Complex software project development: agile methods adoption

Deepti Mishra∗,† and Alok Mishra

Department of Computer Engineering, Atilim University, Ankara, Turkey

SUMMARY
The Agile Software Development paradigm has become increasingly popular in the last few years, since it claims lower costs, better productivity, better quality and better business satisfaction. Supply chain management (SCM) is a complex software development project. Owing to its scope and uncertain, complex and unstable requirements, it is not possible to develop it with predictable software development process models. Agile methodologies are targeted toward such kinds of problems that involve change and uncertainty, and are adaptive rather than predictive. How an agile process is introduced will significantly impact the implementation success of the process change. The objective of this paper is to analyze the agile development methodologies and management approach used in developing a complex software project. This further demonstrates how to overcome risks and barriers in each development phase of such complex inventive software projects. It also provides a set of guidelines regarding how the agile methodologies can be adopted, combined and used in these kinds of complex software projects. These findings have implications for software engineers and managers developing software by agile methods. Copyright © 2011 John Wiley & Sons, Ltd.

Received 24 December 2009; Revised 25 July 2010; Accepted 12 October 2010

KEY WORDS: agile methods; dynamic systems development method (DSDM); feature-driven development (FDD); Scrum; eXtreme programming (XP); adaptive development; supply chain management (SCM); Agile software development

1. INTRODUCTION
Software development is a series of resource-limited, goal-directed cooperative games of invention and communication [1]. Agile methodologies are a recent approach to software development, introduced at the end of nineties and now widely accepted worldwide as ‘mainstream’ software engineering. Agile methodologies are a recent approach to software development, introduced at the end of nineties and now widely accepted worldwide as ‘mainstream’ software engineering. Agile methods offer a viable solution when the software to be developed has fuzzy or changing requirements, being able to cope with changing requirements throughout the life cycle of a project [2]. Agile methods have proved to have a far higher agility and flexibility than the traditional software development [3] and are used to produce higher quality software in a shorter period of time [4]. Adoption of agile software development methods enables a software developer to be more flexible and responsive to the changing environments and customer demands [5].

In this approach, both developers and customers frequently adjust their strategies and actions by directly monitoring the feedback that results from their decisions [6]. Extreme Programming (XP) is the most popular agile methodology and is based on a series of concepts that include: having the business customer on-site, pair programming, collective code ownership, continuous code integration, small releases, designing test before writing code, standup meetings, refactoring and 40-h work weeks [7, 8]. In general, agile processes value the code production more than

∗Correspondence to: Deepti Mishra, Department of Computer Engineering, Atilim University, Ankara, Turkey.
†E-mail: deepti@atilim.edu.tr

Copyright © 2011 John Wiley & Sons, Ltd.
plan-driven processes. In a plan-driven process, developers might treat Unified Modeling Language designs and other non-code items as first-class artifacts. In the agile process, however, these items usually exist only to support the coding activity [9].

While agile methods are effective in some contexts, large and complex software products often require systematic discipline with the required additional process to ensure success. Agile planning is a relatively informal process with many small tasks to ensure optimal delivery results [2]. Applicability of agile methods to large organizations is often considered challenging [10–12]. In large-scale projects, the problem is amplified as the complexity of the application domain is often beyond the experience or expertise of a small number of customers as well as developers. There is a clear need for continuous customer engagement in large-scale complex projects and it is the key actor for XP project success [13]. Currently, organizations are increasingly deploying agile methodologies in their software development projects. However, a systematic and large-scale adoption of agile practices in organizations is still elusive and is mainly focused on project level activities [14]. When suitably adapted for use in projects with complex domains or limited resources, it has been found that XP offers a high degree of security and reliability without limiting the advantages of agile software development [15, 16].

The objective of the study is to illustrate how Agile methods can be adopted in a complex software project development. The remainder of the paper is organized as follows: In Section 2, agile method adoption and related issues in large and complex projects are presented. In Section 3, the project background, requirement analysis, project management, architectural design and project implementation of a case study are illustrated. Section 4 summarizes the discussion and lessons learned. Finally, the paper concludes with a conclusion and the future research directions in this context.

2. RELATED WORK

Large-scale complex projects face dynamic changes in requirements and time-to-market pressure. To mitigate the risks in time-to-market and changing requirements, efforts have been made to adopt the XP in large-scale projects, with mixed results [13].

2.1. Advantages of using Agile methods in large and complex projects

Reifer [17] observed that most experts agree that agile methodology and traditional approaches are philosophically compatible. Paulk [18] mapped XP practices to SW-CMM model that is usually considered appropriate for large-scale projects. Further efforts have been made to tailor the XP methodology for large complex projects to accomplish faster development cycle times [17, 19]. Through an industrial case study, Peterson and Wohlin [20] found that the main advantages of implementing agile and incremental practices in large-scale software development can be listed as follows: (1) requirements are more precise due to reduced scope and thus easier to estimate, (2) direct communication in teams reduces the need for documentation, (3) early feedback due to frequent deliveries, (4) rework reduction, (5) testing resources are used more efficiently, (6) higher transparency of who is responsible for what creates incentives to deliver higher quality, (7) low requirements volatility in projects and (8) reduction of waste (discarded requirements) in the requirements engineering process. Agile practices such as evolutionary requirements, pair-programming and direct stakeholder involvement have been proven to be successful for the development of large-scale systems [21, 22]. Few studies reported that some practices from agile methodologies such as iterative development, frequent testing and feedback, small release and refactoring are suitable for large projects, while others like stand-meetings and the use of metaphors are not appropriate [17, 19].

2.2. Challenges faced while using Agile methods in large and complex projects

Peterson and Wohlin [20] found that implementing agile and incremental practices in large-scale software development leads to benefits in one part of the process, while raising issues in another
part of the process. For example, using small and coherent teams increases control over the project, but leads to new issues at the management level where coordination of the projects has to take place [20]. They also stated the main challenges while using agile and incremental practices in large-scale software development as (1) challenges in regard to realize continuous testing, (2) increased maintenance effort with an increase in the number of releases, (3) management overhead due to the need for coordination between teams, (4) detailed dependencies are not discovered on a detailed level due to lack of focus on design, (5) long requirements engineering duration, due to complex decision processes in requirements engineering, (6) requirements priority lists are hard to create and maintain, (7) waiting times in the process, specifically in design waiting for requirements, (8) reduction of test coverage due to shortage of projects and lack of independent testing, (9) increased configuration management effort.

2.3. Relevance of architecture design while using Agile methods in large and complex projects

As software complexities increase, so do the relevance and significance of software architecture and its related documents. Soundararajan and Arthur [21] further argued that larger scale systems involve lengthy development cycles and due to personnel turnover, also demand comprehensive documentation for support and training. Comprehensive documentation is often necessary when a large-scale system is under consideration in order to provide the third party maintenance organizations with documented information [21]. Agile methods cannot be adopted directly for large, complex projects due to the lack of upfront design and documentation [23]. However, for complex, large-scale projects upfront architectural design is considered to be essential as it provides a strong backbone that could support several services built on top of it [13]. Architectural design becomes even more important for complex large-scale applications because developers and analysts miss the connections and dependencies between stories as the application size and complexity grow [24]. It is found that without an overall design, it is difficult to maintain a big picture of the project as the system grows [13]. Another option could be having a design that evolves with the process instead of a complete upfront backbone design [25].

2.4. Balancing formal and informal communication while adopting Agile methods in large and complex projects

Agile methodologies lack upfront design and investment in the life cycle architecture, and rely primarily on the tacit knowledge of individuals and informal communication which can cause high risks for instance, a team may make irrecoverable architectural mistakes due to lack of an appropriate design [13]. Cao et al. [13] further suggested that informal communication may not be effective when dealing with a large number of stakeholders and vast amounts of information that are characteristic of large projects. However, there is growing recognition that lack of agility makes it difficult to make changes to accommodate evolving requirements. Also, solely relying on formal communication causes problems in understanding and communication within the development team [13]. Soundararajan and Arthur [21] suggested the soft-structured framework (hybrid approach) for a larger scale system development incorporating the desirable benefits of agility like accommodating the changes to requirements even later in the development life cycle. The development of larger scale systems requires a structured approach and therefore, combining the advantages of agile practices and structured methods can be an effective solution to accommodating change in larger scale systems while ensuring that in this hybrid approach the impact on agility should be minimal [21]. There will be some overhead in maintaining the architecture in agile, and a minimal amount of documentation is necessary to help developers to do their work [26]. Sometimes, it is significantly more productive for a developer to draw some bubbles and lines than to simply start hacking out code [27]. Upfront architectural design reduces the development time for new functionalities [13]. Each new functionality is based on the infrastructure backbone and design patterns [28] which can reduce the time and effort in a substantial way for implementing new functionalities [13].
2.5. Coordination of multiple teams while scaling Agile methods in large and complex projects

Moore and Spens [29] found that in large-scale projects, team members must participate in cross-team activities and allocate time to project-wide activities. In particular, technical team leaders are required to dedicate significant portions of their time outside the team room for cross-team communication [29]. This cross-team communication is important because of the dependencies that teams have with each other while developing large software systems and because the teams need input for their decision making and need to be synchronized with other activities [30]. Grewal and Maurer [31] also used this approach while scaling agile techniques to a large-scale project where team leaders shared the progress and issues of their teams with each other.

Rodriguez et al. [32] recently observed that there are still few studies about how real-life agile projects identify and manage customer needs. The previous research does not offer much help in scaling up agile methods [30]. Few research papers exist on this subject, and agile methods in general lack proper research [33, 34]. Furthermore, very few real-life software projects development by agile methods from process perspective is documented in the literature. According to Laanti [30], every company or organization needs to find out its own adaptation based on their own business, operative model and the needs. Therefore, the objective of this work is to analyze the agile methods and the management approach used in the development of complex software projects. This experience report presents an industrial case study that shows successfully integrated practices from the agile software development view and how to overcome risks and limitations in each development phase of a complex and inventive project. It also provides a set of lessons learned about how agile methods may be adopted, combined and used in these kinds of projects.

Generally, the case study method is a preferred strategy when ‘how’ and ‘why’ questions are being posed, and the researcher has little control over events [35]. The case study method, a qualitative and descriptive research method, looks intensely at an individual or a small group of participants, drawing conclusions only about the participants or group and only in the specific context [35]. The case study method is an ideal methodology when a holistic, in-depth investigation is required [36].

3. CASE STUDY

The organization is a small and medium enterprise (SME) situated in technology park and is developing software for different domains by using agile methods for the past six years. This company engages in the design, development and implementation of software for various applications/information systems in the national and international markets.

The success of the supply chain management (SCM) software project was based on starting with agile methods and achieving optimal processes by customizing them according to vision and benefiting from adaptivity. Since there is no systematic way to apply agile methods, the project was benefited from an unsystematic approach that involved applying condition-specific processes during development. The main activities that were performed during SCM project are shown in Table I.

3.1. Project background

When the decision was taken to develop this complex software project, the market analysis was performed within the company. This software was not intended for a specific customer. It was developed to be marketed. A team that included a marketing expert, a manager and a domain analyst interacted with many customers in order to define the potential customer sectors and the required services and functionalities. This market analysis was actually based on optimization requirements since it is the most important functionality that SCM should provide. After a couple of weeks, an abstract scope of the product was defined. The base distinction that needed to be determined was whether the project was predictable or inventive [38, 39]. In general, the development process, management values, planning and estimation models appropriately associated
Table I. Major project activities performed along with Agile methods used.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Agile methods used</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scoping and requirement analysis</td>
<td>Dynamic Systems Development Method (DSDM) and eXtreme Programming (XP)</td>
<td>The fundamental idea behind DSDM is that instead of fixing the amount of functionality in a product, and then adjusting time and resources to reach that functionality, it is preferred to fix time and resources, and then adjust the amount of functionality accordingly. Also, the roles defined in that approach were combined and adapted, according to the team structure of the company. Because of its simplicity and being more test-driven, allowing close collaboration and communication, XP was also a part of this method. The methods that were used in SCM during this study were hybrid and adapted implementation of the above mentioned methods.</td>
</tr>
<tr>
<td>Architectural design</td>
<td>Feature-driven Development (FDD)</td>
<td>Feature-driven Development (FDD), an agile software development approach based on the iterative and incremental development, was used during development of that structure. The FDD approach does not cover the entire software development process, but rather focuses on the design and building phases [37]. However, it has been designed to work with other activities of a software development project and does not require any specific process model to be used.</td>
</tr>
<tr>
<td>Implementation</td>
<td>Feature-driven Development (FDD) and Extreme programming (XP)</td>
<td>This was done using Agile development methods—FDD and XP—in an adaptive and hybrid manner. The main emphasis was on delivering the features as opposed to builds or just code because features imply a sense of completion.</td>
</tr>
<tr>
<td>Project management</td>
<td>Scrum and XP</td>
<td>The project management approach used in the complex software project was mainly based on the hybrid usage of Scrum and XP that are two types of agile software development. The framework of Scrum activities, XP’s feedback and communication were the concepts that were used for the management processes.</td>
</tr>
</tbody>
</table>

with these two domains are different. So, the product was analyzed with the following domain characteristics:

- The scale of the project was large.
- Project complexity was high.
- Acquaintance with the domain was less.
- Initially there was insufficient requirement specification.
- Requirement volatility was high.
- There was a variety of customers.
- Quick release was important to have an edge in the market.
- There were multiple development teams and each team size was small. These teams concurrently developed different parts of the SCM software.
- Getting near the start, the reliable estimate of effort and cost was difficult.

Therefore, it would have been a wrong decision to choose one of the traditional approaches (predictive methods) (i.e., waterfall methodology etc.) that are used for more predictable kinds of projects. As the evidence shows, this project was an inventive project and it had to be developed within the motivation of agile and iterative methods [39]. These methods can provide control over unpredictability by benefits of adaptivity. In order to control unpredictability, the key is iterative and incremental development as well as adaptive development. Within that self-adaptivity approach, each team member or sub-team selected their process according to the characteristics of the module they were developing but tuned their processes to the whole project. Developing with an agile and iterative development process opens the door to the possibility of smaller and more frequent releases. Two primary benefits of this change are: increased responsiveness and reduced risk. If responsiveness increases, then, the newly discovered customer’s needs could be addressed.
in a much shorter timeframe, most probably with a demonstration, since there was a running product although with some core functionalities. The primary mechanism that allows a team to steer toward its release goals was demonstrating working software early and often to product owners, customers and hopefully to end users [40]. Thus, every iteration was an opportunity for the team to get feedback and guidance from customers about how to make the system deliver on the release date, something of vital importance for the customer. It was better than presenting the product only by discussions. Adaptive development is an important concept to reduce the high-risk of a new product, provides ability to customize the product for each customer and increases the responsiveness of production.

3.2. Requirements analysis

Requirements Engineering (RE) activities are considered critical to any software development process. It has been recognized that problems associated with requirements area are among the major reasons for software project failures [41, 42]. The effort to explore and refine RE has increased in the previous years, as observed in their studies about the current and the future of RE [43, 44]. However, there are still few studies about how real agile projects identify and manage customer needs [32].

In this project (as shown in Figure 1), initially, information was collected from many potential customers and then workshops were organized to define a vision and scope, and to identify functions and features at a high level (such as just the names of use cases and features). All information defined in these sessions was collected within a requirement repository. At any point in time, a large number of ‘could do’, ‘should do’ and ‘must do’ requirements were collected. These had to be aggregated in a centralized repository where they could be viewed, prioritized and ‘mined’ for future iterations. Setting priorities early in the project helped in deciding which features to skip under time pressure. Requirements prioritization should be done by the customer. Both the customer and the developer had to provide input to requirements prioritization. The customer marked features providing the greatest benefit to users with the highest priority. Developers point out the technical risks, costs or difficulties [46]. Requirements had to be readily accessible to all team members, available to be enhanced and revised over time, and remain reasonably
current to directly drive testing as they are implemented. It was observed that the product would consist of:

- a core part (solver and many heuristics for the optimization),
- a support part that includes:
  - GIS (Geographical Information System) facility,
  - storage of data (should be able to integrate with an ERP system),
  - reports,
  - GUIs.

Research for each facility and the domain analysis was performed as separately scheduled parallel tasks and each one was assigned to a different team member. A test-driven approach method was used in order to gain knowledge and decide whether it should be developed from scratch or by integrating a pre-developed library or product.

(a) The solver and heuristics was a more domain specific and critical work. There were many performance-related constraints such as response time, accuracy, flexibility, etc. The data transition part was also one of the important parts because of the huge amount of data to be processed within an acceptable time. The process started with the preparation of test data, test cases and test environment (hardware and software), in order to evaluate the existing products (open-source as well as commercial) according to functionality and defined constraints. As the team gained more knowledge about the domain while testing was in progress, it was realized that some customized solutions needed to be implemented for some customers. This reason along with performance-related constraints helped in making the decision to develop from scratch all the optimization parts including solver.

(b) For the support part, the existing commercial and open-source products were evaluated according to the listed functionalities by the business expert. Finally, it was decided that it would be more appropriate to integrate the existing open-source libraries, and not to develop a new one. The mandatory functionalities were provided by these products, and the other functionalities would be developed by team members. Requirements for the additional functionalities which were known were defined and placed into the requirements repository.

3.3. Agile project management

Among the most known and adopted agile methods are Extreme Programming (XP) [47] and Scrum [48]. XP focuses primarily on the implementation of software, while Scrum focuses on agile project management [34]. These have received praise from practitioners because of their abilities to deal with volatile requirements [49].

Rodriguez et al. [32] found in their project experience that Product Backlog management is a complex task in agile methodologies like SCRUM. The Scrum Backlog and progress tracking approaches are minor variations of XP practices, and they are so simple that they are well within the XP spirit of ‘do the simplest thing that could possibly work’. Instead of using Scrum’s 30-day timeboxed iteration length, Scrum’s timeboxing concept was used but with XP adaptation, as XP prefers shorter—one or two weeks—iterations. The reasons were the inventive type of the project, unpredictable requirements and the need for customization for each customer. Also it was important to release the product (may be a core or demo version) as soon as possible to get an edge in the market. It was not possible to make predictive planning using structural methods. In order to overcome high-risk; plans, estimations, schedules, task assignments and measurements related to the management values had to be made within small time intervals (iterations). Later, at the end of these intervals feedback had to be analyzed in order to plan successive iterations.

The approach used for the SCM software project to make plans with development teams is called collaborative planning [7]. The initial planning meeting and each iteration planning meeting was
held between the project manager, the business expert and a representative from each development team. The initial meeting involved the planning of the scope analysis of GIS, solver and heuristics and ERP inclusion in the system. During each iteration planning meeting, developers estimated the work they were responsible for. The planning method used in the development of this project was based on the concept of rolling wave planning which is a refinement of adaptive planning [7]. It implies that there is no detailed plan for the whole project development, except for iterations. There was no fixed plan of how many iterations there will be, how long they are going to last, or what would happen in each. But there were milestones with dates defined in the development; the path of iterations to achieve those milestones is left flexible or adaptive. It was better than trying to plan a detailed speculative schedule for an unpredictable project. Such adaptive planning is the key idea in agile methods. A tentative plan was made in accordance with the existing circumstances, but, again, not for the whole life cycle of the project. The requirements functionalities were defined for the baseline. The primary plan in detail for each iteration was made in order to achieve the baseline. During iteration planning meeting, goals that would be accomplished within the period of a week were listed. The feedback from previous iterations was used to plan the current iteration. Goals were separated into one-week accomplishable tasks according to estimations done by team members who were responsible for the tasks. It means that each iteration time was fixed and tasks were divided into subtasks that could be accomplished within that fixed time period. During these meetings, technology, program structure and the overall system functionalities were also discussed according to the information collected from many potential customers. An iteration planning was based on the existing emergent requirements, the presence of resources and the degree of knowledge of the current iteration. Therefore, planning was closer to optimal in terms of working toward milestones; each step was performed in the most skillful manner to plan regarding risk, productivity and effectiveness because each planning step is taken with the maximum and latest information. The milestones were used for each internal release due date and there were many iterations planned within that period time.

3.4. Architectural design

Preliminary architectural design was also done using the initial requirements. It is supported by Mead [50] that architecture modeling and trade studies are important activities that should take place during requirements engineering activities, not after requirements have been defined. Software architecture must be considered during requirements engineering to ensure that the requirements are valid, consistent, complete, feasible etc. [50]. There were many development teams working concurrently on different parts of the SCM software. To avoid any confusion between these teams and also to have a common picture of what they were developing, a team consisting of a project manager, development team representatives and a business expert made up the core architecture of the system. This was the structure that specifies the whole system as major components, modules; collaborations, interactions and interfaces between them, plus the responsibility of each module. All the defined modules and components were drawn as black boxes and the interactions between them were represented by arrows. The development of each module was assigned to different teams as parallel tasks. The responsibilities and collaborations were defined clearly for each module (i.e., Controller, IO manager) and sites (DBMS, GIS, Application Server). This structure was also allowed to change as a result of customer’s feedback from future iterations. Since it was a basic structure and there was collective ownership on that part by the team members, it was important to document that structure in order for it to be accessed easily. The diagrams, responsibilities, functionalities and scenarios were recorded through the documents. The main technical result of the design phase is the software architecture, consisting of both dynamic and static views of the software. A major advantage of object-oriented (OO) notations is that the same description may cover the whole life cycle and is refined and itemized in every phase [51]. It further increases the applicability of the iterative and incremental development process because OO design provides modularity, minimum coupling and maximum cohesion, thus a flexible structure. Another benefit of using OO techniques was to define the parallel tasks; since all modules provide encapsulation and a loosely coupled structure, each could
be developed independently as a sub-product and then be integrated easily because of well-defined interfaces. The OO architecture style was preferred as it provides formalisms and ease to document with UML diagrams. Wherever required quality estimates, based on views and architectural patterns were derived as quality assurance methods which are most effective when capturing the requirements.

Once the core was built, team leaders returned to their respective teams and the development was done in parallel with multiple teams by using short iterations. Each team leader had a clearer picture and a common vision due to the architectural design and could better convey and maintain that for the rest of the project. Furthermore, each acted as a liaison to other teams. Also, after spending some time with the other team leaders, there was improved communication between them.

Each defined module (such as Computational, IO managers, GUIs, GIS, Reporting) can use different development processes but every module should be synchronized to the architecture because it defines the overall system.

3.5. Project implementation with Agile methods

The significantly different incremental, intertwined approach to requirements development is one of the flagship components of Extreme Programming [52]. In this context, the requirements are recorded as a set of user stories developed jointly by customer representatives and the development team. In keeping with the lightweight approach to software development, teams adopting Extreme Programming are advised to use the traditional index card (story card) to record these user stories [52]. User stories represent product needs that are defined and implemented in reduced time slot. Agile Teams manage a high number of user stories that increase during the development. Non-functional requirements may be associated with many user stories [32]. The programming language used during development was pure ‘Java’ based on a full OO structure. Most Agile software development literature cites its use for application development projects, often implemented in OO languages [38].

FDD consists of a set of ‘best practices’ and the developers of the method claim that even though the selected practices are not new, the specific blends of these ingredients make the five FDD processes unique for each case. All practices available should be used to get the most benefit of the method as no single practice dominates the whole process.

The XP Values are Communication, Simplicity, Feedback and Courage. The essence of XP is simple: Be together with your customer and fellow programmers, and talk to each other. Use simple design and programming practices, and simple methods of planning, tracking and reporting. Test your program and your practices, using feedback to steer the project. Working together in this way gives the team courage [53]. XP aims at enabling successful software development despite vague or constantly changing requirements in small- to medium-sized teams. Short iterations with small releases and rapid feedback, customer participation, communication and coordination, continuous integration and testing, collective ownership of the code, limited documentation and pair programming are among the main characteristics of XP. One of the fundamental ideas of XP is that there is no process that fits every project as such, but rather, practices should be tailored to suit the needs of individual projects.

3.5.1. Development of the core part. For each iteration (new functionality, defect fix, changing) during the development of the core part (solver and many heuristics for the optimization), the requirements were selected from the repository according to their priority and defined the functionality for that iteration. Their use-cases and working scenarios were prepared by the domain expert and supplied to the development team. Then, within the defined scope and process for the overall structure, the design, planning and implementation was done for each module that would be developed in order to make the increment for the system. When each module in a particular increment was developed and integrated to the overall system, then the testing of the overall system was performed and the result was validated. Later, the product was released (end of the release or end of the iteration).
3.5.2. Development of the support part. The other parts of the system included in the support category were GUIs, GIS, reports and storage of data. As mentioned before, these parts of the system were used as a layer to interact with the users of the system.

3.5.2.1. Development of the GIS component. For the GIS component, an open-source library was chosen to be integrated and used in this product. The main functionalities were provided by the library; all other required functionalities and services that were planned were provided by our product. This part was also done with a test-driven approach and the early development approach used in the phases of scope, vision and requirements analysis.

3.5.2.2. Development of data storage part. Users can export or import their real data from or into this system through storage. These activities may be done offline using the corresponding modules and user interfaces, visual services provided by the system itself or online, through the integration of the storage part of the system to the storage mechanism of the customers, which may be a database, an existing system or an ERP system. These considerations were discovered in the early phases of the scope, vision and requirements analysis. This part was defined as a black box in the overall system design but for defining interfaces, a test-driven and early development approach was used at early phases of the scope, vision and requirements analysis. The ERP products were analyzed and tested, their results and possible requirements of database and availability of existing systems were discussed in meetings that included development team members, a business expert and the project manager. The vague or conflicted suggestions were resolved by defining the test cases and test data. Using the early development approach, a prototype or demo version was developed and tested. In these studies, knowledge was gained about the development of storage part, possible integration of that part to other systems or ERP systems with their tested interfaces, usage of the databases, functional and non-functional requirements, constraints and the scope of this part was defined. The actual development and detailed plan of that part was left to the phase of delivering the final product to the customer because, as mentioned before, this was a new product and it was not developed for a specific customer. But an initial design and development of the storage part was performed with a flexible, adaptable and portable structure in order to reduce the delivery risk. As the team had already gained knowledge and experience in the early development and testing of that part and a flexible and adaptable architecture and interfaces were developed using the agile development approach, the delivery and acceptance of customers was achieved successfully.

3.5.2.3. Development of GUIs. The development of GUIs and their integration to the system was done by a team that included highly skilled persons of that subject. The requirements and scenarios of usage were defined by the business expert and development team. The required GUIs and their functionalities were defined for each iteration and release in corresponding meetings in an abstract way. According to the plan defined for each iteration and release of the overall project, meetings were performed before each iteration or release to define everything related with GUIs between the team members that were responsible for the development of GUIs and the business expert. The iterations of GUIs were planned according to development of selected GUIs which were prioritized and already defined in the iteration plan of the overall project. In order to develop the GUIs, first the business expert drew the GUIs using a design tool, and also documented the usage scenario of these GUIs. Also, the information about how and in which part of the system the data would be gathered from these GUIs, would be used. Later, a development team prepared the empty forms of those GUIs just satisfying visual requirements using all the documentation. After the validation process by unit and integration testing of those empty forms, they were integrated according to the usage scenario and their data bindings were done according to the defined data flow into the already built structure of the overall system. The validation of those parts was done with integration tests by the development team and the business expert. Agile approaches use frequent review meetings and acceptance tests for requirements validation. Review meetings show
that the project is in target and schedule increasing customer trust and confidence in the team or highlighting problems at an early stage [46].

3.5.2.4. Development of the report part. The report component part was also based on an open-source library. Since most of the reports were already defined within a business domain, the main selection criteria were technology that was used within the project, adaptability, flexibility, portability and usability of the candidate libraries.

3.5.3. Development of demo versions. The other agile development method used in the development of the SCM software project just for demo versions, had different characteristics. During the walkthroughs with the customers, the customers requested some demonstration of customized solutions. Whenever such requests existed, all release and iteration plans were updated or new plans were prepared. After it was decided to make such a demonstration, the product backlog of that new release was prepared and backlog items were prioritized by the business expert and product owner. The planning of that emergent release was performed within a release meeting that might not be a regular meeting. If the release backlog items fit or are already included in the existing release (current release under development), then the existing release plan had been updated for the successive iterations, but if it did not, then the existing release plan was changed or redirected to an emergent demonstration release. The method used for the emergent development was different from the existing one, which had shorter timeboxes, more customized solutions and more visible development. During the meeting of this emergent release plan, all tasks were defined clearly, listed on a white board and the estimation of each task was done by the module developers. A schedule-driven approach was used during the iteration planning because the time was fixed according to demonstration date and all definition of backlog items and their priorities were defined according to that approach. The defined tasks and the assigned team member’s name were written on cardboards and hung up on walls. The team member responsible for a particular task was required to update those cardboards as the task progressed. So the progress of each task could be seen through those cardboards whether it was finished, tested, committed or should be validated. If some tasks were dependent on each other, the team member could follow the progress and update their individual plans without interrupting other team members. This is a combined and customized approach of agile methods and a very useful approach for such emergent releases. This approach increases the visibility of development and the awareness of how progress is going on. It also increases the morale and focus of the development team members.

4. DISCUSSION AND LESSONS LEARNED

Nerur et al. [54] suggested that migrating to agile methodologies involved many issues pertaining to management, people, process and technology. Agile methods have been demonstrated to enhance customer value, improve organizational morale and provide step improvement in product quality [9]. Following are summarized implications (lessons learned) for software engineers and managers adopting agile software development methods:

(a) The project management approach used in this project was mainly based on the hybrid usage of Scrum and XP. Scrum and extreme programming provide complementary practices and rules [55]. Scrum is used to manage the steps taken to develop software in conjunction with the use of XP to ensure the quality of the software [55]. In this project, instead of using Scrum’s 30 days iteration, XP’s 1 or 2 week iterations were used to get customers’ feedback frequently.

(b) As the product was complex, large-scale and development was done with multiple teams, it was necessary to have the right balance between control (discipline) and flexibility. To ensure the discipline necessary for timely progress in a large project and proper coordination between multiple teams, it was not possible to involve every development team member during the iteration planning activity. Although Agile project planning is collaborative in
nature and the whole team is responsible for planning, it is adapted according to the characteristics of the project. Each iteration was planned in an iteration planning meeting which was attended only by the project manager, the business expert and a representative from each development team to list the goals that would be accomplished within the period of a week. The feedback from previous iterations was also used to plan the current iteration.

(c) Architectural design, which was done by the requirement engineering team, helped in making a full and clearer picture of the entire system among all different development teams working on different parts of the system. Read and Maurer [26] supported this and stated that one way to scale agile methods is via an architecture-centric approach, in which a project is divided into smaller modules on which sub-teams can use agile effectively. They also suggested that by combining lightweight planning with an architecture-centric design strategy will be most beneficial without compromising the spirit of the practices of agile methods. In fact, agile projects normally have some overhead when user stories are gathered and prioritized, development tools chosen, environments configured, etc. Defining the system architecture can be included as one of such startup costs, if the architecture is defined in a quick, lightweight manner that is flexible to change [26].

(d) As this was a large-scale project and there were multiple teams working on the different parts of the software, it was necessary to have some means for cross-team communication and synchronization as supported by Moore and Spens [29]. In this particular project, regular meeting of different team leaders and business experts helped in cross-team communication. They informed each other of the progress of their teams as well as unresolved issues. These unresolved issues sometimes necessitated other team leaders to adjust their plans. According to Laanti [30], Grewal and Maurer [31], scaling agile techniques to a large-scale project requires team leaders to regularly share the progress and issues of their teams with each other.

(e) The presence of a business expert is also important for the success of the project. In this project, a development team member, who had worked in the past in a similar kind of project, played the role of business expert. Since this person was not entirely from this particular domain, relying solely on his knowledge was not possible and that is why the involvement of some prospective customers was sought, but this business expert played a very crucial role in resolving the conflicts among requirements and also prioritizing these requirements. Also, when the customers could not be present, this member filled that space which is thought to be significant for the market-driven project.

(f) The business expert also played another important role. Requirement repository contained requirements defined at a high level (such as just the names of use cases and features). For each iteration, the requirements were selected from the repository; their use-cases and working scenarios were prepared by the expert and supplied to the development team. In this way, the business expert bridged the gap between high-level feature description and the detailed requirements that are required and can be managed via the agile methods.

(g) Non-functional requirements management is one of the tasks that causes more problems in agile method adoption and the right solution still has not been found and requires further research [32]. For instance in this project such requirements and their quality issues were handled by lightweight architecture and views.

(h) In agile methods, customers (system users) work in small teams with developers as active team members. Customers and developers jointly determine the system features to be implemented in each cycle [56]. The involvement of some prospective customers is important for the success of the project. Such a drastic change in user role suggests that successful acceptance of agile methodology is concerned not only with software developers and organizations, but also with customers who are expected to be collaborative, authorized, committed and knowledgeable [57]. Although customers were available most of the time during the project, sometimes it posed challenges too. Sometimes their ideas were entirely different from each another. Furthermore, during a very crucial moment, they could not be present.
(i) A brainstorming meeting among the requirement engineering team played the role of filter before workshops. Issues which could be resolved without the help of all stakeholders were solved here and therefore saved a lot of time. Also, they helped in setting the agenda for the workshops [58].

5. CONCLUSION

Agile methodologies are targeted toward problems involving change and uncertainty, and are adaptive rather than predictive. These methods can control unpredictability by using the benefits of adaptivity. In order to control unpredictability, the key is the iterative and incremental development as well as the adaptive development. Companies can become more demand-driven and value-driven by embracing agility and benefit from agility over a wider spectrum. Agile methods are used to produce higher quality software in a shorter period of time. As the presented SCM software project was an innovative project, key practices of agile methods such as scheduling according to feature priorities, incremental delivery of software, feedback from expert users, emphasis on face-to-face communication, pair development, minimalist design combined with refactoring, test-driven development, daily integration, self-organizing teams and periodic tuning of methods helped significantly to achieve its successful implementation. As agile methods provide flexibility, it encourages the development teams and individuals toward creativity which is essential for successful implementation of innovative projects.

To accommodate evolving requirements lightweight methodologies like Agile are becoming popular. However, these should be tailored (customized) for large-scale complex projects. Creating a stable architectural design upfront is recognized as the striking difference between the agile practices for large-scale projects and agile principles [13]. Agile development gives the architect repeated opportunities to work closely with the business and technical teams to continually guide systems in the direction of good architecture and a skilled architect can adapt to agile development while staying focused on the core architectural work [59]. As it was an innovative, large-scale, high-risk project, the architectural design along with documentation was formally done. This design documentation played an important role in the successful implementation of this project and it will also be helpful in the maintenance phase. The most important characteristic of the development methods used in this project is that they were adapted to the circumstances in each phase of the development. Agile development methods were combined so that new approaches resulted from this self-adaptivity approach. It was not possible to complete and fix all the requirements because of the business domain and product characteristics. The software development team handling such a large project was small. Communication between team members was strong, as they were working in a small office and a business expert who had knowledge of the business domain was in the same office so that they could interact whenever required. The development approaches used in the SCM software project involved less documentation than the process-oriented approaches, usually emphasizing a smaller amount of documentation for a given task or only the critical parts were documented. As a future work, it would be interesting to study and compare other cases of complex software project development by agile methods in order to increase the understanding regarding how to address challenges and management issues in agile software development projects.

REFERENCES


AUTHORS’ BIOGRAPHIES

Dr. Deepti Mishra is an Assistant Professor of Computer Engineering at Atılım University, Ankara, Turkey. She has earned her PhD in Computer Science (Software Engineering) and her Masters in Computer Science and Applications. Her research interests include software process improvement, software quality, requirement engineering and databases. She has published many research papers and book chapters at international and national levels. She is the recipient of the department of information technology scholarship of the Government of India.
**Dr. Alok Mishra** is an Associate Professor of Computer and Software Engineering at Atilim University, Ankara, Turkey. He holds a PhD in Computer Science-Software Engineering in addition to dual master’s degrees in Computer Science and Applications and Human Resource Management. His areas of interest and research are software engineering, information system, information and knowledge management and object-oriented analysis and design. He has published articles, book chapters and book-reviews related to software engineering and information system in refereed journals, books and conferences. Dr. Mishra is also the editorial board member of many journals including *Computer Standards and Interfaces, Journal of Universal Computer Science, Computing & Informatics, Electronic Government—an International Journal* etc. Dr. Mishra has extensive experience in distance education related to computers and management courses. He has received the excellence in online education award by U21Global Singapore. He had also served as the chief examiner of computer science of the International Baccalaureate (IB) organization. He is the recipient of various scholarships including the national merit scholarship and the department of information technology scholarship of the Government of India.