Challenges and Working Solutions in Agile Adaptation: Experiences from the Industry[1]

Özden Özcan-Top[1], Onur Demirors[2] and Fergal Mc Caffery[3,4]

[1] Graduate School of Informatics, Middle East Technical University, Ankara, Turkey
[2] Department of Computer Engineering, Izmir Institute of Technology, Izmir, Turkey Springer
[3]Regulated Software Research Centre, Dundalk Institute of Technology, Dundalk, Ireland
[4]STATSports Group, Newry, Ireland
ozdenoz@metu.edu.tr, onurdemirors@iyte.edu.tr, fergal.mccaffery@dkit.ie

Abstract. Challenges in agile adaptation is inevitable in software development projects and have to be dealt with by software practitioners. The pathway to excellence in agility requires experience of challenges, failure of process scenarios; and the discovery of working solutions by software development teams. The major purpose of this study is to highlight both the challenges organizations faced when implementing agile techniques and the solutions adopted that proved successful. In order to specify these challenges and working solutions, we performed a multiple case study by using the Software Agility Assessment Reference Model (AgilityMod). In this paper, we describe two cases that achieve the highest levels of agility among eight cases and describe their experiences in achieving a good adaptation through the challenges that they faced and the solutions that were found for these challenges. Additionally, we provide two challenges that have not been resolved yet and are subject to further discussions.

Keywords: Agile Software Development, Agility Assessment, Agile Adaptation, AgilityMod.

1 Introduction

Agile software development is one of the most important paradigms that radically changed how software is developed [1]. The agile manifesto and principles [2] inspired many people.

The promises made by the agile manifesto were so tempting that inevitably, agile was considered as a silver bullet. After a while, it was noticed by the software community that agile software development is not a "one size fits all" kind of approach. Every project, whether small or large, distributed or collocated has its own conditions that are specific to those environments. Besides, organizations do not quickly progress from low to high levels of agility. The pathway to excellence in agility requires experience

[1]Copyright ©2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

of challenges, failure of process scenarios; and the discovery of working solutions by software development teams.

Unresolved challenges may end up with the fossilization of software development teams, which is a phenomenon described as being stuck with a condition that prevents someone from improving. Mitigating the impact of such challenges had a significant role on team performance and breaking the domino effect created by such challenges [3].

It is also significant to identify to what degree a software development team compromises agility while discovering the solutions that work for them. Agile maturity and agility assessment models are utilized to identify upon which side of the agility line an organization resides. Previously we assessed the capability of existing models [4], but were not convinced with the quality of such models.

We developed the Software Agility Assessment Reference Model (AgilityMod) to provide a structured model for agility assessment. The Model provides a means for identifying agility gaps in software development projects [36].

We performed a multiple case study to identify AgilityMod's applicability in operational software development projects. The case study included eight industry cases. In this paper, we present the two cases (Case G and Case C), which achieved the best agility results. The main emphasis of this paper is upon the lessons learnt from the most successful implementations of agile and that is why we only focus on the two most successful cases.

The purpose of this paper is to describe both the challenges organizations faced when implementing agile techniques and the solutions adopted that proved successful. We think that it is also very important for the challenges to be described with their own environments where they were observed. From this perspective, the paper will provide a specific insight to readers about what could work, under what kind of conditions. The challenges described here are not at an abstract level, but actual, detailed and specific.

This paper is structured as follows: In Section 2, we discuss the challenges organizations face when embarking upon agile software development and suggested solutions for these challenges, based upon previous documented research. In Section 3, we briefly explain the Software Agility Assessment Reference Model (AgilityMod) that we developed and was subsequently used within the eight case studies. In section 4, we present the case study. In section 5, we both provide the challenges that were faced in agile adaptation and the specific solutions that were successfully implemented in two organizations. In Section 5, we also describe two challenges that still need to be further investigated for effective solutions. Finally, in Section 6, we present our overall opinion on this topic.

2 Background

Agile has had a groundbreaking impact on software development. Debates on what agile is still continue, as to whether agile is: a development and management philosophy; a collection of technical practices; a way of life; or all of these [1].

Nerur *et al.* specify that agile and traditional approaches diverge in a number of aspects: approach to control, management style, knowledge management, role of the customer in development process, role assignment, communication style, development life-cycle, organizational culture and technology [2]. Boehm makes this discussion over developers, customers, requirements, architecture, refactoring, team size and the primary objective of the development [3].

Stober and Hansmann compare the characteristics of software development teams to the characteristics of fractal units in mathematics which are self-similarity, goal-orientation, self-organization, self-improvement and vitality [4] .

The core set of Agile methods or to use Highsmith's phrase Agile Software Development Ecosystems [5] include Dynamic Systems Development Method [6], Scrum [7], Agile Software Process Model [8], Crystal collection [9-12], Extreme programming [13, 14], Internet Speed Development [15-17], Adaptive Software Development [18], Pragmatic Programming [19], Feature Driven Development [20], Agile Modelling [21], Lean Software Development [22] and Test Driven Development [23]. These models which are developed for varying real life conditions, share a set values which were later defined in agile manifesto in 2001[24].

For organizations transitioning from traditional approaches to agile approaches, major challenges are observed in customer involvement, documentation, upfront requirement analysis, document-driven testing, communication and knowledge sharing [25]. Heeagar mentions that it is difficult to find solutions to these challenges applicable in organizational contexts and valid in terms of agility. However, she does not provide any solutions that worked for large scale, and document driven software organizations [25].

Sekitoleko *et.al* [26] specify the challenges associated with using agile within large-scale distributed software development teams. They define "*technical dependency*" as "the relationships and interactions between artifacts and teams during product development". From this perspective, they present the major challenges as "*the planning challenge, the task prioritization challenge, the knowledge sharing challenge, the code quality challenge, and the integration challenge*". Even if the presented challenges are valid in a general context, we think that "technical dependency challenge" term misleads the reader. The term is usually used for the dependencies among architectural elements at the architectural level [27].

Kaisti *et.al*'s study is also valuable as they specified agile challenges in an embedded systems development domain, a domain that had not benefited much from the agility so far [28]. The major challenges listed in this study are "*high cost of change late in development due to hardware components*", "*difficulties in making frequent software releases", "not avoiding documentation at different levels of software development due to integration of different stakeholders to the process and due to the regulatory standards",* and *"long development cycles of electronics and mechanics design".*

There are a small number of studies on identification of agile adoption challenges specified through empirical research and focused on real life cases. Identification of optimum granularity levels for user stories has been studied by Liskin *et.al* [29]. Their work is one of the studies that analyzes the challenges and provides solutions for them.

Ramesh [30] *et. al* identified seven agile requirements engineering (RE) challenges by collecting data from 16 organizations with semi-structured interviews, participant observations and review of documents. The challenges are associated with RE practices are: "*agile estimation approaches and the need for early and accurate project estimates in projects*", "*agile architecture approach and architecture inadequacy with the emergence of requirements*", "*neglect of non-functional requirements and unavailability of on-site customer in requirements elicitation*", "*the conflicts between business value based requirement prioritization and the architecture development*", "*inadequate requirements verification*" and "*misunderstood minimum documentation concept*". Although those challenges point to significant problems in agile software development, all of the practices mentioned are not specific to agile requirements engineering. In addition, the problems mentioned about the neglect of non-functional requirements, minimal documentation and inadequate requirements verification cannot be listed as challenges in agile adoption rather they are examples of poor implementation choices and can be resolved without violating the agile principles.

Conboy *et. al* [31] examined the *people* related challenges associated with agile software development in 2011 through a case study followed by focused group discussions. They specified the following challenges: developer fear caused by skill deficiencies, the need for competence in broad range of skills, increased reliance on social interaction, lack of business knowledge among developers, lack of developer motivation in implementing agile methods, devolved decision making due to self-organization, and implementing agile compliant performance evaluation systems instead of individual evaluation.

Compared to other research papers described above, the last three emphasize the real-life problems. The others basically focus on the challenges from a high level (abstract) perspective. We do not ignore these attempts however, we are sure that we should be more specific about the challenges, the solutions and the surrounding conditions that created those challenges to provide insight about real life situations to the reader.

3 Software Agility Assessment Reference Model (AgilityMod)

AgilityMod is a software agility assessment reference model the structure of which was defined in accordance with the ISO/IEC 15504-Process Assessment Model [32, 33].

From AgilityMod's perspective, Agility is the capability of being able to give and obtain feedback rapidly, being adaptive to changing conditions, having confidence on developing solutions to complex problems, being creative and innovative, respecting others, working with humility, learning from mistakes, improving continuously, solving problems/issues with communication and moving away from complex and bureaucratic procedures.

The model mainly consists of two dimensions: *Aspect Dimension and Agility Dimension* which can be seen on Fig.1.

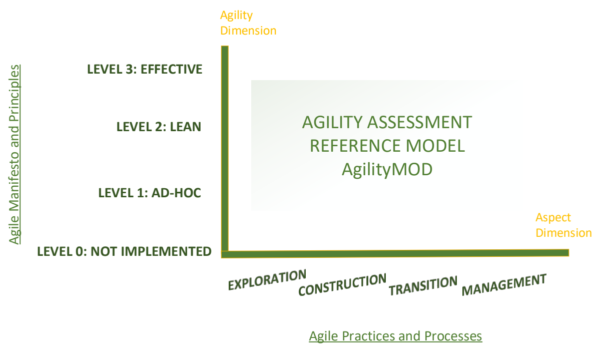


Fig. 1. Dimensions of AgilityMod

The first dimension of the Model is the Aspect Dimension. This dimension is described with Aspects rather than processes. Because, the formal process layers of traditional software development are intertwined to each other in agile software development. Therefore, it is difficult to specify boundaries for agile processes. From this perspective, the Aspects are a collection of interrelated and interacting activities of agile processes and practices that are integrated under meaningful and agile compatible abstract definitions [5]. The aspect dimension consists of four aspects: “Exploration”, “Construction”, “Transition” and “Management”. The cultural issues which are a significant part of agile adaptation are taken into account in the agility dimension.

The purpose of the *Exploration Aspect* is to understand the customer/user needs and to transform these needs into artifacts that initiate communication for elaboration on them during construction and manage the changes to these artifacts. The purpose of the *Construction Aspect* is to develop a high-quality software solution that is ready to be built, including architecture, design, coding and unit testing activities. The *Transition Aspect* includes practices to establish and maintain reliable and repeatable build, integration and deployment practices to keep the application in a working state throughout development. This makes it possible to obtain feedback in relation to the problems in the process, to make the whole process visible to everyone, and to shorten the response time for changes. Finally, the purpose of the *Management Aspect* is to perform planning and tracking activities continuously, and estimating collaboratively to achieve efficiency and performing these practices as value adding activities to the project life cycle

At the Agility Dimension, Agility of an aspect is described with four Levels: “Not Implemented (L0)”, “Ad-Hoc (L1)”, “Lean (L2)” and “Effective (L3)”. When an aspect progresses from the bottom level to the top level, its conformance to agile values and principles increases.

At Level 0, the aspect practices are either not achieved or partially achieved. At Level 1, fundamental development processes such as requirements development, design, coding, integration, testing, and deployment are performed consistently. There are transition attempts towards agility by exploring best fitting agile practices or approaches. Aspect practices are implemented and aspect purposes are achieved; however agile values and principles are not fully incorporated into aspect practices. At Level 2, work products are developed iteratively and incrementally, non-value-added activities are eliminated from the aspect practices, balance is achieved between adaptive and predictive works. At Level 3, each aspect is performed to achieve delivering value with high productivity and low defects by employing agile engineering practices and using agile tools to support a continuously improving environment.

4 The Case Study

We performed this study using a *qualitative research approach* where the researchers collect data in the natural settings through overviewing documents, observing behavior or interviewing participants [34]. Through reviewing other qualitative research strategies, we selected the case study research to collect and analyze the empirical evidence. The case study research is suitable for "how" and "why" type questions and the researcher has little control over events in a real-life context [35].

We performed a multiple case study that included eight cases. We observed the applicability of AgilityMod for the identification of agility gaps in software projects and also to identify strengths and the weaknesses of the Model. We explained the case study and the results in detail in [36]. As the scope of this paper is to describe the agile adaption challenges and lessons learnt from the most successful cases, here we describe only the two cases that had achieved the highest agility levels from the eight cases: **Case G** and **Case C**.

4.1 Description of the Cases

We selected the eight cases randomly in order to observe different levels of agility for each aspect. The rationale behind this approach is that without performing an agility assessment, it is not rational and easy to make judgments about the maturity of the cases, just by knowing the duration of agile adaption efforts.

Below, we both describe the two organizations that the projects were performed in and the two projects that were subjected to this research. Again, we are not describing all eight cases here, just the cases that achieved the hightest agility levels after the assessment with AgilityMod.

Case G

Organization G is a government IT organization responsible for developing e-government software for various governance organizations. It is located in Ankara, Turkey.

The project (Case G) that was subjected to the assessment, is an e-government project, providing solutions to 40 foundations which are located in different cities of Turkey and with approximately 25 million Turkish citizens. Case G includes 21 employees divided into four teams which report to a project manager and an assistant project manager. Three of these teams purely work on software modules, the last one is involved in both system infrastructure and software development activities. Each team has a technical team leader. Other members of the team did not have specific roles, each one was involved in analysis, design and development activities. Since the beginning of the project in 2009, 7 million LOC has been developed. The product is developed in iterations, each of which is one month in length. There is a signed contract between the organization and the customer to specify the deadlines and budget.

The functional domain of the assessed project, is classified as a "Controlling Data System" based on the CHAR group method [37].

Case C

Organization C works within the internet security domain. They develop products with the purpose of securing information on the internet, securing websites and e-commerce applications and personal computers. Org. C is an international company, doing business over 100 countries, having 25 million end users, and over 7000 business partners.

Case C is a digital advertisement sharing platform. It is in use and new versions of the product are being deployed continuously. The purpose of the project is to ensure the security of the advertisements and to deliver harmless and focused advertisements to end users. The project includes 22 employees. There are three different development teams and Scrum Masters for each of the teams. Overall, there are four testers, 13 developers and an architect. Additionally, each team has a program manager. Apart from these members, there is a product owner residing in the US.

Case C is built upon legacy code. Java, PhP and Python languages are being used for different modules of the product. The project includes big data analyzes performed using the tools Cassandra and Hadoop. Scrum is used for project management activities. The product is built iteratively with each iteration lasting three weeks.

4.2 Case Study Conduct

We assessed the Agility of Case G and Case C by both interviewing with project team members and observing of the outputs produced during the project life cycles. Prior to the case study conduct, we developed a list of scripted questions related to the aspect practices and agility practices of the Model. We followed a semi-structured interview which included asking additional questions based on the project context and challenges. Each interview session was recorded and transcribed. The assessor was one of the authors of this paper who had four years of experience on agile software development at that time.

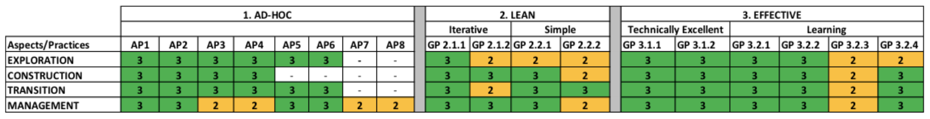
For Case G, the assessment was performed over a three-hour time period with the technical leader of the infrastructure team who had worked formerly as a developer and has knowledge about the project's processes. For Case C, the assessment was performed in a three-hours with the configuration manager and the quality assurance manager who is also the scrum master of the project.

After the assessments, we developed the agility assessment reports and shared these reports with the case organizations. In addition, we presented the results to the assessment teams. The presentations covered the assessment findings, the aspect levels and the improvement suggestions. After or during each presentation, we discussed the results with attendees. Thus, we ensured the validity of the case study results by discussing our findings and observations with the interviewees and managers in the organizations.

We present the agility assessment results in Fig. 2 and Fig. 3 for Case G and Case C respectively. These figures give the colored schema of the assessment ratings to provide a visual high-level view of the findings. Each column refers to the practices of AgilityMod (APx and GPx). The colors and the numbers in each cell refer to the achieved levels of these practices. We used a-four-level rating scale to express the achievement of the aspect attributes: "not achieved (0-red), partially achieved (1-yellow), largely achieved (2-orange) and fully achieved (3-green) and not applicable (NA)". For an agility level to be reached, all the practices should be largely or fully achieved.

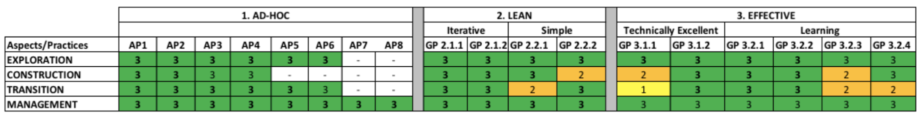
The Case G achieved Level-3: Effective for all of the aspects. This essentially means that Case G's aspects are iteratively performed, lean, technically excellent and continuously improving. Fast feedback is obtained and effectively communicated among team members. On the other hand, the customers are located in a different building. The communication between the customer and the team is not as effective as the communication among team members. The ways to communicate with the customer needed to be improved. It was found that retrospective studies to identify improvement areas are not performed regularly. It is suggested to perform regular retrospective studies that would provide much more value to the processes. Another recommendation provided to teams was to establish a generic measurement framework to improve the decision making.

Similarly, Case C also showed very good results. All of the practices of exploration and management aspects of Case C were rated as fully achieved. The Construction aspect of Case C is at Level 3. The weakest aspect of Case C is the "Transition" aspect which is at Level 2: Lean level. The major improvement areas for this project in terms of the Agility are achieving continuous integration, continuous delivery and increasing unit test coverage and automated test ratio, refactoring continuously and managing technical debt better.



	1. AD-HOC								2. LEAN				3. EFFECTIVE					
									Iterative		Simple		Technically Excellent		Learning			
Aspects/Practices	AP1	AP2	AP3	AP4	AP5	AP6	AP7	AP8	GP 2.1.1	GP 2.1.2	GP 2.2.1	GP 2.2.2	GP 3.1.1	GP 3.1.2	GP 3.2.1	GP 3.2.2	GP 3.2.3	GP 3.2.4
EXPLORATION	3	3	3	3	3	3	-	-	3	2	2	2	3	3	3	3	2	2
CONSTRUCTION	3	3	3	3	-	-	-	-	3	3	3	2	3	3	3	3	2	3
TRANSITION	3	3	3	3	3	3	-	-	3	2	3	3	3	3	3	3	2	3
MANAGEMENT	3	3	2	2	3	3	2	2	3	3	3	2	3	3	3	3	2	3

Fig. 2. Rating of Each Practices of Case G



	1. AD-HOC								2. LEAN				3. EFFECTIVE					
									Iterative		Simple		Technically Excellent		Learning			
Aspects/Practices	AP1	AP2	AP3	AP4	AP5	AP6	AP7	AP8	GP 2.1.1	GP 2.1.2	GP 2.2.1	GP 2.2.2	GP 3.1.1	GP 3.1.2	GP 3.2.1	GP 3.2.2	GP 3.2.3	GP 3.2.4
EXPLORATION	3	3	3	3	3	3	-	-	3	3	3	3	3	3	3	3	3	3
CONSTRUCTION	3	3	3	3	-	-	-	-	3	3	3	2	2	3	3	3	2	3
TRANSITION	3	3	3	3	3	3	-	-	3	3	2	3	1	3	3	3	2	2
MANAGEMENT	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

Fig. 3. Rating of Each Practices of Case C

In Fig. 4, we provide a comparative radar chart to display the differences between the ideal case (blue line) (the case that the all practices were fully achieved) and the current situation of Case G (orange line) and Case C (grey line). The data to draw the current situation of the cases for each aspect was obtained by adding the rating values in each cell along with the horizontal columns given on Fig. 2 and Fig. 3 (E: Exploration, M: Management, T: Transition, C: Construction). This chart allows us to observe the how far each case's *Aspect* is from being fully agile according to AgilityMod.

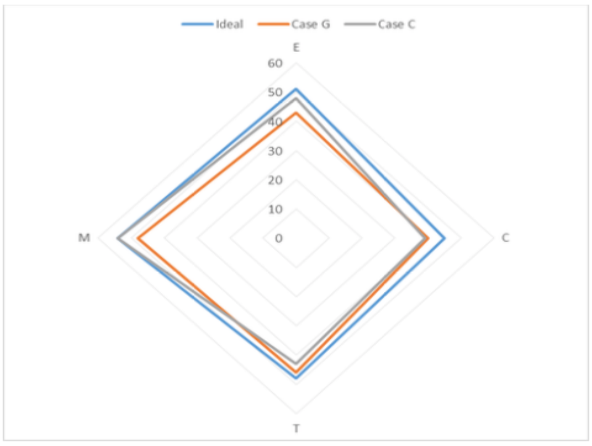


Fig. 4. Comparison of Case G and Case C with the Ideal Case

5 Challenges Faced Agile Adaptation and Working Solutions

In this section, we present two types of challenges: First, the challenges that Case C and Case G teams overcame through implementing successful solutions (Resolved). Second, the challenges that need further discussion (Unresolved). A summary of the challenges and working solutions is provided in Table 1.

5.1 Resolved Challenges

Having no on-site customer: As specified in the Agile Manifesto, one of the critical success factors in agile software development is the level of interaction between software development teams with their customers.

In Case G, the Product Owner (PO) lives in the United States, while the rest of the team reside in Turkey. However, the product owner communicates with the program managers regularly (3 to 5 times in a week) over teleconferencing despite the 8 hours difference. The PO does not only communicate with the program managers but also with the scrum masters and the developers when further clarification is required for the backlog items.

This case shows that "distance" is not an excuse for low levels of communication with the customer. The commitment of the customer is a very significant part of overcoming the *no on-site customer* challenge. In Case C, what we found was that having customers on different continents has given Case C teams significant insight into the importance of communication where they have focused on ways to strengthen their interaction channels.

Granularity level of requirements: In agile software development, user stories are a widespread way of specifying software requirements. However, the identification of optimum granularity levels for user stories still remains a challenge. Coarsely granulated user stories are the source of difficulties and problems in the estimation process [29].

In Case C, a well working process has been implemented for this challenge. It was stated that the business needs were transformed into software requirements through the following stages: First, the product owner and/or program managers define the business needs as “epics” in the product backlog in Atlassian’s Jira tool. Second, the program managers and software development teams perform “backlog grooming meetings” once a week where each epic were split-down into user stories.

It is essential to mention here that these grooming meetings were performed independent of the scope of the upcoming sprints. If the teams require further information to clarify some issues, the PO was requested to involve in those meetings.

The teams use two approaches to decide on the optimum granularity level for user stories. The first one is the “story points (sp) estimation”. A user story is not included in a sprint, if its size is above a threshold (for Case C, it was 20 sp.). A user story is discussed, detailed and re-estimated until it reaches the pre-specified sp level. The second one is the “acceptance criterion” which is essential for both of the development and test teams. Definability of the acceptance criteria is an indicator of a well-defined user story which was spilt-down into a sufficient level of detail.

Growth of product backlog at a constant pace: As specified by Rubin [38] and defined in AgilityMod [39] as a generic agility practice; the balance between the upcoming items and outgoing items in a software development process has a significant effect on the work to be done without interruption. This flow has to be smooth so that developers do not need to wait to receive new items to develop from business analysts; and that testers are also not idle just because there is no current item to be tested.

In Case C, there had been a stage when this flow had been interrupted. The product backlog had not grown in a constant pace. Upon investigation, it was observed that the issue arose due to communication problems among the PO in the US and the program managers in Turkey. Once they sensed the reason for the problem, they established a communication matrix that had to be updated whenever the PO and the program managers communicated with each other. After a while, the problem became so obvious that the frequency of the communication was once or twice in a month between some of the program managers and the product owner. A communication matrix is not a direct solution for this challenge, but it is a very effective tool to observe communication problems.

Since the effectiveness of communication is considered a critical success factor in agile software development and the effect of the communication matrix observed by the Case C team, they started using it to observe all the interactions within the project.

Consequently, Case C established a correlation between the growth of the product backlog and the numbers in the communication matrix.

Another solution for the continuous product backlog growth in the Case C, is the conduct of regular product backlog grooming (PBG) meetings. The program managers, scrum masters, developers, testers and the product owner are the members of PBG meetings. Teams spend almost two hours a week for elaboration activities. Since it is regular, even a two-hour-time makes a big difference in solving such a significant problem.

Nonfunctional retrospective meetings: Retrospective meetings are one of the ways to transform good teams to great teams [40]. It is one of the easiest ways to turn the challenges into successful practices in agile software development. However, they may easily turn into useless meetings.

When the Case C quality assurance manager observed that none of the improvement suggestions proposed in the retrospective meetings were implemented, he decided two things: The first one was to open action items for each of the issues and assign the items to team members using the Jira tool. The second one was to specify a team quality criterion based upon the percentage of closed retrospective issues in Jira. Both these approaches allowed the Case C teams to perform effective retrospective meetings and observe the results very quickly.

Ineffective review meetings: Conventional review meetings might be a waste of time for software development teams in some cases. Especially when lots of ideas are discussed, without a decision being made. The solution found by the Case G team was to provide a tool support for design and code reviews. They utilized the Confluence tool to review class and sequence diagrams in software design and the Crucible tool for design and code reviews. They specified the rule of everyone on the team can comment on the code parts and all the comments are seen by other reviewers. After this initial review, final remarks are decided with a meeting if necessary.

Motivation problems and software quality: There are various reasons for motivational problems in software development teams.

The Case G team members had suffered from high personnel turnover in the testing team and this was not the only problem. They were in a continuous "fire-fighting" reactive state, because of the bugs found in released versions of the product G.

In such a case, it might be very difficult to notice the sources of the problem and easy to blame other people working in the team. Eventually the Case G team noticed that the problem mentioned above was due to a decrease in the motivation level of the testers. The managers in the company were hiring successful, experienced and talented developers to establish a strong testing team. However, they were assigned mostly black-box manual testing roles which do not require good development skills but require domain knowledge.

The managers of the Case G have made a radical decision to quit manual testing and abolished the test team. All of the testers were assigned to different parts of the development team where there is no distinction among team members. Then, they were asked to code the automated unit tests. It was mentioned that this had been one of the breaking points for the Case G team in terms of moving towards agility. They resolved this challenge by collaborative work and adopting shared responsibility.

Ability to manage technical debt: Technical debt is evitable in software development, but it can be managed. In some cases, team needs to develop quick solutions or hot fixes which may cause technical debt. On the other hand, it is very easy to overlook the created technical debt in daily life turmoil, if there is no specific mechanism to control it.

The solution that was found by the Case G team for this challenge by assigning the responsibility of recovery from technical debt to the person who created it and following the progress of such recoveries via a tracking system such as Jira.

Table 1. Summary of the Challenges and Working Solutions

Challenges	Working Solutions
No on-site customer	Take the commitment of the customer for frequent meetings Decide the frequency of the meetings Involve customer and product owner to team discussions
Varying granularity level of requirements	Break down the user stories until the size of each is below a threshold level Define acceptance criteria for each user story
Nonfunctional retrospective meetings	Define action items in issue or item tracking system for each of the improvement items and assign that items to a person Review all the previously specified improvement action items before the retrospective meeting
Ineffective review meetings	Use tool support to review design and code Let everyone in the team comment on the code parts and all the comments seen by other reviewers Decide final remarks with a short meeting
Motivation problems caused by team divisions	Combine test teams and development teams Abandon manual testing except for the exploratory testing Support collaborative work and shared responsibility
Ability to manage technical debt	Give the responsibility to manage the technical debt to the person who created it Make sure you recorded the technical debt to not to overlook it

5.2 Unresolved Challenges

The following challenges were observed in the projects; however, satisfactory solutions have not been found for them yet.

Identification of the dependencies among design elements for change management: Knowing the relationship between design elements has a significant impact on identification of changes within an existing software system. On the other hand, the larger the system, the more difficult it is to specify the relations among modules or lower level design parts. In addition, teams mostly overlook and rely on personal experiences for change impact analyses until the system grows to an unmanageable size.

This was what happened in both Cases. The impact of new requirements on modules and lower level module components were evaluated based on personal experiences. To date, they have not experienced significant issues due to not establishing traceability. But, this is a valid concern for them as their systems grow rapidly.

Brown, Nord and Ozkaya emphasize the importance of architectural agility in achieving success [27]. They suggest dependency analysis among architectural elements and high-level design capabilities. We suggested this approach to both teams at the multiple case study reporting phase. Unfortunately, this approach did not find much interest by agile software developers in our cases. They found it very difficult to establish a relationship matrix manually and maintain it with every change. Therefore, this challenge remains valid and we feel that effective and practical ways to establish design relations needs to be identified.

The efficiency of the code comments: Code comments are significant especially for the living software systems where a policy of little documentation is applied. Today's source control systems do not allow developers to check-out code parts without any comments. But the efficiency of the code comments is not evaluated. There is a need to identify and evaluate efficiency of code comments to increase the clarity of the code especially at maintenance phase of a software development life cycle.

6 Conclusion and Future Work

Agile software development methods are frequently adapted in recent years by the software community as they are seen as well solutions for software development problems. Upon the introduction of this new approach to traditional software development environments, researchers and practitioners started to deal with agile adaptation challenges.

As most of the agile software development models are not highly prescriptive in terms of adoption processes, the experience reports published in this topic remains as an important problem for practitioners in specific contexts.

In this paper, we briefly described the Software Agility Assessment Reference Model (AgilityMod) the purpose of which is to assist software organizations in assessing projects' agility levels, indicating the gaps that prevent fully obtaining the benefits of agile software development and introducing roadmaps in adopting agile principles/practices. Secondly, based on the multiple case study that we had assessed software projects' agility with AgilityMod, we selected most successful two cases among the eight cases. We presented the challenges that these cases had faced during agile adaptation and the best solutions that worked very well for them. The major contribution of this study is the insights provided to readers about real life challenges faced and how these challenges were overcame. We also discussed two unresolved challenges that will require further research.

Further studies need to be performed for the discovery of efficient solutions for such problems. We believe that the software industry will benefit from experience reports discussing challenges observed and lessons learnt in different domains.

ACKNOWLEDGMENTS
This study is partially supported by Turkish Scientific and Technological Research Council of Turkey (TÜBİTAK), grant number 113E528. This research is also supported in part by Science Foundation Ireland under a co-funding initiative by the Irish Government and European Regional Development Fund),and by Lero - the Irish Software Research Centre (http://www.lero.ie) grant 10/CE/I1855 & 13/RC/20194

References

1. T. Dingsøyr, T. Dybå, N. Brede Moe, T. Dingsøyr, and T. Dybå, "Agile Software Development," Agile Software Development: Current Research and Future Directions, ISBN 978-3-642-12574-4. Springer-Verlag Berlin Heidelberg, 2010, vol. 1, 2010.
2. S. Nerur, R. Mahapatra, and G. Mangalaraj, "Challenges of migrating to agile methodologies," *Communications of the ACM,* vol. 48, pp. 72-78, 2005.
3. B. Boehm, "Get ready for agile methods, with care," *Computer,* vol. 35 pp. 64-69, 2002.
4. T. Stober and U. Hansmann, Agile Software Development: Best Practices for Large Software Development Projects vol. 3: Springer, 2010.
5. J. Highsmith, "What Is Agile Software Development?," *The Journal of Defense Software Engineering,* vol. 15, pp. 4-9, 2002.
6. J. Stapleton, DSDM Dynamic Systems Development Method: the method in practice: Cambridge University Press, 1997.
7. K. Schwaber, "Scrum development process," in *Business Object Design and Implementation*, ed: Springer, 1997, pp. 117-134.
8. M. Aoyama, "Agile software process model," in Computer Software and Applications Conference, 1997. COMPSAC'97. Proceedings., The Twenty-First Annual International, 1997, pp. 454-459.
9. A. Cockburn, Crystal clear: a human-powered methodology for small teams: Addison-Wesley Professional, 2004.
10. A. Cockburn, Agile software development: the cooperative game (agile software development series): Addison-Wesley Professional, 2006.
11. A. Cockburn, *Surviving object-oriented projects: a manager's guide*: Addison-Wesley Longman Publishing Co., Inc., 1998.
12. A. Cockburn, "Writing effective use cases, The crystal collection for software professionals," ed: Addison-Wesley Professional Reading, 2000.
13. K. Beck, *Extreme programming explained: embrace change*: Addison-Wesley Professional, 2000.
14. K. Beck, "Embracing change with extreme programming," *Computer,* vol. 32, pp. 70-77, 1999.
15. M. A. Cusumano and D. B. Yoffie, "Software development on Internet time," *Computer,* vol. 32, pp. 60-69, 1999.
16. R. Baskerville, L. Levine, J. Pries-Heje, B. Ramesh, and S. Slaughter, "How Internet software companies negotiate quality," *Computer,* vol. 34, pp. 51-57, 2001.
17. R. Baskerville and J. Pries-Heje, "Racing the E-bomb: How the Internet is redefining information systems development methodology," in *Realigning research and practice in information systems development*, ed: Springer, 2001, pp. 49-68.
18. J. A. Highsmith and K. Orr, Adaptive software development: a collaborative approach to managing complex systems: Dorset House Pub., 2000.
19. A. Hunt, The pragmatic programmer: from journeyman to master: Addison-Wesley Professional, 2000.
20. S. R. Palmer and M. Felsing, *A practical guide to feature-driven development*: Pearson Education, 2001.
21. S. W. Ambler, *Agile modeling*: Wiley, 2002.
22. M. Poppendieck and T. Poppendieck, *Lean software development: An agile toolkit*: Addison-Wesley Professional, 2003.
23. K. Beck, *Test-driven development: by example*: Addison-Wesley Professional, 2003.
24. (2001). *Agile Manifesto*. Available: www.agilemanifesto.org

25. L. T. Heeager, "How Can Agile and Documentation-Driven Methods be Meshed in Practice?," in *Agile Processes in Software Engineering and Extreme Programming*, ed: Springer, 2014, pp. 62-77.
26. N. Sekitoleko, F. Evbota, E. Knauss, A. Sandberg, M. Chaudron, and H. H. Olsson, "Technical Dependency Challenges in Large-Scale Agile Software Development," in *Agile Processes in Software Engineering and Extreme Programming*, ed: Springer, 2014, pp. 46-61.
27. N. Brown, R. Nord, and I. Ozkaya, "Enabling Agility Through Architecture," DTIC Document2010.
28. M. Kaisti, T. Mujunen, T. Mäkilä, V. Rantala, and T. Lehtonen, "Agile principles in the embedded system development," in *Agile Processes in Software Engineering and Extreme Programming*, ed: Springer, 2014, pp. 16-31.
29. O. Liskin, R. Pham, S. Kiesling, and K. Schneider, "Why we need a granularity concept for user stories," in *Agile Processes in Software Engineering and Extreme Programming*, ed: Springer, 2014, pp. 110-125.
30. B. Ramesh, L. Cao, and R. Baskerville, "Agile requirements engineering practices and challenges: an empirical study," *Information Systems Journal,* vol. 20, pp. 449-480, 2010.
31. K. Conboy, S. Coyle, X. Wang, and M. Pikkarainen, "People over process: key people challenges in agile development," 2011.
32. "ISO/IEC 15504-2:2003 Information technology -- Process assessment -- Part 2: Performing an assessment," ed, 2003.
33. "ISO/IEC 15504-5:2012 Information technology -- Process assessment -- Part 5: An exemplar software life cycle process assessment model," ed, 2012.
34. J. W. Creswell, Research design: Qualitative, quantitative, and mixed methods approaches: Sage Publications, Inc, 2009.
35. R. K. Yin, *Case study research: Design and methods*: Sage publications, 2014.
36. Ö. Özcan-Top, and O. Demirors, "Application of a software agility assessment model–AgilityMod in the field". *Computer Standards & Interfaces*, *62*, 1-16, 2019.
37. SO/IEC, "IS 14143-5 Information Technology – Software Measurement - Functional Size Measurement - Part 5: Determination of Functional Domains for Use with Functional Size Measurement," ed, 2004.
38. K. S. Rubin, Essential Scrum: A Practical Guide to the Most Popular Agile Process: Addison-Wesley Professional, 2012.
39. Ö. Özcan Top, "AgilityMod: Software Agility Assessment Reference Model v3.0," Informatics Institute, METU/II-TR-2014-392014.
40. E. Derby, D. Larsen, and K. Schwaber, *Agile retrospectives: Making good teams great*: Pragmatic Bookshelf Raleigh, NC, 2006.