Defining Agile Software Quality Assurance

E. Mnkandla, *Member, IEEE,* B. Dwolatzky, *Member, IEEE*

***Abstract*— Agile software development methodologies have since their inception claimed to improve the quality of the software product. The agile practitioners have also claimed that use of the agile approach has greatly improved the quality of their products. However, software quality is a rather complex concept; in fact some have defined the entire discipline of software engineering as the production of quality software. Quality according to ISO 9000 is defined as "the totality of characteristics of an entity that bear on its ability to satisfy stated or implied needs." In the existing agile literature there has not been a comprehensive definition of which characteristics of software quality are improved by the use of agile processes in developing software. In this paper an innovative technique is introduced for evaluating agile methodologies in order to determine which factors of software quality they improve. The technique uses a set of adapted software quality factors as defined by Bertrand Meyer and McCall.**

***Index Terms*—Agile methodologies, agile quality assurance, software development, software quality.**

I. Introduction

Found in a number of agile methodologies literature is the general notion that agile development is oriented towards the development of higher quality software compared to plan-driven development, see [12] for a detailed comparison. However, a closer look into such claims reveals that there is lack of a comprehensive technique to evaluate how agile processes meet software quality requirements. There are very few articles on agile methodologies that specifically tackle issues of software quality in such a way as to reveal the specific characteristics of quality that agile processes have improved. What literature reveals however is that agile developers implement quality issues differently from the way traditional quality professionals do it. It is also interesting to realise that agile methodologies were not formed out of a deliberate desire to add another group of methodologies that would solve a, b, c, d… problems in the software development process, but rather they were formed out of a discovery of the commonality of practices that developers found themselves following as required by their processes in their different organizations. The task of classifying quality attributes may be quite complex and subjective hence it is not the aim of this paper to develop an exhaustive analysis of the numerous taxonomies of quality. However, due to the conceptual differences between agile methodologies and plan-driven methodologies the existing techniques that have mainly been used for evaluating plan-driven methodologies may not be suitable for agile processes. There is therefore a need to develop techniques that are inline with the highly revolutionary agile development concepts.

Ambler's article [1] starts by stating an assumption that software quality is met through test-driven development in agile methods. Since there is a strong emphasis on developing test cases before writing code in agile development it appears logical to agree with Ambler that "quality is an inherent aspect of agile development" [1]. However, the definition of software quality includes a few other concepts, though correctness comes out as the most important parameter in which testing is a technique for ensuring correctness. A radical approach would be to redefine quality from the agile perspective since agility has redefined a number of concepts in software engineering.

This paper introduces a technique for evaluating agile methodologies with regards to the way they meet software quality characteristics as defined in Meyer [2]. Meyer derives these quality characteristics from McCall's quality taxonomy model. The technique proposed here basically picks an agile method and breaks it down into process activities. Then for each process activity of the agile method, an evaluation of what software quality factors are met is done. This action is repeated until all the quality factors are covered. The rest of the paper starts by presenting a review of previous work on agility and quality followed a description of the proposed evaluation technique. A walk through two of the agile methodologies in common use today is done and a discussion of the results follows. The paper ends by providing conclusions and possible future work.

II. Agility and Quality

This section starts by summarizing the traditional definitions of quality and then presents a summary of the work that has been done in the area of agility versus quality.

A. What is Quality?

Joseph Juran is considered by many quality assurance professionals to be a quality legend. It should be interesting to think of how he would have evaluated agile processes against quality assurance issues. Consider for example what he said about the ISO 9000 when he was asked by Quality Digest if he thought ISO 9000 had actually hindered the quality

•Ernest Mnkandla is with the School of Information Technology, Monash University, Johannesburg, 1725, South Africa, phone. +27 11 950 4038, fax: +27 11 950 4033. Email: Ernest.Mnkandla@infotech.monash.edu.au

•Barry Dwolatzky is with the School of Electrical and Information Engineering, University of the Witwatersrand, Wits, 2050 Johannesburg, South Africa. E-mail: b.dwolatzky@ee.wits.ac.za.

Proceedings of the International Conference
on Software Engineering Advances (ICSEA'06)
0-7695-2703-5/06 $20.00 © 2006 IEEE

movement; "Of course it has. Instead of going after improvement at a revolutionary rate, people were stampeded into going after ISO 9000, and they locked themselves into a mediocre standard. A lot of damage was, and is, being done" [3].

The International Standards Organization ISO 9000 defines quality as the totality of characteristics of an entity that bear on its ability to satisfy stated or implied needs. Where 'stated needs' means those needs that are specified as requirements by the customer in a contract, and 'implied needs' are those needs that are identified and defined by the company providing the product.

Weinberg [17] defines quality simply as: "the value to some people". This definition has been expanded upon to mean the association of quality with human assessment, and cost and benefit.

Juran [4] defines quality as fitness for use which means the following two things; "1) quality consists of those product features that meet the needs of the customers and thereby provide product satisfaction. 2) Quality consists of freedom from deficiencies."

Philip Crosby [5] developed and taught concepts of quality management. The influence of Crosby's work can be found in the ISO 9000:2000 standard. This standard differs from the 1994 standard in the context in which each of the eight principles defines quality, i.e. as conformance to requirements and zero defects.

These definitions of quality have a contextual bias towards the manufacturing industry. A more general application of these definitions to IT and other areas is possible, however when applying quality assurance to software products the context is slightly different because software products are physically visible.

Some software engineers have defined software quality as follows:

Meyer [2] defines software quality according to an adapted number of quality factors as defined by McCall [10], which are; correctness, robustness, extendibility, reusability, compatibility, efficiency, portability, integrity, verifiability, and ease of use etc.

Pressman agrees with Crosby and defines quality as "conformance to explicitly stated functional requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software" [6].

Sommerville [7] defines software quality as a management process concerned with ensuring that software has a low number of defects and that it reaches the required standards of maintainability, reliability, portability and so on.

van Vliet [8] follows the IEEE definition [16] of quality as stated in the IEEE Glossary of Software Engineering Terminology. van Vliet's perspective then combines this definition with the analysis of the different taxonomies on quality.

Pfleeger [9] agrees with Garvin who views quality from five different perspectives namely:

- The transcendental meaning that quality can be recognized but not defined,
- User view meaning that quality is fitness for purpose,
- Manufacturing meaning that quality is conformance to specification,
- Product view meaning that quality is tied to inherent product characteristics and
- The value-based view meaning that quality depends on the amount the customer is willing to pay for the product.

Bass [15] argues that the common practice of defining software quality by dividing it into the ISO 9126 (i.e. functionality, reliability usability, efficiency maintainability, and portability) does not work. His argument that, "in order to use a taxonomy, a specific requirement must be put into a category" [15]. However, there are some requirements that may be difficulty to put under any category, for example, 'denial of service attack, response time for user request, etc'. What Bass [15] then proposes is the use of quality attribute general scenarios.

From an agile perspective quality has been defined by some practitioners as follows:

McBreen [11] defines agile quality assurance as the development of software that can respond to change as the customer requires it to change. This implies that the frequent delivery of tested, working, and customer-approved software at the end of each iteration is an important aspect of agile quality assurance.

Ambler [1] considers agile quality to be a result of practices such as effective collaborative work, incremental development, and iterative development as implemented through techniques such as refactoring, test-driven development, modelling, and effective communication techniques.

In Table 1 is given a summary of the factors that define agile quality. These aspects of agile quality have eliminated the need for heavy documentation that is prescribed in plan-driven processes as a requirement for quality. Quality is a rather abstract concept that is difficult to define but where it exists it can be recognized. In view of Garvin's quality perspective there may be some who have used agile methodologies in their software development practices and seen improvement in quality of the software product but could still find it difficult to define quality in the agile world.

III. A Tool for Evaluating Quality in Agile Processes

This section presents a full description of a tool for the evaluation of quality assurance in agile processes. The contributions of this tool to agile technologies are; 1) the provision of detailed knowledge about specific quality issues of the agile processes, 2) identification of innovative ways to improve agile quality, 3) identification of specific agile quality techniques for particular agile methodologies. It is envisaged that when such knowledge is acquired it will open up innovation horizons for the improvement of quality related benefits in different agile processes.

Before describing the evaluation tool it is worth mentioning

that Huo et al [12] developed a comparison technique whose aim was to provide a comparative analysis between quality in the waterfall development model (as a representative of the traditional methodologies approaches) and quality in the agile group of methodologies. The results of the analysis showed that there is indeed quality assurance in agile development, but it is achieved in a different way to the traditional processes. The limitations of Huo et al's tool however, are that the analysis:

- Singles out two main aspects of quality management namely; quality assurance and verification and validation.
- Overlooks other vital techniques used in agile processes to achieve higher quality management.
- And is mainly based on Extreme Programming which on it own is not a complete representation of agile methodologies.

Another challenge of Huo et al's technique is that whilst the main purpose of that analysis was to show that there is quality assurance in agile processes it does not make clear what the way forward could be. Agile proponents do not seem to be worried about comparison between agile and plan-driven as some of the more zealous "agilists" believe that there is no way traditional waterfall based methods can match agile methods in any situation [Tom Poppendieck, personal email 2005].

The tool developed in this paper picks from where Huo et al [12] left by further identifying some more agile quality techniques and then in an innovative way identifies the agile process activities that correspond to each technique. The new knowledge contributed by this tool is the further identification of possible activities that can be done to improve on the already high quality achievements enjoyed by agile processes.

A. Software Quality Assurance Factors

The factors that define software quality from a top-level view can be rather abstract. However the proposed tool picks each of the factors and identifies the corresponding agile techniques that implement the parameter in one way or another. Possible improvements to the current practice have been proposed by analysing the way agile practitioners work. Of great importance to this kind of analysis is a review of some of the intuitive practices that developers usually apply which may not be documented. You may wonder how much objectivity can be in such information. The point though is that developers tell their success stories at different professional forums and some of the hints from such deliberations have been captured in this tool without following any formal data gathering methodology. The authors believe that gathering of informal raw data balances the facts especially in cases were developers talk about their practice. Once the data is gathered formally then a lot of prejudices and biases come in and there will be need to apply other research techniques to balance the facts. Table 1 gives a description of software quality assurance factors and Table 2 summarize the application of the evaluation tool to particular agile methodologies.

In the field of software quality management quality assurance activities are fulfilled by ensuring that each of the factors defined in Table 1 are satisfied to a certain extent in the software development lifecycle of the process concerned. A brief definition of each of these factors is given in Table 1according to Meyer [2, 19].

Table1. A Description of Software Quality Assurance Factors

In order to give a better understanding of the information in Table1 readers may find a further classification of software quality factors into external and internal useful. This classification is based on the people involved in the development process i.e. developers and customers. Meyer [19] defines external factors as those factors that are: "visible to customers, for example, ease of use extendibility, timeliness". Internal factors on the other hand are those factors that are "perceptible only to the developers, for example, good programming style, information hiding"

Meyer [19] also gives a more technical classification groups software quality factors into product and process based factors. Product based factors are those factors that define the "properties of the resulting software, for example correctness, efficiency" [19]. Process based factors are those factors that reveal the "properties of the procedures used to produce and "maintain" the software, for example, timeliness, and cost-effectiveness" [19]. The derived product software quality factors have not been included here but they are clearly defined in [2].

So fundamental to agile development is the fact that whatever brilliant plans and wonderful documents may decorate the software project if code has not been delivered the effort is worthless. A corollary to this is that no matter how efficient or timely a system is if it does not meet the customer specification it is not worth the effort.

B. Agile Development and Quality Factors

Each of the factors defined in Table 1 will now be evaluated in relation to the corresponding agile techniques that implement the factors. For each parameter we have listed some practices that we believe will improve on the way agile development implements software quality assurance.

Compatibility

Agile techniques that ensure correctness of a system include the following: A general feature of all Object-Oriented (OO) developed applications.

Possible improvement on the agile approach includes design and architectural considerations that aim for platform independence.

Cost-effectiveness

Agile techniques that ensure cost-effectiveness of a system include the following: controlling the scope creep, for example in the Scrum methodology the Sprint technique is used which prevents introduction of requirement changes until the end of the iteration (Sprint).

Possible improvements include avoiding scope creep without locking requirement changes. It is generally difficulty to convince a customer to sign a contract for a project whose cost is based on the cost of each iteration. The advantage of

costing based on iterations however is that since iterations are short (one to four weeks) the customer gets frequent feedback on the project costs.

Correctness

Agile techniques that ensure correctness of a system include the following as elicited from the generic principles that guide agile development: writing code from minimal requirements, specification, which is obtained by direct communication with the customer, allowing the customer to change requirements, user stories, and test-first development. Since all the development in agile processes is done iteratively these techniques ensure the correctness at iteration level before making the decision to continue or cancel the project.

These agile techniques can be improved by implementing the following: Consider the possibility of using formal specification in agile development (which some developers are already using), possible use of general scenarios to define requirements [15].

Ease of use

Agile techniques that ensure ease of use of a system include the following: since the customer is part of the team, and customers give feedback frequently, they will likely recommend a system that is easy to use. The frequent visual feedback that customers get during the delivery of an iteration allows them to provide useful feedback to improve the usability of the system.

These can be improved upon by designing for the least qualified user in the organization.

Efficiency

Agile techniques that ensure efficiency of a system include the following: application of good coding standards.

To improve on the techniques designs based on the most efficient algorithms are encouraged.

Extendibility

Extendibility of a system is a general feature of all OO developed applications; however emphasis should be on technical excellence and good design.

To improve on these techniques use of modeling techniques for software architecture should in agile development.

Integrity

Integrity of a system is ensured at operating system level and also at the development platform level.

Improving the integrity of the techniques that define the product would improve system integrity.

Maintainability

The application of Object-Oriented design principles leads to maintainable systems.

Development technologies that improve the interfaces between different object modules can have a positive impact on maintainability.

Portability

Originally defined as a major part of Object-Oriented design and now further enhanced by the concepts of distributed computing and web services, this quality factor is generally implemented through the concepts of Object-Oriented design.

Reusability

This quality factor is generally implemented through the concepts of Object-Oriented technology.

More work on agility and software architecture, and patterns can improve the reusability of agile products.

Robustness

Agile techniques that ensure robustness of a system originally defined as a major part of Object-Oriented design which agile development follows.

This is case dependent; however agile development ensures robustness in the general sense through the development standards that are inherent to particular development platform in use.

Timeliness

Agile techniques that ensure timeliness of a system include the following: iterative development, quick delivery, and short cycles.

This can be improved upon by reducing the time for the deployment process.

Verifiability and Validation

Agile techniques that ensure verification and validation of a system include the following: test-driven-development, unit tests and frequent integration.

To improve on these techniques more tools could be developed to link the existing testing approaches the concepts of test-driven-development.

C. Validation of the Tool using Extreme Programming and Lean Development

This section applies the evaluation tool to Extreme Programming (XP) and Lean Development (LD) as illustrated in Table 2. These two agile methodologies were chosen to validate the use of the tool.

The evaluation process requires the analyst to have expert knowledge in the agile methodologies to be evaluated. The process starts by selecting a quality assurance parameter and analyzing the meaning of the parameter. For example correctness means "The ability of a system to perform according to defined specification". The analysis should then lead to the identification of features of the development process that ensure performance of the intended system to suit the defined specification. For example when using XP; user stories would ensure that the requirements are represented in a simple language that can be easily understood by the customer. When user stories are combined with the technique of unit testing then each implementation of the user stories is tested as the system is developed ensuring correctness. This approach is followed for each software quality assurance parameter. Table 2 lists the identified techniques for XP and LD. The list is not meant to be exhaustive as some techniques may be case dependent. The tool may also be extended to other methodologies agile or not.

Table 2. Quality Activities in Particular Agile Methodologies

-*Case dependent*: means that the methodology may implement the parameter in particular cases.

IV. CONCLUSION

The paper summarized a thinking tool for the evaluation of

quality management in agile methodologies. The major contributions of the proposed technology are: the further identification of possible activities that can be done to improve on the already high quality achievements realised in agile methodologies, the provision of detailed knowledge about specific quality issues of the agile processes, and the identification of specific agile quality techniques for particular agile methodologies. It is envisaged that when such knowledge is acquired it will open up innovation horizons for the improvement of quality related benefits in different agile processes.One of the questions that may be considered is "could there be limits on the possible improvements on software quality?" This question can be considered in light of the fact that issues of software quality assurance may differ according to situations.

Agility introduces a paradigm shift in project management in the sense that every part of the software development process is reviewed with the aim of reducing the activities and number of deliverables to the minimum needed in any given situation. Such an approach appears to take control away from a traditional project manager. The move is in fact from a command oriented management structure to a facilitator oriented management system. As seen from the way software quality factors are defined in agile processes, the central players in the development process are the customer and developer and not the manager. We hope this paper will spark discussions and research in the field of quality and agile development, especially issues of modeling agile software quality and architecture as agility ventures into areas such as enterprise architecture, and off shore development [21].

REFERENCES

[1] Ambler, S., (2005), Quality in an Agile World, Software Quality Professional, Vol. 7, No. 4, pp. 34-40.

[2] Meyer, B., (2000), Object-Oriented Software Construction, Prentice Hall PTR, pp.4-20.

[3] QCI International, (2002), Juran: A life of Quality: an exclusive interview with a quality legend. Quality Digest Magazine, [online]. Available from: http://www.qualitydigest.com/aug02/articles/01_article.shtml [Accessed 26 July 2006].

[4] Juran, J. M., and Gryna, F. M., (1988), Juran's Quality Control Handbook, Mcgraw-Hill.

[5] Crosby. P. B., (1984), Quality Without Tears, McGraw-Hill.

[6] Pressman, R.S., (2001), Software Engineering a Practitioner's Approach, Mcgraw-Hill.

[7] Sommerville, I. (2004), Software Engineering. Addison-Wesley.

[8] van Vliet, H. (2003), Software Engineering: Principles and Practice, John Wiley and Sons.

[9] Pfleeger, S. L. (2005), Software Engineering: Theory and Practice, Prentice Hall.

[10] McCall, J. A., Richards, P. K., and Walters, G. F., (1977), Factors in Software Quality, Vols 1, 2, and 3, National Technical Information Service.

[11] McBreen, P., Quality Assurance and Testing in Agile Projects, McBreen Consulting, [online]. Available from: http://www.mcbreen.ab.ca/talks/CAMUG.pdf [Accessed 26 July 2006].

[12] Huo, M., Verner, J., Zhu, L., and Babar, M. A., (2004), Softwrae Quality and Agile Methods, Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC04), IEEE Computer.

[13] Turk, D., France, R., and Rumpe, B., (2002), Limitations of Agile Software Processes, Proceedings of the Third International Conference on eXtreme Programming and Agile Processes in Software Engineering, p. 43-46.

[14] Weisert, C., (2002), The #1 Serious Flaw in Extreme Programming (XP), Information Disciplines, Inc., Chicago. [Online]. Available from: http://www.idinews.com/Xtreme1.html [Accessed 26 July 2006].

[15] Bass, L. (2006), Designing Software Architecture to Achieve Quality Attribute Requirements. Proceedings of the Third IFIP Summer School on Software Technology and Engineering, January 2006 South Africa, pp. 1-29.

[16] IEEE. (1990), IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12.

[17] Weinberg, G.M., (1991), Quality Software Management, v. 1 Systems Thinking, Dorset House.

[18] Mnkandla, E., and Dwolatzky, B., (2006), "Agile Software Methods: State-Of-The-Art" accepted for publication in, Stamelos, I. and Sfetsos, P. (Eds.) Agile Software Development Quality Assurance, Idea Group Publishing.

[19] Meyer, B., (2006), Eiffel and Design by Contract, Proceedings of the Third IFIP Summer School on Software Technology and Engineering, January 2006 South Africa.

[20] Todd, P., and Lockheed, M., (2000), Data Model Project: Quality Assurance Plan. [Online]. Available from: http://codecourse.sourceforge.net/materials/Quality-Assurance-Plan.html [Accessed 26 July 2006].

[21] McGovern, J., Ambler, S.W., Stevens, M.E., Linn, J, Jo, E.K., and Sharan, V., (2003), The Practical Guide to Enterprise Architecture, Prentice Hall PTR.

TABLE I. A DESCRIPTION OF SOFTWARE QUALITY ASSURANCE FACTORS

Parameter	Description
Correctness	The ability of a system to perform according to defined specification.
Robustness	Appropriate performance of a system under cases not covered by the specification. This is complementary to correctness.
Extendibility	A system that is easy to adapt to new specification.
Reusability	Software that is composed of elements that can be used to construct different applications.
Compatibility	Software that is composed of elements that can easily combine with other elements.
Efficiency	The ability of a system to place as few demands as possible to hardware resources, such as memory, bandwidth used in communication and processor time.
Portability	The ease of installing the software product on different hardware and software platforms.
Timeliness	Releasing the software before or exactly when it is needed by the users.
Integrity	How well the software protects its programs and data against unauthorized access.
Verifiability and Validation	How easy it is to test the system.
Ease of use	The ease with which people of various backgrounds can learn and use the software.
Maintainability	The ease of changing the software to correct defects or meet new requirements [20].
Cost-effectiveness	The ability of a system to be completed within a given budget.

TABLE II. QUALITY ACTIVITIES IN XP AND LD [18]

Software Quality Parameters	**XP Quality Activities**	**LD Quality Activities**
Correctness	User stories, Unit tests, Customer feedback	Meet customer requirements
Robustness	Generic OO design practices	Generic OO design practices
Extendibility	Simple design	Continuous improvement.
Reusability	Generic OO design practices	Case dependent
Compatibility	Inherent in OO systems	This could conflict with eliminating waste.
Efficiency	The metaphor - encourages simplicity, Coding standard, Pair programming.	Minimize inventory.
Portability	Generic OO design practices	Generic OO design practices
Timeliness	Iterative incremental development (IID).	IID, Maximize flow, Do it right the first time
Integrity	Generic OO design practices	Case dependent
Verifiability and Validation	Unit testing, Continuous integration.	Do it right the first time.
Ease of use	Simple design, On-site customer.	Partner with suppliers.
Maintainability	Iterative development	Iterative development
Cost-effectiveness	Iterative development, quick delivery	Iterative development, quick delivery