International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 10, ISSN No. 2455-2143, Pages 139-145
Published Online August - September 2016 in IJEAST (http://www.ijeast.com)

METRICS FOR SOFTWARE MAINTAINABILITY AND USABILITY IN AGILE ENVIRONMENT

Priyanka
Department of Computer Science & Engineering
BRCM CET, Bahal, MDU, India

Praveen Kantha
Department of Computer Science & Engineering
BRCM CET, Bahal, MDU, India

ABSTRACT: It is not easy to comprehend the quality and features of software unless we are familiar of its software development process and software products. Some measurements process should be there to predict the development of the software, and to evaluate the software products. In the conventional technique for the product advancement, there are a number of measurements to compute the maintenance and utilization of programming. This investigation is to understand whether the same measures apply to Agile, or there is a need to modify a few measurements utilized for the agile environment. This paper gives a brief view on Maintainability and Usability by which the specified quality factors of software can be predicted. Maintainability and Usability are emerging attributes of software quality, which play a very important role in determining the quality and excellence of a software system. Consequently, the usage of software metrics improves quality and excellence of software.

Keywords: Software Maintainability, Software usability, agile environment, Software metrics

I. INTRODUCTION

Agile methodology is a product strategy, which depends on iterative and incremental techniques for programming advancement.
Small groups work upon individual modules. As these modules are created, it will be sent to the customer for audit. This model is adaptable, which incorporates changes considering client needs. Improvement techniques are utilized for programming advancement as per the standards and practices.
Maintainability is the process of altering software after it has been delivered and in proper use is called software maintenance [1]. Maintenance can be referred to as the process that is carried out when software goes through modifications and changes to code and its related documentation and credentials due to fault or the requirement.

Maintenance consumes 40% to 80% of price of the software and is therefore probably the most important phase of software Life cycle. Manufacturing enhancements contributes to 60% of maintenance cost, which is something that makes the systems is going to provide additional value.
Maintainability deals with period of maintenance outages or how long it takes to complete (easiness and speed) the maintenance and preservation actions measured up to a datum. The datum includes maintenance is carried out by recruits having specified expertise levels, using approved procedures and resources, at each prescribed level of maintenance. There are 4 types of maintenance:

1. Corrective Maintenance: This refers to amendments initiated by defects in the software.
2. Adaptive Maintenance: It includes transforming the software to match alteration in the ever changing environment.
3. Perfective Maintenance: It means civilizing the processing efficiency or performance, or streamlining the software to improve changeability.
4. Preventive Maintenance: This may lead enhance the complexity of the software, which reflects deteriorating structure.

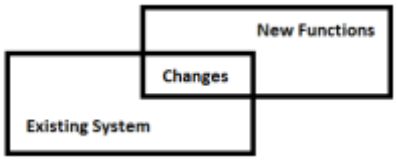

Figure 1: Software maintenance image representation

On the other hand usability relates how the system communicates with the user, and it includes the following five basic attributes: learnability, efficiency, user retention over time, error rate, and satisfaction [2].

- **Learnability:** How simple it is to increase proficiency to complete the job and gain knowledge of the foremost functionality of the system. It generally assess this by calculating the time a user pay out working with the system before that user needs to complete certain tasks in the time it

International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 10, ISSN No. 2455-2143, Pages 139-145
Published Online August - September 2016 in IJEAST (http://www.ijeast.com)

would take an expert to complete the same tasks. This attribute is having a lot of significance for trainee users.

- **Efficiency:** The quantity of tasks per unit of time that any user can carry out using the software system. The higher the usability of system is, more rapidly the user can complete the task.
- **User retention over time:** It is significant for irregular users to be capable of using the system without scrambling the learning curve again. This trait shows how finely the user memorizes how the system will work after a span of non-usage.
- **Error rate:** This attribute contributes negatively to usability. It does not meant to system errors. Oppositely, it points to the number of errors the user commits while doing a task. Good usability results in low error rate.
- **Satisfaction:** This shows a user's subjective impression of the system. According to ISO 9241, Part 11, usability is "the extent to which a product can be used by particular users to accomplish specified goals with effectiveness, efficiency, and satisfaction in a specified context of use." This definition bounds usability of a system to exact conditions, needs, and users.

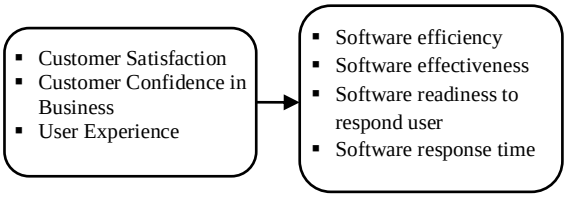

Figure 2: Software Usability representation

II. SOFTWARE METRICS

A mathematical measure of software that is susceptible to differences in the uniqueness of software can be termed as Software Metrics. These metrics measure an attribute which the body of software has [3]. Its main objective is to know the software development process by controlling the different aspects.

Software metrics provide an easy and inexpensive method to detect and also correct the possible causes for low product quality according to the quality factor as this will be perceived by the programmers.

Software metrics will be helpful only if they are characterized effectively and validated to that their worth is proven.

1. A metric should have advantageous mathematical properties.
2. A software metric should increases when positive traits occur or, decreases when undesirable traits are encountered, the value of the metric is supposed to vary in the same manner.
3. Before publishing or for making decisions validation of each metric should be done empirically in all the possible manners.

The information which is gained from software metric can be used to manage, administer and control the development process, which will show the way to improvement in the results of the software product. So, some of the ideal properties of a software metrics are:

- It must be simple and clear.
- It must be robust in nature.
- It must be reasonable.
- It must have an objective.
- It must be valid.

2.1 MAINTAINABILITY AND USABILITY METRICS

According to our research, if the below metrics is used in agile rather than the direct numbering game as in conventional environment, it will be more beneficial to track the progress of project, measuring value delivered to the customer and make sure about the on time delivery of the software to client.

Certain measures and their impact on the maintainability and usability of the software are described below:

TABLE I. Maintainability and Usability Metrics

Metric Name	Importance and their effect on the maintainability
Maintainability Index	Maintainability is used to calculate the state of maintenance. It calculates an index value between 0 and 100, which represents the relative ease of holding the codes. A high value indicates a better maintainability. Evaluations color code can be used to quickly identify trouble spots in your code. A green note is between 20 and 100 indicates that the code has good maintainability. A yellow note is 10 to 19 indicate that the code is moderately maintainable. A red mark is a value between 0 and 9 and indicates low maintainability. For thresholds, the decision is to break into the 20-80 range from 0-100, so noise levels became low, and only code reported that there were really suspicious held.
Complexity	Cyclomatic complexity measures the complexity of the code structure. It is created by calculating the number of different code paths in the program flow. A program that has complex flow control is required more tests, in order to ensure a good coverage and less maintainable code.
Code Hierarchy	It shows the number of class definitions that extend to the root of the class hierarchy. The deeper the hierarchy, the more it can be difficult to understand where methods and fields are defined and / or redefined.
Inter-module relations	It measures the connectivity between unique classes through parameters, local variables,

International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 10, ISSN No. 2455-2143, Pages 139-145
Published Online August - September 2016 in IJEAST (http://www.ijeast.com)

	return types, method calls, generic or model instances, base classes interface implementations, defined types of external decoration attribute. Software design requires that the types and methods should have high cohesion and low coupling. High coupling is a design that is difficult to maintain and to reuse because of its many dependencies on other types.
Size	There are the approximate numbers of rows in the code. The count depends on the IL code and is therefore not the exact number of lines in the source file. A high number may indicate a type or method tries to do too much work and should be shared. It may also mean that the type or method might be difficult to maintain. Knowing the likelihood that a user experiences a problem at any stage of development can be an important indicator to measure the impact of usability and ROI. To know what you experienced, users can rate the discovery of problems and what problems are found
The entire working process time	It can be used to measure the efficiency and productivity. Record the time to carry out for a user to perform a task in a few seconds or minutes. Departure times of tasks when users read work scenarios and ends at the time when the users have completed all actions (including the review period).
Job satisfaction level	When users attempt a task and asked about the difficulty of the task, he answered rarely few questions about the difficulty of the task were. Task satisfactions level immediately report about the difficult task, especially when compared to other tasks.
Test Confidence	After the usability test, ask the participants to answer a few questions about their impression on the overall usability scenario.
Test Confidence	After the usability test, ask the participants to answer a few questions about their impression on the overall usability scenario.
Inaccuracies	Record unintentional actions, slip, errors or Omissions that a user performs during a task. Write down every instance of an error with a Description. For example, "user bore the name in the first name." You can then categorize the severity of error or. Errors provide excellent diagnostic information and, if possible, should be associated with user interface issues.
Anticipation	Users have expectations about how difficult a task should be based on subtle cues in the task scenario. Users are now asking about the difficulties they face during task performance and compare it with actual estimates from the system user (same or different) may be useful in diagnosing problems.
Page visions / clicks	Hits were a strong correlation with the time on the task, which showed a good degree of efficiency. The very enlightening click to investigate a task success depends on the success or failure of the first click.
General metric (GM)	Sometimes it is easier to describe the usability of a system or task through a combination of measures into a single score. GM is mainly composed of three or more metrics.

Now, the focus is to figure out how to measure these properties for enhancing the quality in an agile environment.

2.2 CHARACTERISTICS IMPACTING SOFTWARE MAINTAINABILITY AND USABILITY

The characteristics that impacted the software maintainability are described below:

TABLE II. Characteristics that good maintainable software should possesses

Characteristic Name	Characteristic Meaning
Accuracy	The precision of computations and control
Completeness	The degree to which full implementation of required function has been achieved
Conciseness	The compactness of the program in terms of lines of code
Consistency	The use of uniform design and documentation techniques throughout the software development project
Data commonality	The use of standard data structures and types throughout the program
Error tolerance	The damage that occurs when the program encounters an error
Expandability	The degree to which architectural, data, or procedural design can be extended
Modularity	The functional independence of program components
Traceability	The ability to trace a design representation or actual program component back to requirements

The below characteristics have great impacts on software usability:

TABLE III. Characteristics that usable software should possesses

Characteristic Name	Characteristic Meaning
Communication commonality	The degree to which standard interfaces, protocols, and bandwidth are used
Execution efficiency	The run-time performance of a program
Hardware independence	The degree to which the software is decoupled from the hardware on which it operates
Operability	The ease of operation of a program

International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 10, ISSN No. 2455-2143, Pages 139-145
Published Online August - September 2016 in IJEAST (http://www.ijeast.com)

Security	The availability of mechanisms that control or protect programs and data
Self-documentation	The degree to which the source code provides meaningful documentation
Simplicity	The degree to which a program can be understood without difficulty
Software system independence	The degree to which the program is independent of non-standard programming language features, operating system characteristics, and other environmental constraints
Training	The degree to which the software assists in enabling new users to apply the system

III. RESULTS AND DISCUSSION

A variety of shortcomings and drawbacks are there in agile metrics used now-a-days. A most common issue experienced in the present set of agile metrics is that they may be liable to mix up project and process metrics. Some of them discuss about quality metrics while some others focus only on project metrics. Very few also talk about process metrics. All the suggested approaches and metrics needs to be fragmented over different authors leading to confusion and there is no logical and clear presentation of a comprehensive metrics set that clearly distinguishes and defines project, process and product metrics.

Another primary insufficiency is that most of these metrics not agile-centric, but adaptations of traditional metrics. Now-a-days the most widely used as well as recognized agile project metrics are the Agile EVM. However it too suffers from this intrinsic limitation in that it is a smart effort to adapt traditional metrics to somehow "fit" the agile model.

The solution lies in investigating the Agile Manifesto and building metrics based on the tenets of agile project management principles.

TABLE IV. Metrics based on Agile Project

Metric	Metric Description	Metric Type	Agile Tenet
Sprint effort factor	Sprint effort factor = (Items in current sprint/total feature list) + [∑ (change requests from previous sprints)]. *Sprint effort factor should be evenly spread through all sprints.*	Project Metric	Working software over comprehensive documentation.
Sprint complexity factor	Sprint effort factor = *f* (modules it interacts with # of interface points with other modules.	Project Metric	Working software over comprehensive documentation.
Change request effort	Change request effort = *f* (adding new features + changing previously defined features - deliberate elimination of features).	Project Metric	Customer collaboration over contract negotiation.
Customer expectation baseline	Customer expectation baseline = (minimal set of expectation features from the sprint).	Project Metric	Customer collaboration over contract negotiation.
Impact on budget	Impact on budget = *f* (change request effort, customer expectation baseline.	Project Metric	Customer collaboration over contract negotiation.
Reusability Factor X	Identifying reusable components in system = # of components added to library. *The general guideline is that higher is better. This metric aims to identify more reusable components within the system.*	Product Metric	Responding to change over following a plan.
Reusability Factor Y	Reuse of reusable components in system = # of components reused from library. *The general guideline is that higher is better. The rational is that good system architecture makes more use of reusable components leading to a higher quality product.*	Product Metric	Responding to change over following a plan
Facetime	Facetime = *f* (time each developer is with business person and with other developers on whom their work is dependant).	Process Metric	Individual and interactions over processes and tools.

International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 10, ISSN No. 2455-2143, Pages 139-145
Published Online August - September 2016 in IJEAST (http://www.ijeast.com)

IV. DESIGN ISSUES FOR PROPOSING A MODEL

The traditional approach to develop any software is a layered approach in which the completed software is delivered in last to fulfill customer requirement. If any further changes are required by the customer then it is hard to retain within prescribed budget and schedule but agile uses the functional approach to develop software in which the customer is allowed to adjust budget and schedule at each recurrence according to stand-alone deliverables. The following issues are faced during proposing a model in agile environment.

1. Problem Recognition Time
2. Administrative Delay Time
3. Tool Time Collection
4. Find problem solving
5. Hypothesis Correction time
6. Proposed model

Software Maintainability and Usability of the suggested model address to improve the late changing requirements of software development. Agile processes control change for the customer's competitive advantage. The major success measure for increasing assurance is the working software. Agility is promoted by continuous concern to nominal quality and good scheme. Periodically usability will be able to identify problems better and adjusts them. The proposed model should be good in the agile environment through the implementation of the concept of maintaining serviceability should be focused.

4.1 TEST-DRIVEN DEVELOPMENT (TDD)

The core part of the agile code development approach constrained from Extreme Programming (XP) and the principles of the Agile Manifesto is Test-driven development (TDD).

According to text, TDD is not all new; a previous reference to the use of TDD is the NASA Project Mercury in the 1960's.

As its name symbolizes, TDD is not a testing procedure, but rather it is a development and design technique in which the tests are previously written to the production code. The tests are progressively appended during the implementation and when the test is passed, the code is re-factored for the enhancement of the internal structure of the code. This cycle is repeated until whole functionality is implemented. The TDD cycle consists of the following six fundamental steps:

1. Write a test for a piece of functionality,
2. Run all tests to observe the new test should fail,
3. Write code that passes the tests,
4. Run the test to verify they pass,
5. Re-factor the code and
6. Run all tests to see the refactoring did not change the external behavior [5].

4.2 CONTINUOUS REFACTORING

Refactoring is a significant aspect of the development process for programmers working together in a team. The reason for this is that everyone in the team needs to be able to easily read and understand the code. Code that is not re-factored is often hard to read and understand [4].

The process of clarifying and simplifying the design of existing code without changing its behavior is known as refactoring. Agile teams are maintaining and expanding their code much from iteration to iteration and without continuous refactoring, which is hard to do. This is because undisturbed code tends to deterioration. Deterioration takes several structures: unhealthy dependencies between classes or packages, bad allocation of responsibilities class, too much responsibility for a class or method, duplicate code, and many other sorts of confusion and disorder.

4.3 COLLECTIVE CODE OWNERSHIP

A process in which everyone is responsible for all of the code, which means that everyone is entitled to any change is called Collective code ownership. Pair programming contributes to this practice: working in different pairs, all programmers have the opportunity to see all the parts of the code. A big advantage for collective ownership claimed that it speeds up the development process, because when an error occurs in the code any programmer can fix it. It is a collaboration built upon high-performance, mutual respect and deep trust [6].

Following two measures should be taken to recognize nonconformities to the collective code ownership,:

- Semantic factor assessment project truck.
- Syntactic membership activities defined by switching pair

4.3.1 TRUCK FACTOR

The truck number (or truck-factor) is the numeral amount of people with knowledge; it cannot change if the number of persons went under a truck at the same time it would not be able to continue to develop.

The truck factor (TF) of a project can be defined as “the number of developers on a team who needs to be hit with a truck before the project is in serious trouble”. Clearly, “to be hit with a truck” is an acute thought that can be substituted with more realistic ones such as, for example, to go on vacation, to become ill, to be out of the office or to leave the company for another. Ideally, to avoid potential problems, as advocated by the Extreme Programming (XP) principles of “Collective code Ownership”, the Truck Factor of a project should be as high as possible [9].

The project in serious trouble of course do not really need to be run over by a truck, it could leave the company ill or on holiday.

- A higher number is better truck
- A low truck number is worse

International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 10, ISSN No. 2455-2143, Pages 139-145
Published Online August - September 2016 in IJEAST (http://www.ijeast.com)

4.3.2 SWITCHING PAIR

Pair programming is a style of programming in which two programmers performs their job side-by-side at one computer, continuously collaborating on the same design, algorithm, code, or test [7]. Pair programming has been practiced sporadically for decades [7]; however, the emergence of agile methodologies and Extreme Programming (XP) [8] has recently popularized the pair programming practice.

Proponents of pair programming ("pair") argue that this increases the long-term productivity by significantly improving the quality of the code. But it is fair to say that for a number of reasons, voting is the most controversial and less widely believed agile practices for programmers.

4.4 TRENDS

It is important for application owners to see continuous improvement in an application over the track of successive sprints in an Agile environment. It is likely to see a favorable trend, where iteration of the application is better than the last. This makes it important to monitor application performance trends in terms of requirements. Trending reports allows giving stakeholders regular snapshots of performance, which should ideally show that performance is getting progressively better or it is not degrading at least.

4.5CONTINUAL ANALYSIS

Continual analysis is important in agile processes. Especially when it comes to application functionality and performance, both contributors and stakeholders require maintaining a close track of the progress of the project. Performance analysis should be both continual and comprehensive to provide them the observation they require. Analysis takes place all the way down to the routine scrums that include IT infrastructure and performance testers as contributors and application stakeholders.

Contributors are those active and dynamic members of the sprint team who participate in daily scrums, which give all stakeholders visibility into the present state of the development attempts and effort. When all interested team members know the performance of each sprint, they are in a better situation to maintain the quality of the whole application high. As soon as the problems are found, they can be fixed more early.

4.6 THE DIVERSE EXPERTISE

In the field of IT projects there are a number of diverse expertise that makes up a development team. For example, a typical software development team can include programmers, database administrators, network administrators, security experts, testers, user interface designers, and others. While the expertise diversity of a software development team strengthens the team as a whole, this diversity is often the source of a cultural quality impact [10].

According to our research it can easily trace its maintainability and usability with the help of above metrics if it monitors these properties regularly of any project in agile environment.

V. CONCLUSION AND FUTURE WORK

The limitless writing on programming measurements proposes various methods for measuring programming without giving a traceable and significant interpretation to the multi-faceted thought of value.

Specifically, the Maintainability and Usability Index experiences extreme constraints with respect to underlying driver investigation, simplicity of calculation, dialect autonomy, comprehend capacity, clarify capacity, and control.

A well-picked choice of measures and rules for accumulating and rating gives a helpful extension between source code measurements and the quality attributes.

Light-footed is helpful in the event of programming Maintainability and ease of use as it is conceivable to convey the Working programming inside the briefest conceivable time by utilizing the light-footed. And in addition it builds the consumer loyalty and trust in the individual organization.

This examination utilizes writing to reason about the relationship between deft improvement techniques and practicality or convenience. Future work should be possible so as to accept the discoveries exhibited in this exploration, by setting up a trial to explicitly test the effect of advancement strategies on practicality or usability. Prior experimental examination has not managed expressly with this relationship. Rather, most experimental exploration has concentrated on other particular perspectives, for example, software engineer profitability and blunder check, measured for the most part for the short term. It is fascinating to quantify the measure of hours required for keeping up a system created utilizing lithe techniques when contrasted with a project created utilizing a customary arrangement driven methodology over quite a while.

In coordinated, there exists ceaselessly contact with client, so as indicated by the need of client, the new elements can be acquainted with fulfill client prerequisite and which will make us to go ahead the track to decrease the expense and time if there is any sort of lacking from the arranged cost and calendar.

VI. REFERENCES

[1] Soumi Ghosh, Sanjay Kumar Dubey, Prof. (Dr.) Ajay Rana , Comparative Study of the Factors that Affect Maintainability, International Journal on Computer Science and Engineering (IJCSE) ISSN: 0975-3397 Vol. 3 No. 12 Dec 2011

[2] L. Trenner and J. Bawa, The Politics of Usability, Springer-Verlag, London, 1998.

[3] J. E. Gaffney, Jr.:"Metrics in software quality assurance". January 1981, ACM 81: Proceeding of the ACM '81 conference.

[4] Johan Nisula, Increasing Continuous Refactoring in Agile Projects using Pair Programming **,** MARCH 4, 2014

[5] Shrivastava and Jain, "Metrics for Test Case Design in Test Driven Development", International Journal of Computer Theory and Engineering, Vol.2, No.6, December, 2010, Pg: 1793- 8201.

[6] Ken H. Judy, Ilio Krumins-Beens, Great Scrums Need Great Product Owners: Unbounded Collaboration and Collective Product Ownership, Proceedings of the 41st Hawaii International Conference on System Sciences - 2008

[7] L. Williams and R. Kessler, Pair Programming Illuminated. Reading, Massachusetts: Addison Wesley, 2003.

[8] Beck, Extreme Programming Explained: Embrace Change, Second ed. Reading, MA: Addison-Wesley, 2005.

[9] Filippo Ricca, Alessandro Marchetto, Marco Torchiano, On the difficulty of Computing the truck factor, Volume 6759 of the series Lecture Notes in Computer Science pp 337-351, Springer.

[10] Ambler, S. (2008). When IT gets cultural: data management and agile development. IT Professional, 10(6), 11-14. Retrieved February 7, 2011, from IEEE Computer Society Digital Library database.