[1.]Amel DŽANIĆ, [1.]Amel TOROMAN, [2.]Alma DŽANIĆ

AGILE SOFTWARE DEVELOPMENT: MODEL, METHODS, ADVANTAGES AND DISADVANTAGES

[1.]University of Bihać, Faculty of Technical Engineering, Ulica Irfana Ljubijankića bb, 77000 Bihać, BOSNIA & HERZEGOVINA
[2.]University of Bihać, Faculty of Economics, Ulica Pape Ivana Pavla II/2, 77000 Bihać, BOSNIA & HERZEGOVINA

Abstract: The only constant in today's world is a change and therefore the changes are an unavoidable factor also from the aspect of software production. Poor reaction to changes in traditional software development methods has led to the emergence of a new agile philosophy, which has embraced change and put it in its focus. The agile approach essentially implements the set of values and principles given in the agile manifesto. This paper presents an agile model that can be recognized in any agile method. Furthermore, the paper presents the several agile methods that are often used today. The advantages and disadvantages of these methods are stated, and software development through each of these methods is described. The result of the paper is the discussion on the use of these methods, as well as the main problems and the possible directions of development of these methods is addressed.

Keywords: Agile software, development, advantages and disadvantages

INTRODUCTION

Software has major impact in modern life and business, and therefore it is very important to study and research those methods for software developments. Thus, until the end of the 90s of the last century, the software process, i.e. software development, was exclusively based on traditional methods of software development, which were characterized as "heavy" methods [1]. This software process generally consisted of the following steps [1]:

- The project plan must be completed first – that is why these methods are also called plan–driven methods
- All software requirements must be written down – i.e. requirement specification
- The whole project completed and met requirements
- Created source code that implements project documentation with all requirements
- Fully test the software to see if it meets all the requirements

The biggest problem with traditional methods of software development has been the way they respond to changes in requirements, i.e. the traditional way of software development did not respond well to the changes. There have been situations where changes in software specifications (requirements) have required large and expensive interventions on developed software [2].

A lot of time was spent on software development, so the customer received software that was already obsolete in some areas, and these areas required modifications. Because of all this, there was a need for the so–called an easy process where the main goal was to accelerate software development and successfully respond to changes in requirements. These methods are called agile software development methods.

The table shows the main differences between agile and traditional methods of software development

Table 1. Differences between agile and traditional approach of software development [2]

Parameter	Traditioanl methods	Agile methods
Simplicity of modification	Hard	Light
Development approach	Predictable	Adaptive
Development orientation	Orientation on process	Orientation on customer
Size of project	Large	Small or Medium
Planning scale	Long term	Short term
Mode of management	Command and control	Leadership and cooperation
Learning	Continuous learning with development	Learning is secondary to development
Documentation	High	Low
Type of organization	High income	moderate and low income
Number of employees	Huge	Small
Budget	High	Low
Size of team	Medium	Small

AGILE APPROACH OF DEVELOPING SOFTWARE

Adaptable, "light" methods have been created and promoted through the official agile alliance made up of 17 software engineers and consultants, the famous Agile Manifesto was created in 2001. In this software development philosophy, a set of 4 values and 12 principles reflect the essence of agility, and is available on the website https://agilemanifesto.org/ [3].

— Agile model

There are a very large number of agile methods that fully support the principles and values given in the agile manifesto. It is possible to determine the general form of the agile development process in four steps as show on picture 1 [4]:

- Project selection and approval – is the first phase where team consisting of developers, managers, and customers establishes the scope, purpose, and requirements of the product.

- Project initiation – is a second phase where a working architecture of the system is created, which is discussed by all stakeholders with the necessary deadlines and working frameworks.
- Construction of iterations – is third phase of this model in which iterations are made, i.e., this phase consists of planning and building the iterations. In other words, there is a successive incremental process that results in software that meets the evolution of user requirements. This is a consequence of the close collaboration of all stakeholders, and in this way the quality of the software is most effectively ensured. Also, each iteration of the software needs to be tested.
- Product release – is the final phase of this model. In this phase, the final software testing takes place, as well as work on necessary corrections and software documentation. Then product is realised, and end user training begins. Also, the working team may stay to maintain and improve the project as well as user support.

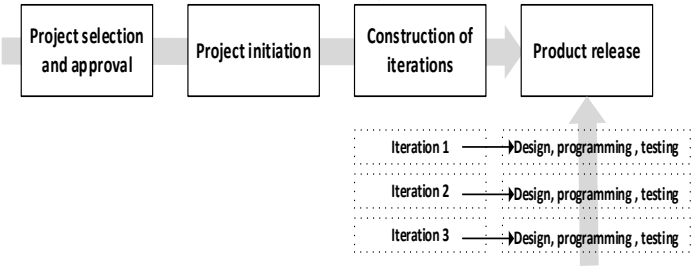

Figure 1. Agile model – general form of the agile development [4]

The advantages of this model are [4]:

- Fast response to the user, the first deliveries are in weeks and not in months. A project always has demonstrable results – the final version of each iteration is usable software.
- Great flexibility, very often combined with other existing models.
- Due to the use of feedback loop and small realises, a high–quality product is created with a high degree of client satisfaction.
- Developers are more motivated – they prefer to produce usable solutions, and they don't like to compile documentation.

The disadvantages of this model are [4]:

- They are problematic for larger projects. There are discussions about their applicability to larger projects.
- The documentation is questionable. Since the documentation is on the back burner, the question arises as to whether the necessary documentation will be compiled at all. This is a big problem, since documentation is an integral part of what developers produce.

AGILE METHODS

— Extreme programming – XP

The extreme programming method – XP is an agile software development method developed by Kent Beck in 1996. The XP method represents a lightweight, flexible, disciplined, low–risk approach to software development that has the ability to manage unclear or rapidly changing requirements. The XP method emphasizes teamwork, therefore managers, developers and clients are part of the team. Team size is one of the limitations of extreme programming, as this method is considered to be suitable for small and medium–sized teams, reportedly anywhere from 2 to 12 team members, although projects with 30 members have been reported successful [5][6].

The life cycle of XP method has six phases and they are as follows [6]:

- The research phase deals with requirements modelling and system architecture. In this phase, user requirements, tools and technology are defined. A schedule of system versions is created, i.e. software releases are planned. Based on the schedule, plans are made for each individual iteration. The user writes user stories that represent the software specification [7]. For each user story, it is necessary to create at least one test that will confirm the correctness of the user story [5]. According to the assessment, the user story should be implemented within 3 weeks, and if it takes longer, it should be divided into several stories.
- In the Planning Phase, priorities for the implementation of user stories are set, and the content of the first small release of the software is agreed upon. The developer makes an estimate of how much effort and time each user story requires, and then an implementation schedule is agreed upon. The first small release of the software should be completed within two months. If the research phase is done well, then the planning phase should last a few days. The basic goal of planning a software release is to find software functions, and to make a schedule for delivering these functions [8]. During the planning phase, the team size, schedule, and who owns the code and working hours are determined [9].
- The iterations to release phase includes basic development activities such as: coding, testing and integration [9]. This is an iterative phase where each iteration can last from one to four weeks. In the first iteration, the stories that make up the structure of the entire software architecture were chosen [7]. After coding, functional testing is performed, and if it is successful, then the code is integrated. If it happens that the code does not fulfil the request, then the so–called refactoring. Quick meetings are used to review development progress or to resolve any issues should they arise. After the final iteration of the code, the production phase begins.
- Production phase – software is delivered in small releases. A small piece of planned software is released that implements some business need or function. Frequent releases allow XP to build the desired system incrementally. The duration for one release is from one to four weeks, and it can contain a certain number of

iterations. In order to check whether the software is ready for the production phase, tests must be performed [8].

≡ Maintenance phase – the software continues to evolve for some time. In the maintenance phase, certain new functionalities are created, while the old ones continue to work [7]. It can also lead to the introduction of a new software architecture, but then the team must be much more attentive to the software in use [9].

≡ Death phase – there are two possible situations in which the software reaches this phase. The first reason is that all software functionalities that users need have been developed, and users are satisfied, and there are no more user stories to implement [9]. Then it is time to carry out the final release of the software, and approach the creation of software documentation. Another reason is that the system does not provide the desired outputs, or if it becomes too expensive for further development, then it is better to stop the software development, which is called entropy death of the system.

The main advantages of the XP method can be summarized in the following few points [2]:

≡ Incremental development is supported through small and frequent software releases.

≡ Improving productivity through a feedback mechanism.

≡ Maintaining simplicity through constant refactoring.

≡ Improving quality through the creation of automated tests before installing functionality.

On the other hand, the method also has its disadvantages, namely [2]:

≡ Reduced capability for distributed teams as it focuses on community and co–location.

≡ This approach to software development requires additional training for newly joined team members.

≡ XP depends on informal documentation such as user stories, code, etc.

≡ The practice of user involvement is effective, but on the other hand, it is also very stressful as well as quite expensive because it can keep the customer away from his real work for a long time.

— SCRUM method

Scrum represents an agile method of software development that focuses on the management of the iterative process, instead of individual technical approaches. The Scrum method was developed to manage the system development process. The basis of this method is an empirical approach that applies the ideas of industrial process management theory to system development, resulting in an approach that reintroduces the ideas of flexibility, adaptability and productivity. This approach does not define any specific software development techniques for software implementation. Scrum concentrates on how team members should function to produce a system flexibly in a constantly changing environment [10].

The lifecycle of SCRUM method has three phases as follows [2]:

≡ The initial phase is the outline planning phase in which the general goals of the system being designed and developed are stated. The project team, necessary tools and resources are also defined. A PBL (product backlog) is generated, which is used to document requirements in the form of user stories and functionality. The requests are then analysed and given specific priorities, and the assessment of the work required for each request is carried out by the product owner, who is also responsible for maintaining a visible and transparent PBL [10]. The PBL is subject to continuous updating since user requests are implemented incrementally, and also during software development there may be a change in the priority of user requests. It is important to note that the documentation is never complete and is supplemented during the process itself in the so–called Sprint Backlogs.

≡ Sprint phase is a time period of one month or less (most often two weeks) in which one increment of the iterative process is made [10]. The advantage of this way of working is that it is possible to deliver part of the software product to the client at the end of the sprint and to adapt part of the product to new requirements in the next cycle based on his suggestions. In this way, it is possible to learn in the course of the work, and not at the end, based on experience and introduce changes and improvements that give a better end result, that is, a good software product with which the client will be satisfied. Sprint contains the following activities, which can be said that correspond to the traditional stages of the life cycle [10]:

❖ Sprint Planning – represents a meeting that lasts a maximum one day and very often shorter at which tasks from the so–called Product Backlog are ordered by priority from the highest to the lowest. The so–called Product Backlog represents the place where the functionalities are defined by the product owner. In this activity not all stakeholders are involved, and it is estimated what needs to be done for these tasks and how many of them can be done during the engineering work.

❖ Sprint – is a time–limited period, from 2 to 4 weeks, in which tasks taken from the Sprint Backlog are executed. In short, these tasks are coded, tested, integrated and documented. The goal of this agile methodology is that at the end of each sprint, functionalities have been developed that can go into production.

❖ Daily Sprint Meeting – is held every working day at the same time and lasts a maximum 15 minutes. Any deficiencies or obstacles in the system development process or engineering practices are sought, identified and removed to improve the process.

❖ Sprint Review – the Scrum development team and the Scrum Master present the results of the sprint, i.e. the work product increment presents to the product

owner, customers and users. Participants evaluate the product increment and decide on the next activities. The preview can even change the direction of the system being built.

- ❖ Sprint Retrospective Meeting – is a meeting where the team and the Scrum Master answer the questions: What was good in the last sprint? What was not good in the last sprint? What to do to work better? After that, the new cycle starts again with a new sprint planning.

≡ The project closure phase occurs when the user requirements are met and the required software goals are achieved based on the dialogue between the product owner and the team. The latest version of the product is ready for "release" and distribution, and the user documentation is being completed [2] [10].

The advantages of the SCRUM method are as follows [2]:

≡ The software product is divided into a smaller set of manageable and understandable components shared by teams resulting in increased communication and shared knowledge.
≡ Transparency – the development team has visibility into everything, including communication and feedback from product owners through the various meetings held throughout the development process.
≡ Self–organization – all teams share responsibilities.
≡ Self–retrospective – provides a tool to self–assess goals achieved against those needed after each iteration or sprint, increasing productivity through continuous testing.
≡ Simple process.
≡ Ignoring any change in sprint duration by prohibiting the addition of any functionality to the sprint, allowing the team to finish their current in–progress functionality.

The disadvantages of the Scrum method are as follows [2]:

≡ Violation of responsibility may occur, since there are no precisely defined responsibilities for each team member.
≡ SCRUM does not prescribe any specific practices, work methods or any guidelines on engineering practices.

— Feature driven development (FDD) method

This method manages short incremental iterations that lead to functional software. The basis of the FDD method is the management of software development based on a list of required characteristics of business needs. The FDD method is a highly adaptive software development method that can account for late changes in software requirements. The main focus of the FDD method is the delivery of high–quality outputs during all phases of the development process [2] [9]. The life cycle of the FDD method contains five sequential processes that are performed incrementally and iteratively, and in this way the final software is delivered. The listed steps are [9] [11]:

≡ Development of the overall model: In this step all team members and experts define the required context and scope of the overall project. Different teams and experts can generate many models, which are then reviewed and the optimal model is selected based on the requirements.
≡ Creation of a list of characteristics: based on the model and the required documentation an overall list of characteristics is created, i.e. a specification for the software. A list of characteristics is created, which is grouped into sets by subject areas.
≡ Planning by characteristics: a high–level plan is created based on a previously approved list of characteristics. The plan is created as an order based on the client's priorities and depending on the characteristics. The master developer assigns characteristics to a specific developer called the class owner.
≡ Design by characteristics: is an iterative step where each iteration can last up to two weeks. The master developer and the class owner create a package project for each class with sequence diagrams. Package design and diagrams are reviewed before approval.
≡ Build by Features: Designs are implemented, after which the code will be reviewed and tested. This is also an iterative step like design by features. After all the iterations are done, the developed features will be published in the main version, then a new set of features is launched, and so on.

The FDD model has certain advantages which can be summarized as [9]:

≡ FDD is a highly adaptable method that can take into account late changes in client requirements.
≡ Delivers high quality results after each stage.
≡ The results of each iteration can be delivered within one to four weeks which helps as we can have quick feedback from clients.

However, there are certain limitations and disadvantages when using the FDD model, such as [9]:

≡ There are no guidelines on requirements gathering, analysis and risk management.
≡ The FDD model requires an expert team with a high level of design and modelling skills.
≡ The FDD model does not take into account issues of project criticality.

OVERVIEW OF AGILE METHODS AND DISCUSSION

Agile methods are essentially iteratively incremental methods. Iterative nature is achieved by using user feedback with which a certain functionality is "polished" until the moment it meets user requirements. Small software releases are given always which is also iterative nature. The first release of software is essentially software with a minimum number of functionalities that it can work with. In this way, a quick response to the customer is offered, the first deliveries are in weeks, not in months.

The project always has demonstrative results – the final version of each iteration is usable software, i.e. each subsequent release adds some new functionality to the software. Due to its approach where users together with the

development team choose which functionalities have higher priority, and incremental and iterative nature, this approach is adaptive and customer–oriented, not the development process–oriented. The agile methods that have been processed are suitable for small to medium–sized projects because agile methods also prefer small to medium–sized teams of highly trained professionals [5]. Due to its informal nature, documentation is weak and often not even created. Due to adaptability, there are no long–term plans. It can be said that there are only short–term plans. In agile teams there are so–called self–organizing teams that, through cooperation, communication and leadership, achieve set goals as opposed to the command and control which could be found in the traditional approach [11].

For large, long–life systems developed by a software company for an external client, using an agile approach presents a number of problems. The informality of agile development is incompatible with the legal approach to defining contracts that is commonly used in large companies. When a system user uses an external organization for system development, a software development contract is concluded between them. A software requirements document is usually part of that contract between the customer and the vendor. Because requirements and code development are intertwined, fundamental to agile methods, there is no definitive requirements document that can be included in the contract.

Agile methods are best suited for new software development, not software maintenance. However, most software costs in large companies come from maintaining their existing software systems. If maintenance involves adapting and changing systems in response to new business requirements, there is no clear consensus on the suitability of agile methods for software maintenance. Three types of problems can occur, especially with maintenance [12]:

- Lack of product documentation – agile methods in most cases do not care so much about documentation. The collection of requests is conducted informally and gradually. Shorter meetings and face–to–face communication are also preferred. Therefore, there is no coherent document of requirements, unlike traditional methods. This further leads to the problem of how to subsequently maintain and upgrade the system. However, this is a big problem if the continuity of the development team cannot be ensured.
- Keeping customers involved – initially the customer's representatives in the development team will be fully engaged. But as time goes on, their interest is lost. Thus, there will have to be adjustments and changes to try to get interest back.
- Development team continuity – a fundamental aspect of agile methods is that team members know the system without having to consult the documentation. If the members of this team leave, then the knowledge about the system is also lost. New members who come to the team can have great difficulties while understanding this knowledge of the system. In addition, programmers prefer to develop new software rather than maintain existing ones. Therefore, even when the intention is to keep the development team together, people leave if they are assigned maintenance tasks.

As mentioned before, agile methods are not very suitable for building large systems. That is why it is necessary to perform the so–called scaling agile methods. The fundamental purpose of scaling agile methods is their integration with a planning approach. To solve these problems, most large "agile" software development projects combine practices from plan–driven and agile approaches [12].

Recently, with the arrival of internet of things – IoT, a wide range of devices are integrated into software systems, large amounts of data become available for analysis, virtual reality systems are developed, and therefore there are increasing expectations of intelligent solutions. These new technologies have completely renewed interest and revealed new possibilities in exploring the full potential of both artificial intelligence and user interface. Because of this, a strong role of agile software development in the emergence of new technologies is predicted [13].

CONCLUSIONS

Certainly, agile methods represent an unstoppable trend in software development. As stated, they are very successful in the realization of small to medium software projects, with small teams. They have successfully responded to the changes that and have enabled software engineers to deliver quality software that meets user requirements.

Agile methods follow the presented agile model. The agile model is valid for all agile methods that are iteratively incremental by its nature. This very nature ensures success of agile methods in the fight against changes in requirements and the release of better software versions.

There is great discussion about their performance on large projects in terms of documentation, consistency of the team as well as the size of the team itself. Furthermore, teams must primarily be in the same locations due to the informal nature of the agile process, which leads to problems with distributed teams. Of course, large systems are also long–lived, so they must have maintenance as well as upgrades. Maintenance and upgrading are processes that require the existence of documentation. Agile methods do not really focus on formal documentation, even proponents of agile methods say that documentation is unnecessary, because documentation is often not updated and is therefore worthless. If there is no continuity of the team, and if there is no documentation, the big question is how such software systems could be maintained and upgraded. Because of these problems, agile methods are scaled, i.e. there is a process when the practices of planning and agile methods are combined to a certain extent.

Since the agile approach has fully justified its existence in the struggle with continuous changes and the production of high–quality software, it will certainly find its place in the future.

Note: This paper was presented at International Conference on Applied Sciences – ICAS2022, organized by University Politehnica Timisoara, Faculty of Engineering Hunedoara (ROMANIA) and University of Banja Luka, Faculty of Mechanical Engineering Banja Luka (BOSNIA & HERZEGOVINA), in May 25–28, 2022, in Banja Luka (BOSNIA & HERZEGOVINA)

References

[1] Braude, E. J., & Bernstein, M. E. 2016 Software engineering: modern approaches, Second Edition, Waveland Press, Inc., Illinois, USA
[2] Alsaqqa S., Sawalha S., Hiba A. N. 2020: Agile Software Development: Methodologies and Trends, International Journal of Interactive Mobile Technologies – Vol. 14 NO. 11
[3] Manifesto for Agile Softvware Development, https://agilemanifesto.org
[4] Isaias P., Issa T. 2015: High Level Models and Methodologies for Information Systems, Springer Science+Business Media, New York, USA
[5] Džanić A., Dujmović D. 2011: Razvoj aplikacija korištenjem agilne metodologije Extreme programming – xp, 8th International Scientific Conference on Production Engineering RIM 2011, Velika Kladuša, BiH, September 29 –October 1, pp 401 – 407
[6] Mnkandla E., and Dwolatzky B. 2004: A survey of agile methodologies, The transactions of the SA institute of electrical engineers, Vol. 95 NO. 4
[7] Beck K. 2000: Extreme programming explained: embrace change, Addison–Wesley Professional, USA.
[8] Abrahamsson P., Salo O., Ronkainen J. and Warsta J. 2002: Agile software development methods: Review and analysis, VTT publ., Finland.
[9] Anwer F., Aftab S., Shah S. S. M., Waheed U. 2017: Comparative Analysis of Two Popular Agile Process Models: Extreme Programming and Scrum, International Journal of Computer Science and Telecommunications, Volume 8, Issue 2
[10] Schwaber K. 2004: Agile Project Management with Scrum, Microsoft Press,
[11] Zaimović T., Kozić M., Efendić A., Džanić A. 2021: Self – Organizing Teams in Software Development – Myth or Reality, TEM Journal. Volume 10, Issue 4, pages 1565–1571
[12] Sommerville I. 2016: Software Engineering, Tenth Edition, Pearson Education, Inc
[13] Hoda, R., Salleh, N., & Grundy, J. (2018). The rise and evolution of agile software development. IEEE Software, 35(5), 58–63

ISSN: 2067–3809