with an evaluation, which is an extended stand-up meeting where both the process and achieved increments are on the assessed. When needed action is taken in order to improve practice (Activity 9).

ANALYSIS OF THE DEVELOPMENT PROCESS

This section is an analysis of the current software development practice used by the software team. In effect it is an evaluation of the software team's practice based on the human activity system in Figure 1. The analysis consists of three parts. First, the overall structure of the development process is assessed; second, the activities of the development process are examined (including an analysis of how well the activity concerned is performed in the software team); third, the perspective will be broadened and the focus will be on the impact of the project structure, which the software team is a part of.

The Overall Structure of the Mixed Development Process

The mixed approach consists of operational activities that are performed either as sequential or as iterative, and of management activities that are both sequentially and iteratively performed, see Table 2. The core of the mixed process is primarily circular in the tight interchange between the activities: '2: Plan the iteration' and '3: Programming and test of increment', see Figure 1. Several other activities are undertaken concurrently, e.g., '7: produce the work products'; but these are largely sequential.

The inputs to the software development process are the overall requirements for the device as a whole as stated by the management of the device project – outside the software team. The requirements were frozen and written into different documents such as the device specification and the hazard analysis. The requirements were to a high degree formed at the beginning of the device project and were partly based on a user survey. The software team therefore has no direct customer contact or any contact with a potential end-user as recommended by agile methodologies. The software architect of the software team is responsible of interpreting the requirements in the project documents, write them up as software tasks and include them in the SRS. The SRS serves as the software backlog; however tasks are neither created nor prioritized by customers (neither in the sense of potential patients nor in the sense of the device project as a customer) as usually required in an agile approach. The software project's context has thus required sequential problem solving where the SRS could be detailed further, but overall requirements could not be changed.

The main activity of the development process is '3: Programming and test of increment'. This activity is strongly tied to the activity '2: Plan the iteration'. These are done iteratively and follow the Scrum methodology. The sprint backlog is created at the planning meeting and the tasks within it are estimated, the sprints are of four weeks durations and these are controlled by daily stand-up meetings.

The iterative activities are time-boxed and the output is supposed to be a fully working software increment. However in practice not all iterations deliver a new slice of the product and the increments are often not fully tested at the end of the iteration. The activities 4 and 5 were supposed to be iterative, however they are not. It was similarly the intention with activity 7 that it should be performed prior or at least in the beginning of every iteration, but that never happened.

Activities and their linkage	Sequential	Iterative
Operational activities	1, 4, 5, 6, 7	2, 3
Management activities	8, 9	8, 9

Table 2: The actual performance of the activities in the mixed approach

The dual nature of the management activities 8 and 9 reflects that there is in practice management of both the iterations and of the sequential activities.

The Software Team's Activities in the Development Process

In the following each of the activities of the development process of the software team will be analysed.

Understand the role of the software in the device (1). The hazard analysis is necessary due to the fact that the system is highly safety critical. The software team receives the results of the hazard analysis through documentation. Most of these results are formed as requirements for the software in order to mitigate the patients' exposure to the device's risks. To understand each development task the developers consult the SRS to get acquainted with the requirements for the relevant part of the system. If this information is not sufficient they

contact other sources in the project, who have more information on this issue. The software team has in general a superficial understanding of the context in which the product is to be used. In the beginning of the project the developers participated in a course on general medicine. However, the knowledge gained through this course has not been maintained. The project has been ongoing for several years and some substitution and addition of developers has occured. The superficial understanding of the context is also caused by the non-existing user contact.

Plan the iteration (2) and Programming and test of increment (3). These activities are central to the development. The software team performs time-boxed development. Following Scrum they chose iterations of four weeks length. However, they recently expanded the length of an iteration to five weeks. The sprint backlog is created from the software backlog by the developers. They define the tasks for an iteration on the basis of the prioritization of the requirements and each task is then estimated. The tasks in an iteration are displayed on a large board in the open office alongside the burn-down chart displaying the progress in the iteration. Estimates often do not fit within the iteration and tasks are being postponed to the next iteration. This is a problem for two reasons: (1) the tasks which are postponed are often the same, and (2) the developers have gotten used to the fact that they never reach their stated goal for an iteration and have stopped striving for it. The attitude has become: "if we do not manage to implement it in this iteration, we will just do it in the next". The estimates slip not only due to lack of ability to judge the size of the tasks but also because of interference from other groups in the project. These interferences stem primarily from errors discovered and reported by the system engineering group during system test. Some of these are so severe that they required to be corrected right away – hence the time-box is broken. The hardware group and the human-computer interaction group also interfere inside an iteration with new requirements details just invented – hence without a discipline the time-box is again broken.

The iteration is controlled by the daily stand-up meetings (Activity 8). The duration of these is approximately 15 minutes. Due to the size of the software team the stand-up meetings have taken too long time when everybody gets to speak. Therefore one person from each sub team is now responsible of getting a short summary from his co-workers and presenting this at the stand-up meeting. This has however reduced the quality of the information presented. As one sub team is much bigger than the other sub team most of the meeting time concerns their tasks and is often to a high degree irrelevant to the other sub team.

All development tasks are to end with a peer review of developed software performed by a colleague before the task can end. Each developer is responsible for implementing this. But even though the developers participating in the interviews of this study expressed great satisfaction with the review process, too few peer reviews are actually conducted. This may have several reasons: (1) the developers perceive the implementation of additional software more important than doing the peer review, (2) some developers do not like to have their software criticized by others, and (3) some developers regard their software as flawless and therefore see no point in taking the time to having it peer reviewed.

The requirement that unit tests should provide 100-percent condition-decision coverage stems from an interpretation of the FDA standard. However, the current unit test coverage is much lower and the performed unit tests find too few errors in the software. These two issues seem to be interconnected. When testing is insufficient the full benefit does not show. Developers are thus de-motivated and see little point in unit testing. The project manager and the software architect are the problem owners who are responsible for this activity. It is generally acknowledged that the unit tests do not find enough errors and that everything is not tested, however, the problem has so far remained unresolved.

Integrate increment (4) and Test integrated increments (5). The software team has not yet defined an integration process or a process for testing the integrated software. The process descriptions are being formulated by one of the software testers. Currently, integration takes place concurrently with activities 2 and 3, but works independently hereof.

Understand FDA (6) and Produce work products (7). The development manual of the software team was developed on the basis of an interpretation of the FDA standards. The software requirement specification, the human-computer interaction navigation flow and the software architecture are the primary work products for the software team. These are updated continuously; but never synchronised with the iterations. The software architect is the primary responsible for maintaining these documents. The software architecture has troughout the project undergone several radical changes and was not in place until recently, which is several years into the project. This entailed many changes of the software and has made it difficult to implement an increment at each iteration into a working prototype.

Set goals and monitor performance (8) and Take action (9). Activity 8 is divided into two parts. The first part is setting the goals for and then monitoring the iterations. At the beginning of each iteration the goals are set in the iteration plan. The process is primarily monitored by visual checking of the tasks on the board and by the burn-down chart. The second part is setting the goals for and then monitoring the software project as a whole.

The overall goal of the project process is not clear to the developers in the software team. Contradicting goals are communicated from the higher levels of the project hierarchy and create confusion in the software team. The software team has not defined measures of performance for monitoring the software project. This issue is properly connected to the contradicting goals. Action is taken when the goal for an iteration is not reached, then the remaining tasks are moved to the next iteration. Actions are taken to improve the software project on the basis of a feeling that something needs to be improved, but this is not based on monitoring systematically.

The Impact of the Project Structure

The software team is only a small part of the entire project with more than 100 people being involved in different roles. The overall management structure is bureaucratic and the software team is the only part of the organization using an agile methodology. This influences the software development process. The hardware group for example, which the software team works closely with, works primarily sequentially as the hardware takes time to develop and each prototype is very expensive. The hardware has therefore not been available for the software team until late in the project. The early software has consequently only been tested on a simulator at the end of the iterations.

The coordination of the device project takes place on two levels: (1) at the manager level and (2) at the developer level. At the manager level the coordination takes place at a monthly device project meeting. The software project manager represents the software team in this meeting. This coordination suffers from too many stakeholders thus leading to very difficult decision-making. The software team depends on the decisions to be taken and they are the most interested in effective coordination because they are more flexible that other groups. At the developer level the coordination takes place when the developers talk to each other. The software team primarily needs to coordinate with the hardware group and the human-computer interaction group. As these are placed in the same building this coordination is relatively easy. However, they have limited understanding of how these two other groups work and what they are working on, and this limits the coordination.

The software requirements have changed several times during the project. This has mainly been due to the discovery that two requirements contradicted each other or due to the device project management changed its mind about a feature. The requirements are easy to change in the SRS and in the sprint backlog. However, it is very difficult to change a requirement in the higher-ranking documents because many stakeholders from different groups have to agree on this change.

The fact that the system test is not done by the software team, but by the system engineering group, limits the flexibility of handling the errors found at this test. When an error is found at the system test a new entity is created in the error reporting system. This reporting system handles the process of correcting the error. However, this process is very long and time-consuming as many developers have to validate and approve the changes before the error can be corrected and deleted from the task list. It is opposite with errors found at the unit tests performed by the software team as they are handled in a less formal way and the documentation load is kept at a minimum. To ensure agility in the process of handling errors it would therefore be advantageous to find as many errors as possible before releasing the software to the system engineering group.

DISCUSSION

The software team has successfully implemented parts of the agile methodology in their mixed software development approach. They are facing several problems both due to internal issues in the software team and external issues caused by the project structure. The internal and external issues are highly intertwined and their possible solution similarly interdependent. In this section the results from the analysis are discussed and their possible alleviation are related to the literature.

As the software team has no direct customer contact and receives requirements from the device project management this part of the approach becomes very document-driven. This is very much in line with traditional requirements engineering (e.g., Kotonya & Sommerville 1998; Sawyer et al. 1999). Theories of agile methodology, on the other hand, recommend close customer contact throughout the development of the product to ensure that the end-product meets the needs and expectations of the customer (Schwaber & Beedle 2001; Cockburn 2002; Beck & Andres 2004). As the requirements currently only are processed by the software architect on the basis of written project documents before entering the software backlog there is a high risk that misunderstandings occur and the product will not fit the needs of the end-user. However, this product is developed for a market and is not ordered by a known customer. This makes it difficult or impossible for the software team to create such contact on its own. But to improve the agility of the process of dealing with the requirements one or several persons from the marketing group could be assigned the role of a user (e.g., Grudin 1994). As the marketing group have in part developed the requirements in the project documents on the basis of a user survey, they are the people inside the device project having the best overview of the user needs. They could

therefore serve as a middleman between the users and the software team and present the results of the user survey and ideas of the marketing group to the software team for example in the form of user stories. This suggestion is not a simple solution to this issue as the marketing group currently do not have allocated resources for it and the device project management have yet to recognize the need for more agility in this overall device project before assigning more resources.

The software team is expected to perform time-boxed iterations with a fully implemented and tested increment as output. However, the increments are often not fully tested before they are moved to the system engineering group. Testing is crucial in agile development. Most notable in XP, testing is a core practice and development is generally test-driven (Beck & Andres 2004). It is not only suggested that test cases are written before the software; it is also most importantly suggested that there is testing in of all parts of an increment and that completion is determined by the increment passing all tests. The software team does not perform unit tests to an adequate degree and their unit tests do not find a sufficient amount of errors. The connection seems to be that when the amount of unit tests is not sufficient the software team does not experience the full benefit of the tests and then the unit tests seem for them to be not worth the effort. As the tests do not find enough errors the developers are de-motivated to test further. The developers recognize that the unit tests do not work properly and that the software is not sufficiently tested. However, many of the developers do not see any value in unit testing and the implementation of new functionality seems to be valued much higher than testing. The task board and the burn-down chart give a nice overview of the remaining tasks of the iteration. However, developers also seem to be emphasising new functionality prior to testing as this is the only parameter, which the progress of the iteration is measured according to. This results in a too low quality of the software, in low stability and in several errors, when increments are delivered to the system engineering group. This is a serious issue and perhaps the main problem facing the software team. Errors detected at the system test have to be handled much more formal than the errors detected at the unit tests and integration tests, which decreases the agility. It also means that it is very difficult to determine whether an increment is complete and the iteration has been ended properly. Resolving this issue is not easy, as it will require a significant change of the developers' attitude to increase and improve the unit tests.

Initially and until recently the software team followed the recommendations of Scrum and ran iterations of the length of 30 days (Schwaber & Beedle 2001). As the increments of the software team often did not get fully tested it was decided by the project manager and the software architect to expand the iterations with one week, giving the developers 4 weeks to develop the tasks and one additional week to stabilize the software. At the time of this case study the software team had not yet finished a 5-week iteration and they were therefore not able to reflect on whether this is a solution to the problem. Unstable software did force the software team from time to time to break off from the iterations and focus on stabilization instead. This contradicts the intentions with Scrum. The problems of getting the increment fully tested taken together with the developers expressing frustration of having to implement a full increment during 4 weeks was an indication that the iterations would have to be longer. Further, the software is complex due to its safety criticality and because it is embedded in unique hardware; hence the amount of written documentation is higher than in regular agile development projects.

It was frequently mentioned during the interviews that the estimates of the iterations often do not fit, as the tasks took longer than expected or other important tasks were being pushed into the time-boxes. When the estimates did not fit some tasks were postponed to the next iteration. It seems that the estimates did not fit due to several reasons: (1) the developers do not seem to fight to reach the goal of the iteration as they do not see the importance anymore, (2) at the planning session the developers are positive and push everything they can into the iteration leaving no time for interruptions, and (3) there are many interruptions from outside the software team. In the agile methodologies interruptions during iterations are not allowed, however, in this case several of the interruptions have to be dealt with as they are serious matters affecting the whole device. The interruptions caused by severe errors in the existing software would be reduced if the quality of the increments were increased. However, the interruptions from other project groups working sequentially undergo changes in requirements as all other projects and these are more difficult to align with. This affects the software team in a significant way. This resembles the difficulty experienced by agile software teams when trying to align with organisational change processes, which are sequential too (Bygstad & Nielsen 2003; Bygstad et al. 2009). This can only be resolved if the management of the larger project understand the potential of agile software development.

The peer reviews are considered useful by the developers and several of them wish peer reviews were done more frequently. The reasons that peer reviews are not done more often anyway seem to be: (1) some of the developers have the opinion that their own software is of high quality and that they will not gain anything from a peer review, (2) some developers feel a great ownership of their software and fear getting it critiqued, and (3) new functionality has high value and the time for peer reviews is not considered when the task is estimated. As with the issue of testing the increment, this issue also seems to need a change of attitude of the developers. They need to focus more on the quality of the software instead of the number of finished tasks. Peer review is a document-

oriented technique where peers read the text and assess its quality (Freedman & Weinberg 1982; Weinberg 1992). Whether this can co-exist with the emphasis on agile methodologies cannot be answered by this case study. If the software team would peer review many their software and the documentation hereof they would immediately adhere better to the FDA requirements, but the linkage between the documentation and the software is so far not really traceable.

Taken together, there are aspects of the development approach being practiced by the software team that is clearly agile (though there are parts they are not practicing well). There are also aspects that are clearly practiced in a document-driven way (which they have to practice well to remain FDA compliant). In addition, the linkages between the agile parts and the document-driven parts are not well performed. This includes: (1) little or no feedback from iteration planning (Activity 2) to producing the FDA-compliant documentation (Activity 7), (2) integration and integration testing are not performed (Activities 4 and 5), and severe interference from the outside sequential activities into the iterations. Hence, the mixed approach is a truly mixed approach, but it is not an integrated approach. In particular, it is not an integrated approach where the agile and the document-driven parts are aligned.

CONCLUSION

In this paper we have studied the software development process of a software team in a pharmaceutical company developing a highly safety critical embedded software system for drug dispensing. The software and the device in which it is embedded are therefore to be compliant with the FDA standards. To answer our research question whether agile software development is compatible with the document-driven approach, we have developed a model encompassing both the agile and the document-driven aspects and we have used that to evaluate the development practice of the software team.

The mixed software development process was designed to combine these two approaches. The software team is at the same time agile and compliant with large parts of Scrum as they run time-boxed iterations with an increment as the output. The tasks are controlled by the use of product and sprint backlogs, by the use of a task board and by the use of a burn down chart. The iterations are coordinated through daily stand-up meetings. The software team is however also document-driven as the requirements are fixed early in the project and there is no customer contact or validation of the interpretation of the requirement before implementation. The iterations are often interrupted and sometimes broken off. The load of work products is furthermore high as FDA and the project requires documentation of the development process.

This study proves that it is in one sense possible for the agile and document-driven approach to co-exist; however, they are not easily integrated. We find that the agile process suffers from its adaptation to the FDA standard, the bureaucratic management of the project and the sequential work processes of the other project groups, which cause interruptions within the iterations, changes to the hardware and user interface that the software has to cooperate with. This implies that the software team experience trouble implementing a fully tested increment during the iterations and it leaves the software unstable. To improve the stability of the software iterations are sometimes cut off for a period. At the same time many of the genuine qualities of agile development have been maintained. First, due to the use of iterations the software team was able to create running software early in the project for the management and marketing to evaluate the device early. Secondly, the use of self organizing teams, where the developers choose their own tasks results in a high degree of ownership and commitment for each task. Third, the task board and the burn-down chart are useful for visualizing the tasks and the process for the developers. Fourth, the stand-up meetings give everyone an overview of the progress of the iteration and make sure that no one is stuck in a problem for too long.

It is left as an open question how a more integrated approach could be practiced. This study rather shows the limitations of a mixed approach. Further research would be needed to investigate what characterizes an integrated approach, which parts can be integrated and to what extent such an approach would still be compliant with the FDA requirements for software development.

REFERENCES

Aaen, I., "Software Process Improvement: Blueprints versus Recipes," *IEEE Software*, (20:5), 2003, pp. 86-93.

Abdeen, M. M., Kahl, W., & Maibaum, T., "FDA: Between Process & Product Evaluation," in *Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability*, IEEE Computer Society Press, Los Alamitos, 2007.

Beck, K. & Boehm, B., "Agility through discipline: A debate," *Computer*, (36:6), 2003, pp. 44–46.

Beck, K. & C. Andres, *Extreme programming explained: embrace change*, Addison-Wesley Professional, 2004.

Bygstad, B. & Nielsen, P. A., "The Meeting of Processes," in *Proceedings of the 26th IRIS*, Helsinki, 2003.

Bygstad, B., Nielsen, P. A., & Munkvold, B. E., "Four integration patterns: a socio-technical approach to integration in IS development projects," *Information Systems Journal*, (19), 2009.

Checkland, P. & J. Scholes, *Soft Systems Methodology in Action,* Wiley, Chichester, 1990.

Chrissis, M. B., M. Konrad, & S. Shrum, *CMMI: Guidelines for Process Integration and Product Improvement,* Addison Wesley, 2003.

Cockburn, A., *Agile software development: software through people,* Addison-Wesley Professional, 2002.

FDA., (2002). Guidance for Industry, FDA Reviewers and Compliance on Off-The-Shelf Software Use in Medical Devices. Retrieved July 10, 2009, from http://www.fda.gov/MedicalDevices/DeviceRegulationandGuidance/GuidanceDocuments/ucm073778.htm

Freedman, D. P. & G. M. Weinberg, *Handbook of Walkthroughs, Inspections, and Technical Reviews,* Little, Brown and Company, Boston, 1982.

Fritzsche, M. & Keil, P., "Agile Methods and CMMI: Compatibility or Conflict?," *e-Informatica Software Engineering Journal*, (1:1), 2007.

Grudin, J., "Groupware and social dynamics: eight challenges for developers," *Communications of the ACM*, (37:1), 1994, pp. 92–105.

Jakobsen, C. R. & Johnson, K. A., "Mature Agile with a Twist of CMMI," in *Agile'08*, 2008.

Jones, M., Gomez, E., Mantineo, A., & Mortensen, U. K., "Introducing ECSS Software-Engineering Standards within ESA," *ESA bulletin)*, 2002, pp. 132–139.

Knight, J. C., "Safety Critical Systems: Challenges and Directions," in *ICSE'02*, 2002.

Kotonya, G. & I. Sommerville, *Requirements Engineering: Processes and Techniques,* Wiley, Chichester, 1998.

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

McMichael, B. & Lombardi, M., "ISO 9001 and Agile Development," in *Agile 2007*, IEEE, 2007.

Mutafelija, B. & H. Stromberg, *Systematic Process Improvement Using ISO 9001:2000 and CMMI,* Artech House, 2003.

Opelt, K. & Beeson, T., "Agile Teams Require Agile QA: How to make it work - an experience report," in *Agile'08*, 2008.

Rakitin, R., "Coping with defective software in medical devices," *IEEE Computer*, (39:4), 2006, pp. 40-45.

Rottier, P. A. & Rodrigues, V., "Agile Development in a Medical Device Company," in *Agile'08*, 2008.

Sawyer, P., Sommerville, I., & Viller, S., "Capturing the benefits of requirements engineering," *IEEE Software*, (16:2), 1999, pp. 78-85.

Schrenker, R. A., "Software engineering for future healthcare and clinical systems," *IEEE Computer*, (39:4), 2006, pp. 26-32.

Schwaber, K. & M. Beedle, *Agile software development with Scrum,* Prentice Hall PTR, Upper Saddle River, NJ, 2001.

Sutherland, J., Jakobsen, C. R., & Johnson, K., "Scrum and CMMI Level 5: The Magic Potion for Code Warriors," in *AGILE 2007*, 2007.

Theunissen, W. H. M., Kourie, D. G., & Watson, B. W., "Standards and Agile Software Development," in *SAICSIT 2003*, 2003.

Turner, R. & Jain, A., "Agile Meets CMMI: Culture Clash or Common Cause?," in *XP/Agile Universe 2002*, D. Wells & L. Williams, editors, Springer, Berlin, 2002.

Walsham, G., "Interpretive Case Studies in IS research: Nature and method," *European Journal of Information Systems*, (4), 1995, pp. 74-81.

Weinberg, G. M., *Quality Software Management,* Dorset House Publishing, New York, 1992.

COPYRIGHT