IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 10, Issue 2 (Mar. - Apr. 2013), PP 87-93
www.iosrjournals.org

Enhancing Software Quality Using Agile Techniques

Amran Hossain[1], Dr. Md. Abul Kashem[2], Sahelee Sultana[3]
[1](Computer Science and Engineering department, Dhaka University of Engineering & Technology, Bangladesh)
[2](Computer Science and Engineering department, Dhaka University of Engineering & Technology, Bangladesh)
[3](Computer Science and Engineering department, Dhaka University of Engineering & Technology, Bangladesh)

Abstract: *Agile techniques may produce software faster as well as enhance software quality so that they fulfill quality requirements of the product. In this paper we have considered some quality factors and we have shown how agile techniques enhance software quality. We have presented an agile development life cycle that showing its software quality support processes. Finally we have shown summarization of software quality evaluation with agile techniques that enhance software quality.*

Keywords- *Agile methods, Architectural spike, Software Quality, Software Quality assurance, System metaphor.*

I. Introduction

Software quality is the degree to which a system, component, or process meets specified a requirement i.e. meets customer or user needs or expectations. Quality consists of those product features which meet the needs of customers and thereby provide product satisfaction. Agile software development is a controversial software engineering method. Many researchers have the knowledge about the benefits of it [1]. Agile is a heavyweight document-driven software development methodology and this methods have gained tremendous acceptance in the commercial arena. Agile approach has two of the most significant characteristics these are as follows: 1) It can handle unstable requirements throughout the development life cycle 2) It deliver products in shorter time frames and under budget constraints when compared with traditional development methods [1]. The Agile approaches are changing the conversation about software development. Agile shifted our attention to small teams incrementally delivering quality software. We need to think about the role of Quality Assurance in Agile Projects. Agility is strategy, release iteration, daily and continuous working software. Agility provides enough rigors to ensure quality, as do traditional development methods, e.g., waterfall model. Here we did not consider waterfall model but we have shown agile methods that is how to achieve quality. We have shown the quality assurance techniques of agile methodology to enhance software quality. Our approach consists of the following parts: 1) Assemble a complete life cycle of the agile model including its supporting processes. 2) Software quality factors with details description 3) Recognize those techniques within agile methods that purpose to ensure as well as enhance software quality. 4) Summarizes after evaluating software quality factors in the particular agile technique. Through applying such an approach, we can systematically investigate how agile techniques integrate support for software quality within their life cycle. The remaining part of the paper is organized as follows: Section 2 presents a short description of software quality, quality factors and quality assurance. Section 3 gives an overview of agile methods life cycle to highlight the reasons why it become fashionable. Section 4 explains a brief description of agile techniques to enhance quality. Section 5 explains evaluation of SQF. Section 6 concludes the paper.

II. Software Quality Factors (SQF) & Quality Assurance(QA)

1.1 Software Quality Factors

Quality defined as high levels of user satisfaction and low defect levels, often associated with low complexity. The quality of software is assessed by a number of variables. These variables can be divided into external and internal quality criteria. External quality is what a user experiences when running the software in its operational mode. Internal quality refers to aspects that are code-dependent, and that are not visible to the end-user. External quality is critical to the user; while internal quality is meaningful to the developer only [2].Table 1 shows a version of the software quality model [3]. This model categorizes 14 quality factors in three steps of the development life cycle: Quality of design, Quality of performance, Quality of adaptation. This model provides a superior structure of reference to recognize software quality. We used this model throughout the study. In this paper we have shown that how to enhance the criteria of quality factors using agile techniques.

Software Quality Factors

Quality of Design		**Description**
Correctness	:	Extent to which the software conforms to its specifications and conforms to its declared objectives
Maintainability	:	Ease of effort for locating and fixing a software failure within a specified time period
Verifiability	:	Ease of effort to verify software features and performance based on its stated objectives
Quality of Performance		
Efficiency	:	Extent to which the software is able to do more with less system (hardware, operating system, communications, etc.) resources
Integrity	:	Extent to which the software is able to withstand intrusion by unauthorized users or software within a specified time period
Reliability	:	Extent to which the software will perform (according to its stated objectives) within a specified time period
Usability	:	Relative ease of learning and the operation of the software
Testability	:	Ease of testing to program to verify that it performs a specified function
Quality of Adaptation		
Expandability	:	Relative effort required to expand soft ware capabilities and/or performance by enhancing current functions or by adding new functionality
Flexibility	:	Ease of effort for changing the soft ware's mission, functions or data to meet changing needs and requirements
Portability	:	Ease of effort to transport software to another environment and/or platform.
Reusability	:	Ease of effort to use the software (or its components) in another software systems and applications
Interoperability	:	Relative effort needed to couple the software on one platform to another software and/or another platform
Intra-operability	:	Effort required for communications between components in the same software system

TABLE 1

1.2 Software Quality Assurance (SQA)

Software quality assurance (SQA) is a systematic, planned set of actions necessary to provide adequate confidence that the software development process or the maintenance process of a software system product conforms to established functional technical requirements as well as with the managerial requirements of keeping the schedule and operating within the budgetary confines. The main objective of quality assurance is to minimize the cost of guaranteeing quality by a variety of activities performed throughout the development and manufacturing processes/stages. These activities prevent the causes of errors, and detect and correct them early in the development process. As a result, quality assurance activities substantially reduce the rate of products that do not qualify for shipment and, at the same time, reduce the costs of guaranteeing quality in most cases [4]. SQA governs the procedure to build the preferred quality into the products. Quality assurance (QA) techniques can be categorized into two ways, one is static and another is dynamic**.** Nothing like dynamic techniques, static techniques do not engage the execution of code. Static techniques engage examination of documentation by individuals or groups. This inspection may is assisted by software tools, e.g., examination of the requirements specification and technical reviews of the code. Testing and simulation are dynamic techniques. Sometimes static techniques are used to support dynamic techniques and vice versa. Agile methods typically use dynamic techniques.

III. Overview of Agile Methods

Agile promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change. It is a conceptual framework that promotes

foreseen interactions throughout the development cycle. Agile methods were developed to develop the system additional rapidly with restricted time spent on analysis and design [5]. Agile methods are iterative, focus on teamwork, collaboration between consumer and developer, feedback from consumer during the lifecycle of the software project [5]. Traditional model such as waterfall model is software model where any stage should not start until the last stage is complete while agile methods based on short release and go through all the stages at a time [5]. In fig.1 shows a simple agile methods life cycle that has a number of steps. The steps are: 1) User stories 2) release planning 3) Iteration planning 4) Unit testing 5) Code developing 6) Continuous integration 7) Acceptance testing 8) Small release and 9) System in use. These steps enable agile methods to deliver product releases in a much short period of time compared to the traditional approach. We explain generalized agile method development life cycle in details in section 3 that gives some techniques to ensure product quality.

IV. Agile Techniques to Enhance Software Quality

McBreen[6] defines agile quality assurance as the development of software that can respond to change, as the customer requires it to change[6].Beck[7] defined agile software quality in terms of efficiency, timeliness, cost effectiveness, ease of use, maintainability, integrity, robustness, extendibility, and reusability. But in this study we have considered 14 quality factors shown in Table1 to improve quality using agile techniques. In Fig. 2 we present agile techniques development life cycle in diagrammatic form. In Fig. 2 the left side shows the main sequence of agile process and right side includes some agile techniques that enhance QA ability. These techniques are marked by an underline. In an agile methods phase the development practices and quality assurance practices cooperate with each other strongly and substitute the result quickly in order to keep up the speed of the process. The agile techniques are discussed in details in the following sub section.

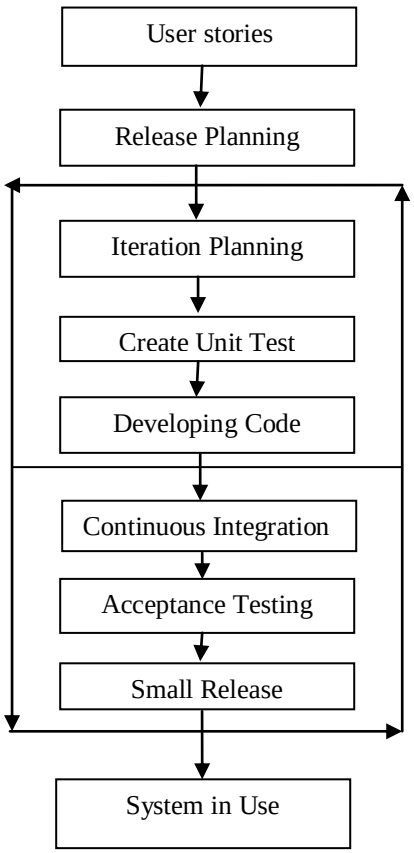

Figure 1: simple agile methods life cycle.

4.1 System Metaphor

Metaphor is a way to express meaning in a condensed manner by referring to qualities of known entities. Metaphors are useful because they are efficient. The system metaphor is a story that everyone: customers, programmers, and managers, can tell about how the system works. We seek a system metaphor for several reasons: common vision, shared vocabulary, generativity, and architecture [8]. It presents a simple shared story of how the system works, this story typically involves a handful of classes and patterns that shape

the core flow of the system being built. The idea of using a System metaphor to facilitate communication works toward revealing the reality of the team towards its task. For a team starting out, metaphors are a comfortable and flexible starting point and they leave open the chance to use the metonymy that patterns provide [9]. System metaphor does not seem to address bigger architectural issues such as if the system should even be implemented using objects, the hardware component of architecture, the process and inter-process communication component of architecture, or the separation of the system into layers and/or components [9].System metaphor is helpful for communication between customer and developer. It helps the agile development team in architectural evaluation by increasing communication between team members and users. So enhance maintainability, efficiency, reliability and flexibility.

4.2 Architectural Spike

An architectural spike is technical risk reduction techniques popularized by Extreme Programming (XP) where write just enough code to explore the use of a technology or technique that you're unfamiliar with. For complicated spikes architecture owners will often pair with someone else to do the spike [10]. It intends to identify areas of maximum risk, to get started with estimating them correctly. Agile projects are designed for iteration at a time. It is a thin slice of the entire application built for the purpose of determining and testing a potential architecture.

4.3 Onsite Customer Feedbacks

Onsite customer is one of the most practices in most agile projects that help the developers refine and correct requirements throughout the project by communicating. Customer are only involved during requirement collecting in traditional software developments but they are directly involved in agile methodology. All phases of agile project require communication with the customer, preferably face to face, on site. It's best to simply assign one or more customers to the development team. A real customer must sit with the team, available to answer questions, resolve disputes, and set small-scale priorities. Agile is intended to improve software quality and responsiveness to changing customer requirements. As a type of agile software development it advocates frequent releases in short development cycles, which is intended to improve productivity and introduce checkpoints where new customer requirements can be adopted.

4.4 Refactoring

Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior. Its heart is a series of small behavior preserving transformations. Each transformation (called a 'refactoring') does little, but a sequence of transformations can produce a significant restructuring. Since each refactoring is small, it's less likely to go wrong. The system is also kept fully operational after each small refactoring. Practically refactoring means making code clearer and cleaner and simpler and well-designed. It can reduce the chances that a system can get seriously broken during the restructuring [11, 12]. So refactoring reduces the probability of generating errors for the period of developments, hence improve software quality factors such as efficiency, reliability, intra-operability and interoperability, testability.

4.5 Pair Programming

Pair programming is a technique in which two programmers or engineers work together at one workstation. One writes code while the other, the observer, reviews each line of code as it is typed in. The best way to pair program is to just sit side by side in front of the monitor. Slide the key board and mouse back and forth. Both programmers concentrate on the code being written and continuously collaborating on the same design, algorithm, code or test [13]. The two programmers switch roles frequently. While reviewing, the observer also considers the strategic direction of the work, coming up with ideas for improvements and likely future problemsto address. This procedure increases software quality without impacting time to deliver. Pair programming can improve design quality factors such as correctness, verifiability, and testability and reduce defects [14].

4.6 Stand-up-Meeting

Stand-up-meeting is the most important practice in agile methods. It increases the communication between term members and developers. A stand-up meeting (or simply "stand-up") is a daily team meeting held to provide a status update to the team members. Communication among the entire team is the purpose of the stand-up- meeting. The meeting is usually held at the same time and place every working day. All team

members are encouraged to attend, but the meetings are not postponed if some of the team members are not present.

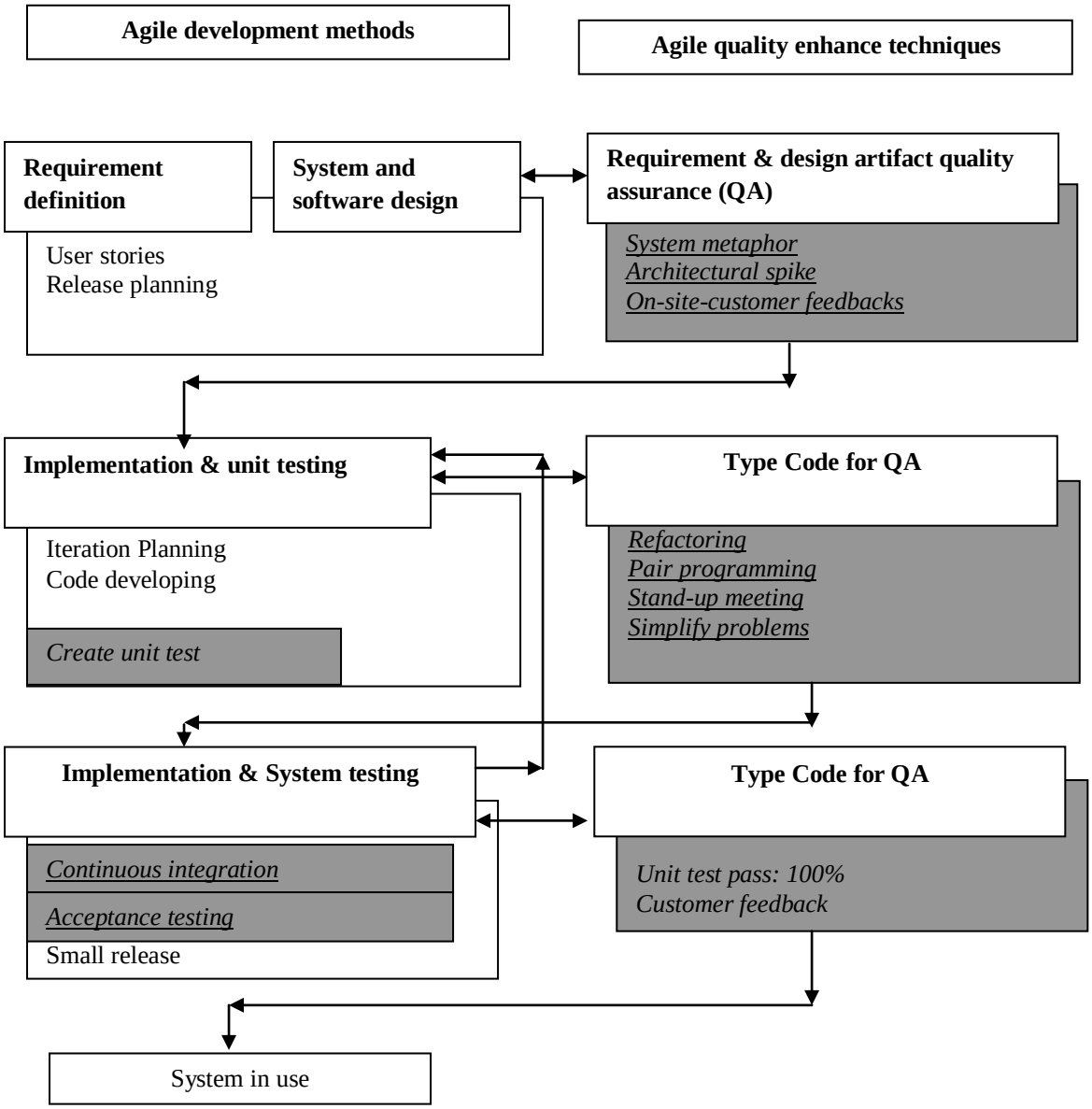

Figure 2: agile techniques and quality assurance

This meeting is used to communicate problems, solutions, and promote team focus. Everyone stands up in a circle to avoid long discussions. It is more efficient to have one short meeting that everyone is required to attend than many meetings with a few developers each. During a stand up meeting developers report at least three things; what was accomplished yesterday, what will be attempted today, and what problems are causing delays. Stand-up-meeting improve software quality factors such as reliability and flexibility.

4.7 Continuous Integration (CI)

Continuous Integration (CI) is a fashionable practice among agile methods where members of a team integrate their work frequently; usually each person integrates at least daily - leading to multiple integrations per day. It was first named and proposed as part of extreme programming (XP). Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software

more rapidly [15]. Its main aim is to prevent integration problems, referred to as "integration hell" in early descriptions of XP. This continuous application of quality control aims to improve the quality of software such as integrity, usability, testability and to reduce the time taken to deliver it, by replacing the traditional practice of applying quality control after completing all development. This is very similar to the original idea of integrating more frequently to make integration easier, only applied to QA processes.

Software Quality Evaluation with Agile Techniques

Quality Factors		Agile techniques
Correctness	:	Architectural Spike Onsite customer feedback
Maintainability	:	Continuous Integration
Verifiability	:	Continuous Integration
Efficiency	:	Pair programming System metaphor
Integrity	:	Continuous Integration
Reliability	:	Refactoring System metaphor
Usability	:	Continuous Integration
Testability	:	Pair programming Acceptance testing Unit testing Refactoring
Expandability	:	Continuous Integration Onsite customer feedback
Flexibility	:	Stand-up meeting System metaphor
Portability	:	Pair programming Acceptance testing
Reusability	:	Refactoring Continuous Integration
Interoperability	:	Refactoring System metaphor
Intra-operability	:	Continuous Integration

TABLE 2

4.8 Acceptance Testing

Acceptance testing is a term used in agile methodologies, particularly Extreme Programming, referring to the functional testing of a user story by the software development team during the implementation phase. The customer specifies scenarios to test when a user story has been correctly implemented. A story can have one or many acceptance tests, whatever it takes to ensure the functionality works. Acceptance tests are black-box system tests. Each acceptance test represents some expected result from the system. Customers are responsible for verifying the correctness of the acceptance tests and reviewing test scores to decide which failed tests are of highest priority. Acceptance tests are also used as regression tests prior to a production release. A user story is not considered complete until it has passed its acceptance tests. This means that new acceptance tests must be created for each iteration or the development team will report zero progress. Quality assurance (QA) is an essential part of the XP process as well as agile method. On some projects QA is done by a separate group, while on others QA will be an integrated into the development team. The acceptance phase may also act as the final quality gateway, where any quality defects not previously detected may be uncovered. A principal purpose of acceptance testing is that, once completed successfully, and provided certain additional (contractually agreed) acceptance criteria are met. Acceptance testing occurs much earlier and more frequently in an agile methods with respect to traditional approach.

V. Evaluation of Software Quality Factors (SQF)

Agile process addresses quality issues repeatedly and continuously, and focusing on the role of quality. SQA is not to find defects but to avoid defects. Agile method seeks Software quality through increased customer value, fewer defects, faster development and flexibility to changing environment. Agile methods do have practices that have QA abilities, some of them are inside the development phase and some others can be separated out as supporting practices. Many of the agile quality activities occur much earlier than they do in traditional process. Most of these activities will be included in each iteration and the iterations are frequently repeated during development. Table 2 lists the identified quality factors as defined in Table 1 and summarizes after evaluating software quality factors in the particular agile technique.

VI. Conclusion

Most agile techniques attempt to minimize risk by developing software in short time boxes called iterations. While iteration may not add enough functionality to warrant releasing the product, an agile software project intends to be capable of releasing new software at the end of iteration. An agile technique aims to develop and implement software quickly in close cooperation with the customer in an adaptive way and enhance software quality. When new changes are needed to be implemented then agile techniques is used. The freedom agile gives to change is very important. New changes can be implemented at very little cost because of the frequency of new increments that are produced. The main advantages of agile techniques are able to respond to the changing requirements, customer satisfaction by rapid, continuous delivery of useful software. Disadvantage of agile technique is in case of some software deliverables, especially the large ones; it is difficult to assess the effort required at the beginning of the software development life cycle. In conclusion we can say that agile technique try to find software quality as well as enhance software quality all the way through increased customer value, defects and flexibility to changing environment and finally we also say that quality attribute or factors depends on upon customer contentment in terms of time, budget, usability, serviceable requirement and security.

Reference

[1] Ming Huo, June Verner, Liming Zhu, Muhammad Ali Babar, Software Quality and Agile Methods, Proceedings of the 28th Annual *International Computer Software and Applications Conference (COMPSAC'04)*

[2] What Is Software Quality? , http://www.ocoudert.com/blog/2011/04/09/what-is-software-quality

[3] G. Gordon Schulmeyer, *The handbook of software quality Assurance*, Prentice Hall, 1998.

[4] Daniel Galin, Software Quality Assurance, First published 2004.

[5] Osama Suhaib and Khalid khan, The Role of Software Quality in Agile Software Development Methodologies,Technology Forces (Technol. forces): *Journal of Engineering and Sciences January-June 2010.*

[6] Pete McBreen. Mcbreen, *Quality Assurance and Testing in Agile Projects, Consulting 2003*.

[7] K. Beck: Extreme programming explained: Embrace change, Second edition Kent Beck Publisher.2000. http://www.mcbreen.ab.ca/talks/CAMUG.pdf (last accessed January 2013)

[8] Explore Extreme Programming: The system Metaphor. http://xp123.com/articles/the-system-metaphor/ (Last accessed 26th January 2013)

[9] System metaphor http://c2.com/xp/SystemMetaphor.html (Last accessed 26th January 2013)

[10] The Architecture Owner Role: How Architects fit in on Agile Teams, Scotttw.amble http://www.agilemodeling.com/essays/architectureOwner.htm

[11] Martin Fowler, '*Information regarding Refactoring* ', http://refactoring.com (Last accessed 27th January 2013)

[12] Martin Fowler, *what is refactoring?* http://refactoring.com (Last accessed 27th January 2013)

[13] Extreme programming: Pair Programming. http://www.extremeprogramming.org/rules/pair.html

[14] A. Cockburn and L. Williams, "*The Costs and Benefits of Pair Programming," in Extreme Programming examined*, G. Succi and M. Marchesi, Eds. Boston: Addison-Wesley, 2001, pp. xv, 569 p.

[15] Martin Fowler, *Continuous Integration*. http://martinfowler.com/articles/continuousIntegration.html (Last accessed 27th January 2013)