International Journal of Computer Applications (0975 – 8887)
Volume 109 – No. 1, January 2015

The Sway of Agile Processes over Software Maintainability

Balraj Kumar
Department of Computer Applications
Lovely Professional University, Punjab, India

ABSTRACT

Software engineering is escalating in different dimensions at precipitous speed and coercing the developers and researchers to find new ways for easy and swift software development. The induction of agile methodologies is one such step towards achieving these goals. Today the agile processes and techniques are very extensively accepted and fostered in the software construction by the software industry and considered as the valuable tools for hasty software development. This paper throws light on different agile software development methodologies and their sway over the software maintainability. It emphasizes on the impact as perceived by the agile fans and foes. The paper also aims to provide an insight into the effect of agile methods when used as maintenance processes to enhance the software maintainability.

Keywords

Agile methodologies, agile software development, sway of agile processes, software maintainability, maintenance

1. INTRODUCTION

Now software has become an integral part of our daily life. Due to its indispensability, it has managed to make a unique place in the sophisticated society and is finding its own ways to change our frame of mind, culture, knowledge base and working environment. When the software is modified, enhanced and adapted to the changing environment, it becomes more difficult and drifts-away from its original design; it lowers down the software quality. This is the only reason that the key portion of the total software development cost is sacrificed to software maintenance [22]. The need of hour is to use the state of the art software development tools and techniques that could produce highly maintainable software to satisfy the ever changing customer requirements.

Software development practice is actually a framework in which the development process is carried out. There are two types of methodologies available for software development: First, family of plan-driven methods, the heavyweight processes (waterfall etc.), originate from the academic world [17] and use mathematical models for software development and have been taken into use in the industrial world as well. Second, family of agile methodologies, the lightweight processes (XP, Scrum etc) is significantly younger and developed in industrial companies rather than the academic world and embrace changes at any stage rather than trying to avoid them. Agile ratifies the planning for change and thus improving the software quality and especially the maintainability [4].

The main aim of this paper is to critically examine the sway of agile development methodologies on the maintainability of software systems. For this purpose, the paper has been organized into several sections. Section-2 describes about the literature of agile development and the maintainability. Section-3 highlights the relationship between agile methods and maintainability. Section-4 demonstrates that the agile practices can be employed in maintaining software systems too. Section-5 stresses upon the perspectives of agile protagonists and antagonists. Finally, conclusion is presented in section-6.

2. BACKGROUND

The present section is divided in two parts. The first part explores the historical and recent literature of agile development and covers the various aspects of agile processes and tools. The next part discusses about the maintainability in brief and states about the various factors affecting the maintainability along with the metrics needed to measure a software system’s maintainability.

2.1 Agile Software Development

The term ‘Agile Software Development’ (ASD) attempts to serve as an umbrella for large number of processes, practices and methodologies used for software development and project management [4]. But what is the meaning of being Agile? According to Jim Highsmith, being Agile means being able to deliver quickly, change quickly and change often [11]. It was introduced in 2001 in the Agile Manifesto [4]. This statement values:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Here, both the left and the right side of each of the core values are vital, but the items on the left are valued more than the ones on the right. Incremental, cooperative, straightforward and adaptive development is called the agile software development [5]. Its main goal is to deliver working software in small cycles/iterations. The steps for ASD are shown in the following figure:

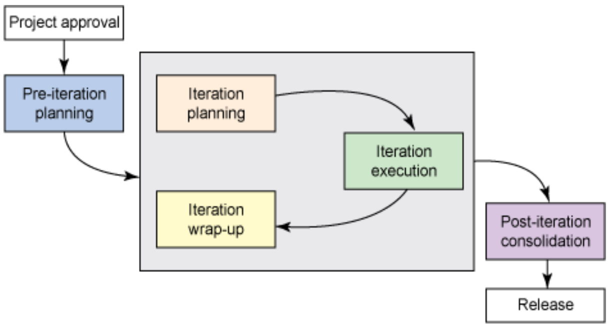

Figure 1: Agile Software Development

In *fig-1*, the agile process begins with the project approval and moves through the pre-iteration planning and repeats the cycle of planning, execution and wrap-up again and again depending on the project requirements, which is further followed by a post-iteration consolidation and at last a release. As ASD is based on iterative and incremental approach it promises to deliver more productivity, better quality and high project success rate in software development.

2.1.1 Agile Processes - A Brief Portrayal
Though there are many agile processes available, but here only few are being discussed which form the basis of this paper. These include:

Extreme Programming (XP) is envisioned to hone the quality of software and agility to the changing needs of customer, industry and market. Initially, it was started simply as an opportunity to get the work done [15] with the processes and practices which have been found effective during the development of a software system. Scrum is an iterative and incremental model for project management and development. Under this, project plans are continuously scrutinized and improved based on the empirical project reality. Analysis, development and testing take place in the 2-4 week iterations, called sprints. Extreme programming and Scrum methods complement each other very well. Where XP is accountable for technical aspects, Scrum is responsible for project planning and tracking [14].

Open Source Software (OSS) development is relatively a novel agile approach of constructing and deploying large software systems on world-wide basis. The OSS approach advocates the source code to be freely available for amendments and its redistribution without any charge. The Object-Mine-Adopt is generally regarded as an agile process because it is more like people-oriented rather than process-oriented and has many features like other agile methods [6]. It is swift, adaptive and self-organizing in nature and used to provide means to construct more maintainable software.

2.2 Maintainability - An Improvement Opportunity
The maintenance effort is the most time-consuming part of software-development-life-cycle which may generally range from 65% to 75% of total time spent on software development [23]. Most of the time is spent during maintenance-phase which significantly affects the cost of the software product. Software can be made highly maintainable by putting more efforts in the initial stages of its SDLC and this may considerably reduce the overall software cost [10]. Software maintainability or simply maintainability is defined as - *“The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment”*[3].

Software maintainability can be viewed as an opportunity to improve the software system the way it is constructed so that it can be maintained more easily. The maintainability literature can be classified in two broad categories: factors swaying the maintainability and the metrics to measure a software system’s maintenance effort.

2.2.1 Factors Swaying the Maintainability
Factors can affect the maintenance effort positively or negatively. Structured design, analysis methodology and lack of application experience are the factors that have a negative impact on the maintenance effort [24], whereas good response time of the hardware for development and proficient staff are the key factors that influence it negatively. Three factors were identified based on literature and empirical observations that significantly influence the software maintainability. They include functionality, development practice and software complexity [25]. Age and size are the two control factors that also directly or indirectly affect the maintenance work. In case of age, fixing errors become more expensive the later if they are found in software system [19], whereas the size of a component or module has a strong impact on the maintenance effort required for making changes in that component [27]. Developer skills and experience are generally ignored, but they also have greater impact on the software maintenance [28]. Program comprehension plays a major role in maintaining a software system [29]. Without complete understanding of the code, it is a big challenge for the maintenance team to perform alterations in software modules i.e, the more difficult a program is to comprehend, the more difficult it is to maintain and the more difficult a program is to maintain the higher will be its maintainability risk [10].

2.2.2 Maintainability Metrics
Maintainability Index (MI) was proposed to determine the software maintainability precisely based on the status of the corresponding source code [22]. The MI is a composite number that depends upon a number of different metrics for a software system. It is based on the Halstead Volume (HV) metric [26], the Cyclomatic Complexity (CC) [25] metric, the average number of Lines of Code per module (LOC), and optionally the percentage of comment lines per module (COM). The higher the MI, the more the system is considered maintainable. The MI agonizes from austere limitations like root-cause analysis, language liberty, ease of computation and control.

Authors in [31] presented a huge number of metrics to measure the maintainability of a software system and elucidated how to consolidate them to form the maintainability index for a software system. They presented all the metrics in a hierarchical form. The hierarchical structure is divided in 3 major components - source code, maturity attributes, and supporting documentation. The leaf nodes represent measurable attributes of a software system. It is amazing that none of the metrics belong to the skill / experience of the software developers. But other research has admitted that the maintenance effort can decrease by engaging the experienced developers [32]. Authors in [7] proposed a viewpoint for assessing adaptive maintenance effort in terms of person hours based on the projected number of LOC to be modified and/or the number of operators to be altered.

3. CONNECTION BETWEEN AGILE METHODS AND MAINTAINABILITY
The major problem being faced by software industries is that they spend more on maintaining existing software than they do on novel software development. So, if the agile processes are to be successful, they have to support software maintenance with original development. There are two basic concerns involved that need to be addressed: First, are systems that developed using an agile approach maintainable with emphasis on the development process and reducing documentation part? Second, can agile methods be used effectively for developing a new system/project keeping in mind the unstable customer needs? Since the maintainability of existing software system is related to how the system supports easy error correction in future, addition of new feature and further related activities of software improvement, so it cannot be measured directly. For this, the maintainability

is stated as a set of measureable properties of the system which all influence the software maintainability. There are large number of metrics in literature that are used for measuring the maintainability of a software system and an approach to unite these measurements into a single MI [31].

Since no study has been found that specifically deals with the maintainability of a software system developed using agile approach, so few empirical studies about development teams that use agile methods were carried out. In a study, the development team adopted two agile processes (XP and Scrum) together and reported low defect density and improved team communication [14]. In another study, XP and Scrum together were found to be the most effective combination of agile methods [21]. Identification of few bugs reduced overall maintenance work. The major finding of another study that used XP for development is the low code complexity, whereas the complexity control work is relatively high [30].

4. AGILE PROCESSES FROM MAINTAINABILITY PERSPECTIVE

Agile methods can also be used as software maintenance processes to provide better quality software. Following are some of the illustrations where agile processes can be witnessed from different maintainability point of views:

4.1 Extreme Programming (XP)

Extreme Programming, one of the popular agile methods, could be used for software maintenance as proposed in [20] instead of the traditional approaches like waterfall paradigm. The extreme programming practice revealed that refactoring refined the quality and constancy of two programs written in Java [1]. When XP was again used in another research [16], it demonstrated that the coding standard of XP, regular testing and mutual program ownership were the most appreciated and the most challenging practices to adopt. Also, the productivity of maintenance was increased by three times.

4.1.1 Benefits of using XP

The use of XP as a software maintenance model may help:

- Reduce complexity by removing code not in use
- Enforce compliance to source code guiding principles
- Promote ownership and obligation for programming style
- Provide a proactive approach to problem solving
- Refactor to reduce code size by more than 40%
- Eliminate code complexity and stagnation
- Apply user stories to request bug fixes
- Fully automate the build and test process
- Reduce staff and productivity goes up by three times
- Provide a precise set of rules to govern the merging of solutions and augmentations
- Improve the software quality by 67%

4.2 Scrum

Scrum is built on the core principles of providing an iterative and incremental approach for effective software development. Initially, it was designed for software development only, but now it is equally effective in software maintenance and the overall project management. Scrum is the perfect agile methodology for the situation which needs quick reaction to changes in the customer requirements. Each project is special and unique and that’s the underlying philosophy on which Scrum is built. Software maintenance (aka backlog grooming) is taken care by the Scrum process. Like other scrum stand-up meetings, backlog grooming meeting is also crucial from the software maintenance point of view and must be attended by the team, stakeholder and the ScrumMaster. During the maintenance meeting, everyone helps prepare the scrum backlog for the sprint planning meeting like adding new stories and epics, extracting stories from existing epics and estimating effort for existing stories. When the backlog items satisfy the acceptance criteria and are estimated by appropriate team members, the planning process will not be tense or long.

4.3 Open Source Software (OSS)

The OSS development methods are another type of agile methods that can be used as software maintenance processes. That’s why they need not to have separate traditional sequential methods for software maintenance. For this purpose, a lot of research has been carried out. The quality of software was 20% higher in a study of 6 OSS products representing six million lines of code [2]. A research study of two OSS products and 4 industry projects showed that OSS projects had more than 5 times lesser faults [9]. In another study of 53 OSS products, total sixteen million LOC, the proportion of major to minor contributors did not increase the size, complexity, or number of changes [13]. Lastly, in a study of 75 OSS projects, the use of the OSS development methodology not only enhanced quality of software, but also did not raise the development cost [18].

4.4 Observe-Mine-Adopt (OMA)

The OMA is an agile approach that helps organizations to identify and adopt software processes to enhance the maintainability and also emphasizes to ascertain highly maintainable modules, difficult to maintain modules and best practices to develop easy to maintain modules [6].

4.4.1 Benefits of using OMA

The OMA approach provides the following benefits:

- The OMA operations are clearly defined and close to the natural ways of human thinking.
- The methodology overhead is trivial and can be performed by holding a few OMA meetings.
- The OMA supports the development team to get the feedback so that people can quickly adapt to changes.
- The OMA primarily depends on human experiences and is well suited for empirical methods.

In a study [8] to determine the maintainability, the OMA was applied on two real-world projects. After integrating the specified changes, one minor and one major change was observed in applications. A minor change did not affect any core requirements for the specification and did not require any prior knowledge of the domain area, whereas a major change affected the core processing requirements of the system and necessitated domain familiarity and strong knowledge of the system. In a different study, another group of 3 projects written in Java used the OMA model to assess the software maintainability. The original code of all 3 projects along-with their modified versions (after a minor or a major change) was intensively scrutinized. The major and minor changes were similar in scope and size to those found in [8] study.

5. VANTAGE POINT OF AGILE FANS AND FOES

According to supporters, agile methods are the best practices that have happened to software development in the recent years, whereas the critics see them as a backlash to software engineering and compare them to hacking. In the light of critical views and annotations by both fans and foes, this section presents the different representations found in literature [12].

5.1 Affirmative Representations

According to the advocates of agile methods, these practices are much better for maintenance purposes than the superseded ones and the software developed using agile methods stimulates customer collaboration to extract correct maintenance requirements, development of regular software advancements to deliver competence to maintenance customers quickly, cooperation within the development teams to enforce high maintenance communication and quick response to fast changing customer requirements.

Agile methods provide the means to develop software by undertaking software maintenance in a systematic manner. The people working on the software maintenance usually prefer to work with agile methodologies. They develop a good quality software usually at a nominal cost. With agile methodologies, software maintainability is more effective when constructed with other best practices. These include project management, team empowerment and architecture.

To achieve high software maintainability, the major emphasis is on tracking requirements, design, testing etc. Agile methodologies, not just at the start rather throughout the development process, facilitate the regular improvement in the design and architecture. Likewise, testing must be performed on regular basis and not just at the end. Automating the testing process lowers the cost of testing software regularly. After a long period, it becomes extremely difficult to predict what changes are needed. With regular and proper customer feedback, software must be amended to satisfy the client requirements. The software must be flexible enough to incorporate new changes.

5.2 Negative Representations

Software maintainability is one of the quality attributes that must be put up into the structural design of a software system. The trivial role played by the software architecture in agile software development is a barrier to long-term software maintenance. The agile practices seem to work well in a set of ideal conditions that do not hold true in many projects like the same team will be maintaining the software throughout its life cycle. Secondly, the new feature addition should not have a major impact on the entire software system. If it does have, there are chances for the team to go back and redesign or refactor the code again.

The purpose of agile methods is to adapt to the ever changing requirements of the customers. They lose the long-term product quality for the sake of achieving short term goals such as marketing period and transient customer contentment. It is evident that requirements may change and the customer perceptions may also get change over time. Software systems usually have long life once they are developed and ready for use. Then what are the benefits gained through agile practices compared to the costs of maintaining a product in the years to come. In agile approach, the original development is exposed to the bare minimum. This may be a good way of keeping tight deadlines, but it is not the best way to save costs. The truth is that the costs of adding quality later are much higher than building in the quality from the start. When the customers actually comprehend and feel the reality, they would not be as excited about the agile campaign as they are.

Inadequacy of system documentation may lead to increased system complexity, poor maintainability, dearth of system familiarity, difficulties to assess impact of change and side effects and chaos in the challenging and complex maintenance tasks. Like system documentation, the process documentation is equally important. Its role is to record all the relevant process steps in order to provide process transparency to all the people in development team so that they can control the process and take right decisions in present and future. Insufficient process documentation makes the monitoring impossible. This is the main reason for losing control over the software processes and systems and for decreasing the chances to improve their quality.

6. CONCLUSION

Agile processes are a set of methodologies that promise swift and quality software development. Agile promoters strongly support and recommend these lightweight processes in comparison to traditional ones, whereas the antagonists, on the other side, have declared the agile methodology as a “license to hack” or “code-and-fix” method. The paper finds the agile methodologies best suited for the projects that have little visibility for future requirements. With their use, decisions can be made throughout the development when the requirements become clearer. Self-organizing teams, close cooperation between business people and developers, pair programming, timely software releases, regular adaptation to unstable needs and refactoring are the key factors for their success. Here the paper has validated that agile processes have significant sway on the maintainability of a software system and these methods can be applied to software maintenance as well. In fact, the software complexity and defects get reduced. The use of agile processes may decrease software maintenance costs, improve productivity and enhance the software quality.

7. ACKNOWLEDGMENT

This paper has come into present shape with invaluable guidance and support of my friends and colleagues. Also, I want to acknowledge my family, parents especially my beloved daughter Jivita who have always been a guiding force and source of inspiration for me.

8. REFERENCES

[1] Alshayeb, M., & Li, W. (2005). An empirical study of system design instability metric and design evolution in an agile software process. *Journal of Systems and Software*, *74*(3), 269-274.

[2] Samoladas, I., Stamelos, I., Angelis, L., & Oikonomou, A. (2004). Open source software development should strive for even greater code maintainability. *Communications of the ACM*, *47*(10), 83-87.

[3] IEEE, Authoritative Dictionary of IEEE Standards Terms, ANSI/IEEE Std. 100. Institute of Electrical and Electronic Engineers, New York NY, 2000.

[4] Fowler, M., & Highsmith, J. (2001). The Agile Manifesto Software Development Magazine.

[5] Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile software development methods: Review and analysis.

[6] Huffman Hayes, J., Mohamed, N., & Gao, T. H. (2003). Observe-mine-adopt (OMA): an agile way to enhance software maintainability. *Journal of Software Maintenance and Evolution: Research and Practice, 15*(5), 297-323.

[7] Hayes, J. H., Patel, S. C., & Zhao, L. (2004, March). A metrics-based software maintenance effort model. In *2011 15th European Conference on Software Maintenance and Reengineering* (pp. 254-254). IEEE Computer Society.

[8] Hayes, J. H. (2002). Energizing software engineering education through real-world projects as experimental studies. In *Software Engineering Education and Training, 2002.(CSEE&T 2002). Proceedings. 15th Conference on* (pp. 192-206). IEEE.

[9] Dinh-Trong, T. T., & Bieman, J. M. (2005). The FreeBSD project: A replication case study of open source development. *Software Engineering, IEEE Transactions on, 31*(6), 481-494.

[10] Kumar, B. (2012, September). A Survey of Key Factors Affecting Software Maintainability. In *Computing Sciences (ICCS), 2012 International Conference on* (pp. 261-266). IEEE.

[11] Highsmith, J. A. (2000). Extreme programming, e-business Application Delivery, vol.

[12] Kajko-Mattsson, M., Lewis, G. A., Siracusa, D., Nelson, T., Chapin, N., Heydt, M., ... & Snee, H. (2006, September). Long-term life cycle impact of agile methodologies. In *Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on* (pp. 422-425). IEEE.

[13] Scotto, M., Sillitti, A., & Succi, G. (2007). An empirical analysis of the open source development process based on mining of source code repositories. *International Journal of Software Engineering and Knowledge Engineering, 17*(02), 231-247.

[14] Fitzgerald, B., Hartnett, G., & Conboy, K. (2006). Customising agile methods to software practices at Intel Shannon. *European Journal of Information Systems, 15*(2), 200-213.

[15] Beck, K. (2000). *Extreme programming explained: embrace change*. Addison-Wesley Professional.

[16] Svensson, H., & Host, M. (2005, March). Introducing an agile process in a software maintenance and evolution organization. In *Software Maintenance and Reengineering, 2005. CSMR 2005. Ninth European Conference on* (pp. 256-264). IEEE.

[17] Plat, N., van Katwijk, J., & Toetenel, H. (1992). Application and benefits of formal methods in software development. *Software Engineering Journal, 7*(5), 335-346.

[18] Capra, E., Francalanci, C., & Merlo, F. (2008). An empirical study on the relationship between software design quality, development effort and governance in open source projects. *Software Engineering, IEEE Transactions on, 34*(6), 765-782.

[19] Shen, V. Y., Yu, T. J., Thebaut, S. M., & Paulsen, L. R. (1985). Identifying error-prone software—an empirical study. *Software Engineering, IEEE Transactions on*, (4), 317-324.

[20] Poole, C., & Huisman, J. W. (2001). Using extreme programming in a maintenance environment. *IEEE Software*, *18*(6), 42-50.

[21] Parsons, D., Ryu, H., & Lal, R. (2007). The impact of methods and techniques on outcomes from agile software development projects. In *Organizational Dynamics of Technology-Based Innovation: Diversifying the Research Agenda* (pp. 235-249). Springer US.

[22] Coleman, D., Ash, D., Lowther, B., & Oman, P. (1994). Using metrics to evaluate software system maintainability. *Computer, 27*(8), 44-49.

[23] Muthanna, S., Kontogiannis, K., Ponnambalam, K., & Stacey, B. (2000). A maintainability model for industrial software systems using design level metrics. In *Reverse Engineering, 2000. Proceedings. Seventh Working Conference on* (pp. 248-256). IEEE.

[24] Albrecht, A. J., & Gaffney, J. E. (1983). Software function, source lines of code, and development effort prediction: a software science validation. *Software Engineering, IEEE Transactions on*, (6), 639-648.

[25] McCabe, T. J. (1976). A complexity measure. *Software Engineering, IEEE Transactions on*, (4), 308-320.

[26] Halstead, M. H. (1977). Elements of Software Science, Operating, and Programming Systems Series. *Elsevier Science*, 7.

[27] Niessink, F., & van Vliet, H. (1997, October). Predicting maintenance effort with function points. In *Software Maintenance, 1997. Proceedings., International Conference on* (pp. 32-39). IEEE.

[28] Banker, R. D., Datar, S. M., Kemerer, C. F., & Zweig, D. (1993). Software complexity and maintenance costs. *Communications of the ACM*, *36*(11), 81-94.

[29] Yau, S. S., & Collofello, J. S. (1980). Some stability measures for software maintenance. *Software Engineering, IEEE Transactions on*, (6), 545-552.

[30] Capiluppi, A., Fernandez-Ramil, J., Higman, J., Sharp, H. C., & Smith, N. (2007, May). An empirical study of the evolution of an agile-developed software system. In *Proceedings of the 29th international conference on Software Engineering* (pp. 511-518). IEEE Computer Society.

[31] Oman, P., & Hagemeister, J. (1992, November). Metrics for assessing a software system's maintainability. In *Software Maintenance, 1992. Proceerdings., Conference on* (pp. 337-344). IEEE.

[32] Kitchenham, B. A., Travassos, G. H., von Mayrhauser, A., Niessink, F., Schneidewind, N. F., Singer, J., ... & Yang, H. (1999). Towards an ontology of software maintenance. *Journal of Software Maintenance*, *11*(6), 365-389.