Towards A Broader Adoption of Agile Software Development Methods

Abdallah Alashqur
Software Engineering Department
Faculty of Information
Applied Science Private University
Amman, JORDAN

***Abstract*—Traditionally, software design and development has been following the engineering approach as exemplified by the waterfall model, where specifications have to be fully detailed and agreed upon prior to starting the software construction process. Agile software development is a relatively new approach in which specifications are allowed to evolve even after the beginning of the development process, among other characteristics. Thus, agile methods provide more flexibility than the waterfall model, which is a very useful feature in many projects. To benefit from the advantages provided by agile methods, the adoption rate of these methods in software development projects can be further encouraged if certain practices and techniques in agile methods are improved. In this paper, an analysis is provided of several practices and techniques that are part of agile methods that may hinder their broader acceptance. Further, solutions are proposed to improve such practices and consequently facilitate a wider adoption rate of agile methods in software development.**

***Keywords*—Agile Methods; Agile software development; SCRUM**

I. INTRODUCTION

Software systems research and development has resulted in many applications covering various aspects of our lives [1-5]. Over the years, two major approaches for managing the software development process have evolved. These approaches are the traditional engineering approach exemplified by the *waterfall model* and its variations [6-8] and the more recent approach called *agile software development methods* [9-12]. The waterfall model as was originally introduced by Winston Royce [13] does not allow feedback from later steps to earlier steps in the process, thus adopting the engineering approach. In the engineering approach (e.g., civil engineering) requirements and specifications have to be fully completed and approved before construction starts.

Some level of flexibility has been incorporated in later versions of the waterfall model as shown in Figure 1. This flexibility is achieved by enabling feedback to previous steps of the model, which makes it possible to perform limited modifications to prior phases of the development lifecycle. Another disadvantage of the waterfall approach is that the user cannot see any running components of the software being developed until the entire system is completed, which is normally way too far down the road. Furthermore, a lot of focus and effort is invested upfront in just documentation and planning.

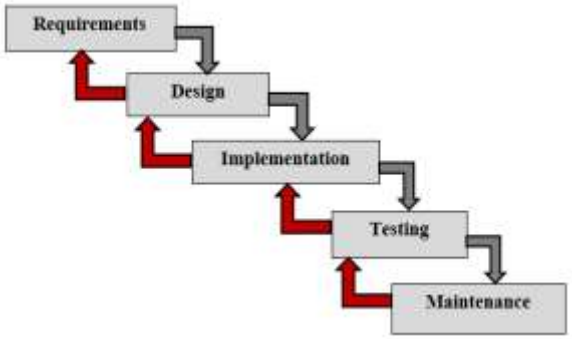

Fig. 1. Enhanced Waterfall Model

Due to the shortcomings of the waterfall-based development methods, a new approach called *agile software development*, or sometimes called Agile Methodology or just A*gile Methods (AM)*, has emerged as a viable and powerful approach to software development. In agile software development, portions of the software are designed and developed in short iterations in an incremental way. After each iteration, the user has a chance to see the outcome in the form of a running subsystem and to provide more feedback to the development team. This iterative approach allows for flexibility and takes into consideration the fact that the user may not know for sure, and in detail, what he/she wants prior to starting the development process.

Despite the advantages provided by agile methods over traditional methods, there are still several aspects in which agile methods can be further improved and several issues that need to be addressed. This paper aims at exposing several of these issues for the purpose of understanding them and being able to identify workarounds and solutions to handle them. In addition, in this paper some solutions to tackle these issues are proposed. The overall objective is to make agile methods more appealing to a wider audience in the software development community.

The rest of this paper is organized as follows. In section 2 a brief description of agile methods is provided with a focus on the values and principles of agile software development. Also a brief description of one of the agile methods is

described in some briefly. Section 3 describes several of the issues with agile methods with a description of each one of these issues and its impact. Solutions are proposed in Section 4 on how to deal with these issues. Conclusions are given in Section 5.

II. Agile Methodology And SCRUM

In this section a brief description of the agile values and principles is provided. Then a brief descriptions of SCRUM [14], which is an important agile methodology is given. Some terms and concepts of SCRUM will be used in subsequent sections of this paper.

A. Agile Methodology

The term *agile software development* was introduced after extensive meetings and discussions conducted by seventeen experienced individuals in the area of software development in 2001. The outcome of those meetings was summarized in a document called *The Agile Manifesto* [15], which describes the values and principles of agile software development.

The authors of The Agile Manifesto [15] cited four qualities that they value in agile development over four related qualities that exist in traditional software development. These four qualities are:

1) Individuals and interactions are valued more than processes and tools. Individuals are team members of the agile development teams. Agile teams are usually self-organizing and cross-functional teams.

2) Working software is valued more than comprehensive documentation. The primary objective of agile teams is to provide the client with early and working subsystems to keep the customer engaged and to obtain feedback.

3) Customer collaboration is valued more than contract negotiation. Collaboration between the customer and the agile team on a continuous basis is necessary to obtain feedback and make sure that deliverables meet customers' expectations.

4) Responding to change is valued more than following a plan. Permitting flexibility and providing a culture where requirements are allowed to evolve results in a final software that better meets the customer's needs. On the contrary, adhering to a fixed plan may result in software that is not exactly what the customer needs.

The Agile Manifesto [15] cited twelve principles for agile software development. These principles are summarized in Table 1.

TABLE I. Principles of Agile Methods

No	Principle
1	Satisfy the customer through early and continuous delivery of valuable software.
2	Welcome changing requirements during the development phases.
3	Produce working software on a frequent basis.
4	Business people and developers must work jointly and daily throughout the project.
5	Build projects around motivated individuals. Provide them the environment and support they need.
6	Face-to-face conversation is the best way of communication
7	Working software is the primary measure of progress.
8	Promote sustainable development. Maintain a constant pace indefinitely.
9	Continuous attention to technical excellence and good design.
10	Simplicity (the art of minimizing the amount of work that is done) is essential.
11	The best architectures, requirements, and designs emerge from self-organizing teams.
12	At regular intervals, the team reflects on how to become more effective.

B. SCRUM

SCRUM is the most popular agile methodology [16-18]. Development in SCRUM is performed as a series of iterations called *sprints* as shown in Figure 2. A sprint is a time-box whose duration is 2 to 4 weeks. The output of a spring is an increment or subsystem of the overall system being developed. Each sprint can be viewed as a small project that has its own system development life cycle (SDLC). The final, aggregate product is the result of integrating the subsystems produced by these sprints.

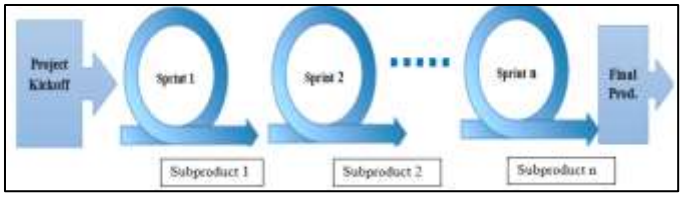

Fig. 2. SCRUM sequence of sprints

Figure 3 details a single sprint. The *Product Backlog* shown on the left side of Figure 3 is a list of requirements and features that need to be included in the end product. In addition to requirements, the Product Backlog contains

description for any changes to be made to what has been produced so far. The content of the Product Backlog evolves overtime to permit requirements to change. The *Sprint Backlog* is a subset of the items in the Product Backlog that are selected for implementation in the current sprint.

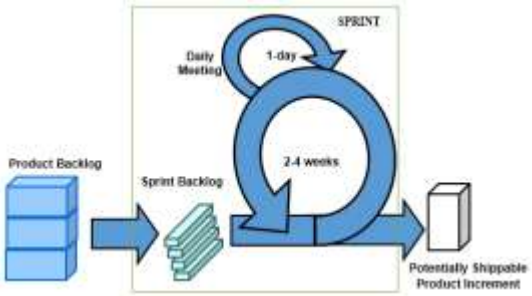

Fig. 3. Details of a single sprint

On a daily basis, the SCRUM team holds a short meeting, also called daily-standup meeting (or SCRUM meeting). This meeting is represented in Figure 3 with the arrow labeled "Daily Meeting." In this meeting, team members present what was done in the past 24 hours and discuss the plan for the coming 24 hours. In SCRUM the product that is produced at the end of a sprint is called potentially shippable product increment, which is the release or subsystem produced by the sprint.

Scaling up the agile process. Sprints can be performed serially by the team. However in large projects, there can be several parallel sprints, where multiple teams can be working on different sub-products simultaneously. In this case we can have what is referred to as "team of teams" or "scrum of scrums" [19]. In its purist form, agile methods do not allow for team-of-teams structure in order to stay away from forming a hierarchical management structure. The scrum of scrums technique is used to scale-up SCRUM to handle large projects. However when this is done, coordination and collaboration between the teams become an overhead. Teams have their own daily meetings as usual. But then each team selects a representative to attend the scrum of scrums meeting to plan the overall project and coordinate the various development efforts. The scrum of scrums meetings may be scheduled less frequently than the scrum (or sprint) meetings.

III. Agile Practices that May Discourage Wider Acceptance

Below are the agile practices that need improvement and can be considered obstacles preventing many organization from adopting agile methods fully. They are based on the author's extensive experience in software research, design, and development.

1) Pushing items back to the product backlog. Whenever a team encounters or discovers a major bug, the team may push it back in the product backlog (with the approval of the product owner or user). In reality this bug can be a problem with current sprint implementation. But instead of solving it during the current sprint, the team postpones it to a future sprint by "kicking the can down the road." Because of scheduling pressures, the team may be tempted to postpone some genuine current-sprint work by hiding it as a bug, thus effectively taking it out of the sprint backlog and pushing it back to the product backlog. That way the team can meet strict deadlines and appear as a team of high performance.

2) Not valuing individuals. In agile methods, a team is treated and measured as a single entity. The performance of the entire team is measured without much regard to differences in the performance of individual members of a team (except may be for the purposes of discovering very low performers and taking some measures towards them). By not allowing the "stars" in a team to shine and not giving them credit for their achievements, their incentive for doing outstanding work diminishes. This negatively impacts the overall project. In a team of twenty individuals, the real stars of overachievers could be three or four individuals. These are the ones who can do magic in solving very hard problems and overcoming tough obstacles. You don't incentivize them by telling them that no matter what they do, their work will be considered as a team achievement and that they will not be rewarded for it.

3) Treating programmers as interchangeable resources. A tendency may exist in some development environments to treat programmers just as a bunch of interchangeable "techies" or "resources". This behavior negatively affects moral and enthusiasm towards the work environment as a whole and makes it harder to attract talented individuals to fill future open positions. This problem is aggravated if most of those "techies" are people who have language accents that are different from the main stream accent. This usually gives a false impression that "accent" is an intentional line of segregation where people with accent are given low-level implementation tasks and are treated as pluggable resources. A similar problem can happen in a distributed agile team, where some members of the team are in one country and other members are in another country. Team members in one country may perceive themselves as the "thinkers" and decision makers whereas team members in the other country are perceived as just "doers."

4) Agile Methods focus on short term iterations. This means that the time available for developers to learn and experiment with new ideas is limited if not totally eliminated. Many of those developers are highly intellectual individuals who would dislike it if their work is transformed to cookie-cutter, non-intellectual, and repetitive task patterns. Computer science is a fast-evolving field and giving developers some room to experiment with new ideas is important. In traditional software engineering methods tasks and modules are usually large in size and not very limited and short-term as in the case of agile methods. This gives developers who use traditional methods the leeway to perform some level of research and experimentation that will not only benefit the specific task at hand but the overall project.

5) Scaling-up to handle Large Projects. Agile methods emphasize teams but avoid the adoption of a management hierarchy. In traditional methods, there is normally a

management hierarchy that can expand in size (in either depth or breadth) as much as needed to accommodate all components of a project. Because of this, traditional software engineering methods are more capable of scaling up to handle projects of large size. Agile methods are more appropriate for handling small size projects and, to some extent, medium size projects. A need exists to scale-up agile methods to handle larger projects.

6) High complexity of the system Integration process. Because agile methods emphasize short iterations that produce small subsystems, the integration of these many small subsystems into a coherent, working, and bug-free system becomes a very complex task that is difficult to accomplish.

7) Determining a project budget upfront. At the heart of agile methods is the idea of not freezing the requirements at the very beginning of a project in order to give the client the ability to introduce new requirements and modify existing requirements on an on-going basis. But this gives rise to the problem of not being able to have a clear agreement with the client regarding budget and schedule before the start of a project. This opens the door for potential disputes between the client and the developer during project execution, which is a major risk factor. How can we preserve the agility and benefit from the flexibility it provides, but at the same time avoid running into budget-related issues during project execution? This is a big question. The problem can be less sever if the client and developer are two different departments within the same company. However, if they are two different companies, the budget issue becomes a high risk area that hinders the adoption of agile methods especially in projects of large size.

IV. Proposed Ways of Dealing with the Above Issues

1) Pushing items back to the product backlog. The following three solutions can be implemented. (1) A titer approval process needs to be put in place, in order to avoid pushing items back to the product backlog unless it is absolutely necessary. (2) A record of these incidents need to be saved, in order to expose situations in which a team frequently resorts to pushing items back to the product backlog, which may indicate a potential problem. (3) The number of items placed by a team on the product backlog during a sprint implementation needs to be used as one of the metrics for measuring team performance. Less items pushed to the product backlog contribute to a higher performance measure.

2) Not valuing individuals. In addition to performing team appraisals, individual appraisals are necessary. Some sort of reporting hierarchy needs to exist in order for a manager or team leader to perform individual reviews and reward exceptional achievers. Even though the agile methodology tries to avoid having a management hierarchy, it is necessary to have some form of reporting hierarchy for the purposes of assessing and rewarding team members.

3) Treating programmers as interchangeable resources. Involving some of those programmers, especially the senior ones, in the decision making process at the strategic level as well as at the tactical level may help alleviate this problem. Recognizing that the skills and experience of each individual are distinguished and appreciating the uniqueness of each individual is a step in the right direction.

4) Agile Methodology focuses on short term iterations. Allowing for extra time during a sprint or between sprints to reflect, learn, and experiment is one way to reduce the impact of sprints of the agile methods being very short term and tactically focused. Sending agile team members to short training courses (e.g., one week) on a quarterly or semi-annual basis may partially satisfy the need of those individuals to progress at their careers. This elevates their moral and enthusiasm towards the work environment and the projects they work on.

5) Scaling-up to handle Large Projects. Because of the nature of agile methods, it may be hard to solve this problem. Drastic modification to the agile methodology may be required to make handling large projects more natural and systematic. Formalizing the idea of "team of teams" or "scrum of scrums" may be a necessary prerequisite to enable scaling-up agile projects in a smooth way. A lot of research is needed in this area to be able scale-up to handle large projects but at the same time try to preserve the agile spirit and core concepts.

6) High complexity of the system integration process. The best solution here is to use CASE tools that aid in the integration process and track versions of components. Few CASE tools tailored to agile methods have started to appear on the market such as JIRA Software by Atlassian.

7) Determining a project budget upfront. Though in agile methods it is impossible to have an exact budget estimate, it may be possible to come up with reasonably correct estimate if we limit the variability of the requirements. One can think of demanding that 75% or more of the requirements be specified, detailed and finalized before starting the project and allowing up to 25% to be identified, modified, or added later. This gives a better guideline for estimating the budget. Again, more research is needed here.

Overall, it seems the time is ripe for blending best practices from agile methods and from traditional methods to come up with a new model of software development. The new methodology should try to avoid many of the shortcomings of agile methods as well as many of the shortcomings of traditional methods, which requires loosening some restrictions in both worlds. This approach is sometimes referred to as *hybrid* software development. Examples of pioneering research in this area can be found in [20, 21]. More research is needed in order to crystallize and identify the nature and characteristics of such a hybrid software development methodology.

V. Conclusion

Agile methods have proven over the years that they provide many advantages over traditional methods in the area of software development. Enabling the user to modify

requirement or add new requirements after the start of a project, providing the user with working subsystems at early phases of a project, and emphasizing a closer interaction between the development team and the user are some of these advantages. However, there are many issues pertaining to agile methods that act as barriers to its adoption by a wider community of software developers. Some of these issues are practices that can easily be improved, whereas others are deeply rooted in the methodology itself. In this paper, many of these issues are highlighted and a brief description of each one is provided. Furthermore, the paper proposed possible solutions/guidelines on how to deal with these issues in order to minimize their negative impact. This, in the end, will contribute towards improving the quality of software products developed using agile methods, which results in increased customer satisfaction. Consequently agile methods are expected to gain more momentum.

REFERENCES

[1] Wang, Lizhe, Rajiv Ranjan, Joanna Kolodziej, Albert Y. Zomaya, and Leila Alem. "Software Tools and Techniques for Big Data Computing in Healthcare Clouds." *Future Generation Comp. Syst.* 43 (2015): 38-39.

[2] Alashqur, Abdallah. "Mapping Data Between Probability Spaces In Probabilistic Databases." International Journal Of Database Management Systems (IJDMS), Vol. 7, No. 3, pages 1-12, June 2015.

[3] Dvorak, Carl, Khiang Seow, and Charles Young. "Healthcare Information System with Clinical Information Exchange." U.S. Patent Application No. 15/131,738.

[4] Zhai, Huixia. "Application of Information Construction in University Financial Management." *2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*. IEEE, 2015.

[5] Ian Sommerville, Software Engineering, 10 edition, 2015. Publisher: Pearson,

[6] Balaji, S., and M. Sundararajan Murugaiyan. "Waterfall vs. V-Model vs. Agile: A comparative study on SDLC." *International Journal of Information Technology and Business Management* 2.1 (2012): 26-30.

[7] Elghondakly, Roaa, Sherin Moussa, and Nagwa Badr. "Waterfall and agile requirements-based model for automated test cases generation." *2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS)*. IEEE, 2015.

[8] Hasnine, M. N., Chayon, M. K. H., & Rahman, M. M. (2015). A Cost Effective Approach to Develop Mid-size Enterprise Software Adopted the Waterfall Model. *World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering*, *9*(5), 1140-1149.

[9] Lassenius, Casper, Torgeir Dingsøyr, and Maria Paasivaara, eds. *Agile Processes, in Software Engineering, and Extreme Programming: 16th International Conference, XP 2015, Helsinki, Finland, May 25-29, 2015, Proceedings*. Vol. 212. Springer, 2015.

[10] Brhel, Manuel, Hendrik Meth, Alexander Maedche, and Karl Werder. "Exploring principles of user-centered agile software development: A literature review." *Information and Software Technology* 61 (2015): 163-181.

[11] S. Ambler. Agile software development at scale. *Balancing Agility and Formalism in Software Engineering*, pages 1–12, 2008.

[12] Brenner, Richard, and Stefan Wunder. "Scaled Agile Framework: Presentation and real world example." *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*. IEEE, 2015.

[13] W. Royce. Managing the development of large software systems: Concepts and techniques. In *Proceedings of IEEE WESCON*, pages 328–339. IEEE CS Press, 1970

[14] Mahnič, Viljan. "Scrum in software engineering courses: an outline of the literature." *Global Journal of ngineering Education* 17.2 (2015).

[15] M. Fowler and J. Highsmith. The agile manifesto. *Software Development*, 9(8):28–35, 2001.

[16] Rubin, Kenneth S. *Essential Scrum: a practical guide to the most popular agile process*. Addison-Wesley, 2013.

[17] Akif, R., and H. Majeed. "Issues and challenges in Scrum implementation." *International Journal of Scientific & Engineering Research* 3, no. 8 (2012): 1-4.

[18] Almseidin, M., Khaled Alrfou, Nidal Alnidami, and Ahmed Tarawneh. "A Comparative Study of Agile Methods: XP versus SCRUM'." *International Journal of Computer Science and Software Engineering (IJCSSE)* 4, no. 5 (2015): 126-129.

[19] Scheerer, Alexander, Tobias Hildenbrand, and Thomas Kude. "Coordination in large-scale agile software development: A multiteam systems perspective." In *2014 47th Hawaii International Conference on System Sciences*, pp. 4780-4788. IEEE, 2014.

[20] Hayata, Tomohiro, and Jianchao Han. "A hybrid model for IT project with Scrum." In *Service Operations, Logistics, and Informatics (SOLI), 2011 IEEE International Conference on*, pp. 285-290. IEEE, 2011.

[21] Batra, Dinesh, Weidong Xia, Debra VanderMeer, and Kaushik Dutta. "Balancing agile and structured development approaches to successfully manage large distributed software projects: A case study from the cruise line industry." *Communications of the Association for Information Systems* 27, no. 1 (2010): 21.