Software Maintainability and Usability in Agile Environment

Monika Agarwal
M.TECH, CSE
Amity University, Noida

Rana Majumdar
Assistant professor
Amity University, Noida

ABSTRACT
This research is based on software maintainability and usability in the agile environment. Maintainability of the system is the ability to undergo changes relatively easily. These changes can affect components, services, interfaces and functionality when adding or changing functions, errors, and respond to business needs. Usability is defined as the application that meets the requirements of users and consumers by providing an intuitive, easy to locate and globalize and provides good access for disabled users and leads to a good overall user experience. In the conventional method of the software development, there are many metrics to calculate the maintenance and use of software. This research is to determine whether the same measures apply to Agile, or there is a need to change some metrics used for the agile environment.

The goal of software engineering is to develop good quality maintainable software in schedule and budget. Inflated software costing, delayed time frame, or not meeting quality standards express a failure. A survey suggests about 45% of software fails due to the lack of quality. It is therefore one of the most important aspects for the success of software. .

General Terms
Agile, software Maintenance and usability

Keywords
Software Maintainability, software usability, agile environment, software metrics.

1 INTRODUCTION
Agile development is a software method, which is based on iterative and incremental methods for software development. The individual modules are built by small teams. As they are developed, it will be sent to the client for review. This model is robust and flexible, which includes changes based on customer needs, with an emphasis on teamwork and freedom of the developer. Based on the principles and practices, different agile development methodologies are used for software development.

Maintainability is concerned with the duration of maintenance outages or how long it takes to reach (quick and easy) maintenance compared to a datum. The benchmark includes maintenance which will be carried out by persons with specified level of knowledge, with prescribed procedures and resources required at each level of maintenance. Maintainability characteristics are generally identified by the design of equipment maintenance procedures which define and determine repair time. There are 4 types of maintenance:

1.1 Corrective Maintenance
This started when defect is found in the software.

1.2 Adaptive Maintenance
It includes the modification of software in response to changes in the environment, sustainable development.

1.3 Perfective Maintenance
It means improving processing efficiency or performance, or restructuring the software to improve changeability. This may include enhancement of existing system functionality, improvement in computational efficiency etc.

1.4 Preventive maintenance
These are long-term effects of the correction, adaptive and improved changes. This leads to an increasing complexity of software that reflects the deteriorating structure. The work needs to be done to maintain or to reduce it if possible. This work may be designated as preventive maintenance.

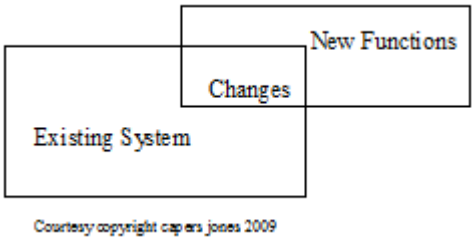

Fig 1: Software maintenance image representation

Application interfaces with the user in mind and the consumer must be designed so that they are intuitive, can be localized and globalized, access for disabled users and provide a good user experience overall. Key issues for the user experience and usability are:

- Too much Interaction (too many clicks) for a task requires. Make sure that the screen design and inflows and modes of interaction with the user in order to maximize ease of use.

- Recognize the wrong steps in the multi-level interfaces. Verify that appropriate workflows to optimize operations in several stages.
- Data and controls are grouped incorrectly. Select the appropriate control type (such as sets of options and check boxes) and place controls and content using models accepted user interface design.
- User comments are bad, especially for errors and exceptions, and the application is not responding. Consider enabling the implementation of technologies and techniques, the maximum interactivity, such as Asynchronous JavaScript and XML (AJAX) in web pages and client-side input validation.

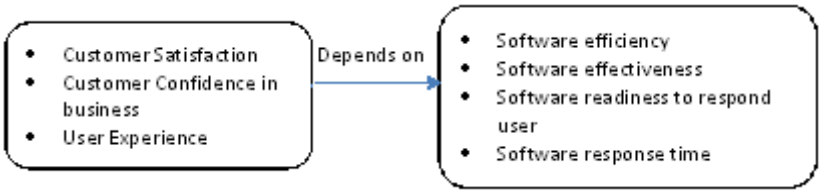

Fig 2: Software Usability image representation

2 RELATED WORK

This research focuses on the modelling of software maintainability and usability in an agile environment, how to measure the indicators for maintenance and ease of use in the traditional working environment presented in an agile environment. This work deals with the modelling approaches and how best to implement such measures, and simplified approaches should be used as a model and as a model for other benefits on agile projects.

The investigation has been identified in a literature review, analysis and aggregation of identified ideas. It was found that no such previous research exists. Additional literature was either general, as the basis of the use of software metrics in agile methods or more concentrated aware specific measures or aspects of agile software development. Estimate; quantify the effectiveness of available resources, productivity measurement and modelling for maintenance is the focus of research. Our proposed work is taken into account domain specific knowledge regarding the individual in the absence of full suite of tests and other aspects such as code analysis and analysis of historical data, counting, task decomposition, measurement complexity.

Antonellis et al. [1] proposed a method for **mapping of object-oriented source code metrics maintainability** features according ISO 9126. The parameters are selected from the series of Chidamber and Kemerer [2].

Agile deployment model works by the organizational framework and objectives:

1. Assessing the potential of Agile practices, methodologies and tools
2. Some Deployment Planning for agile practices and pilot project to select
3. The first pilot deployment project prepared
4. Post-iteration iterative feedback
5. Continuous improvement of the organizational practices

Once it reaches the first deployment phase, it begins pilot project maintainability.

Broy et al. [3] have independently developed a **model of maintainability** in which maintenance activities are strictly separated from facts about the system being maintained. Both activities and facts are organized into hierarchical trees whose leaves are related through a (weighted) matrix that indicates which atomic facts influence each atomic activity.

Oman et al. [4] suggested a **hierarchical structure of measurable properties of maintainability**, based on an analysis of 35 publications. You create software metrics specific to the leaves of the tree and propose a formula for combining them into a single index.

Jane Huffman Hayes, Naresh Mohamed, Tina Hong Gao [17] presented a technique **Observe-mine-adopt (OMA): an agile way to enhance software maintainability** in which they introduce two maintainability measures, maintainability product and perceived maintainability. Mining activities lead to validated discoveries of processes, techniques or practices that improve the software product; they are formalized and adopted by the team. OMA has been studied experimentally using two project studies and a Web-based health care system which is maintained by a large industrial software organization.

Wiebe Hordijk, Roel Wieringa [18] Faculty of Electrical Engineering Mathematics and Computer Science University of Twente in his Research Design **Surveying the Factors that Influence Maintainability** establish a solid theory of maintainability; the factors that influence maintainability .They investigate which design decisions influence maintenance effort for software systems. They conclude that higher the quality of the documentation, more maintainable the system will probably be and measured documentation effort as part of change effort.

Software metrics provide to recognize a simple and cost-effective manner and to correct possible causes of the low quality of the products to the maintenance factor based perceived by programmers. Implementation of measurement programs and metric standard will help to prevent errors before servicing and reduces the effort required at this stage. Internal parameters are strongly correlated with the opinion of the programmers. However, dissatisfaction with internal quality standards are not lead to necessarily low maintenance, but it is generally expected and, more importantly, how the effectiveness of available resources can be quantify.

2.1 Metrics for the maintainability and usability

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

TABLE I. Metrics to Measure software Maintainability

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 colour 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, 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.

It is a proven fact that agile development in order to reduce the complexity and the number of defects, which has a positive influence on the maintainability seems.

Now some measures and their impact on software usability are discussed below:

TABLE III. Metrics to Measure software Usability

Metric Name	Importance and their effect on the Usability
Task Completion Degree	It is used to determine the degree of fulfilment of the tasks these metrics measure can be measured. It is usually stored as a binary metric (1 = success working and 0 = task failure).
UI issues	Firstly describe the problem, and note how often encountered and by which users. 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.
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 Limitations of the research

The underlying idea is that it is not possible to find relationship of cause and effect which can be detected with traditional statistical approaches to software metrics. In this research, the aim is to explore how this approach is related to traditional statistical approaches and if these notation schemes could be taken up with them. This research needs to be followed, and hope to incorporate the new standard into proposed maintainability and usability model.

- Especially literature on software maintenance is from the 1980s or early 1990s. The field of software engineering has changed since that time, and the factors that may affect the maintenance process to be updated.

- In practice, the focus is on extreme programming and only in close combat very high standard with all other existing agile development methods.
- Relationships are identified through empirical research does not support strongly, but are based on the properties and the philosophy of agile development methods. The drawback with this approach is that the philosophies and characteristics presented by the authors of the Agile Manifesto, and cannot completely impartial.
- How the number of hours required for a program developed using agile methods in relation to a program with a plan to the traditional approach for a long time developed based hold can be measured.
- In addition, other methods for predicting how algorithmic techniques and no other algorithms and their applicability to predict maintenance are examined.
- What are the factors and parameters as predictors of the maintainability of software applications have been studied? Which of these predictors are to be successful?
- The model can estimate the software development of practical software maintenance be extended and iterative and agile methods of software development.
- How to stay involved throughout the development process, to ensure that correct usability problems when agile environment is used in the next sprint addressed?
- How agile solves the usability problem may prevent communication that the teams have on complete solutions in a distributed environment.

3 RESEARCH QUESTION

The team will use the best practices in as many programs as could find in community that are using or have used Agile for software development. For this report, the concern is to deal with software only or software intensive systems. This research was focused to answer the below two questions:

• Is the use of Agile has advantage in the case of maintainability and usability of the software?

• Will it produce a better end product developed with maintainability and usability within cost and schedule parameters?

This Research will use the best approaches to answer the two questions at the same time because believe is that regardless of one of the benefits of Agile environment (and it is quickly apparent that there were many), it would only remain academic interest if there will not be any solid experience in the use of effective Agile environment.

4 CURRENT RESEARCH

The software quality is affected by various factors that can be measured directly as well as indirectly. The indirectly factors includes usability (related to product operation) and maintainability (related to product revision).

4.1 Characteristics affecting software Maintainability and Usability

The characteristics that impacted the software maintainability are described below:

TABLE IIIII. 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 IVV. 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
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

4.2 DESIGN ISSUES FOR PROPOSING A MODEL

The conventional approach to develop any software can be described as a layered approach in which the completed software to fulfil customer requirement is delivered in last so if it needs an further changes, It is hard to maintain within prescribed budget and schedule but agile uses the functional approach to develop a software that allows the customer to adjust budget and schedule at each repetition 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

The proposed model of software Maintainability and Usability should be able to get the old practices of software maintainability and usability and address to improve the late changing requirements of software development. Agile processes harness change for the customer's competitive advantage. It is better to provide working software frequently from a few weeks to a few months, with the shortest possible timescale. The major success measure for increasing confidence is the working software. Agility is promoted by unceasing care to nominal quality and good scheme. Periodically usability will be able to identify problems better tunes and adjusts them. The focus is that the proposed model should be good in the agile environment through the implementation of the concept of maintaining serviceability.

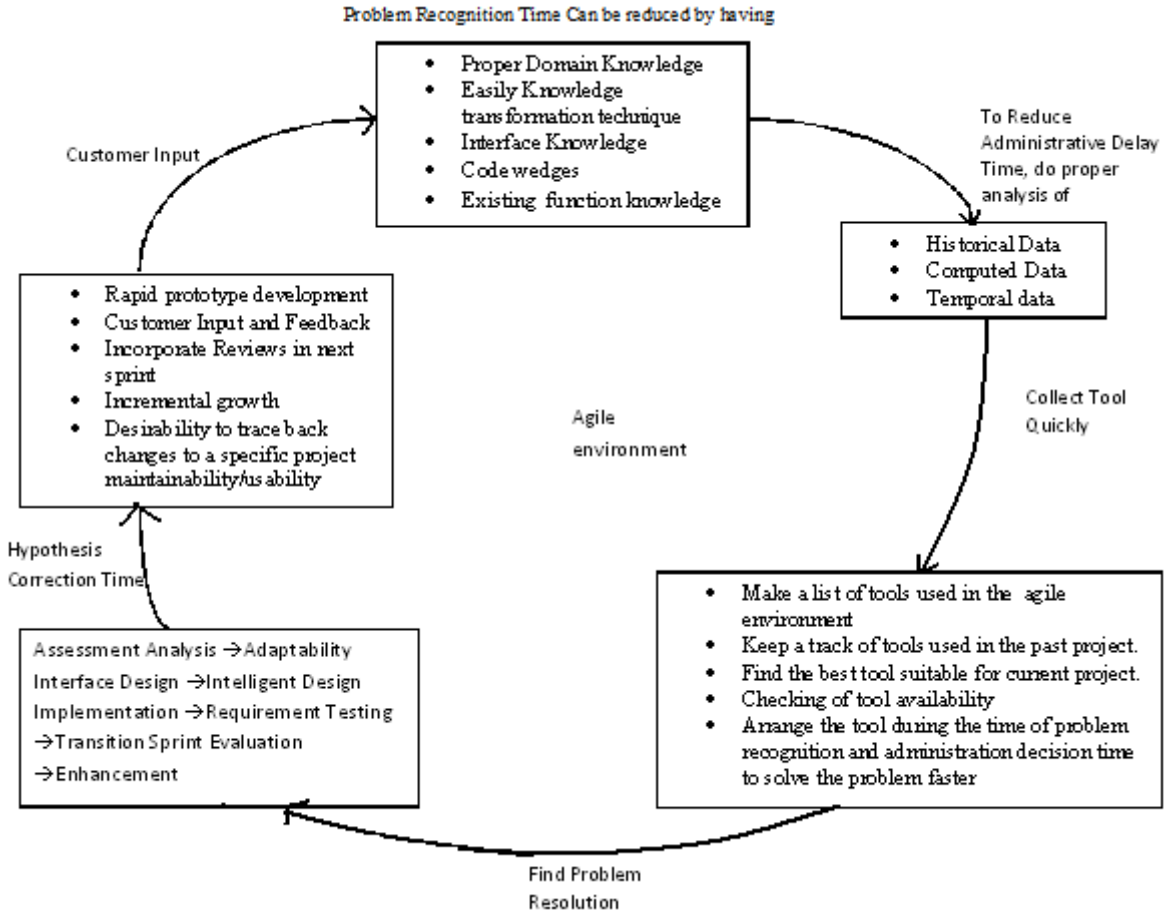

Fig 3: Proposed Model

5 RESULT

This research is under affect the properties of the agile development methods, maintainability and use:

5.1 Test-Driven Development (TDD)

In agile software development process, Test Driven Development (TDD) depends on the recurrence of a precise little progress phase: the developer writes a case firstly, automated tests that defined an improvement desired or new function, then the least possible code is written to pass this test, and finally the new code refactors to acceptable standards.

The time for the test programming concepts first extreme programming is created, started in 1999, but also, more recently, programmers will use it to refine the service of the unwanted code developed by the conventional methods.

5.2 Continuous Refactoring

Refactoring is the process of clarifying and simplifying the design of existing code without changing its behavior. 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 rot. Red takes several forms: 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.

Every time the code is changed without refactoring, code it is getting worse and spreading. Code red frustrates us, costing us time and reduces your life of useful systems. In an Agile context, it can mean the difference between meeting and not meeting deadline iteration.

Refactoring means that the code is easy to maintain and extend. This extensibility is to check on the design and measurement of success. But it should be noted that only "safe" refactor the code now far if the choice is to have large suites of unit tests when the working style is test-first. Without the ability to run these tests after each small step in a refactoring, there is a risk of introducing bugs. If you do so you have no choice regular refactoring, because that is how you change the theme real test driven development (TDD), in which the design is constantly changing.

5.3 Collective code ownership

Collective code ownership means that everyone is responsible for all of the code, which means that everyone is entitled to change anything. 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.

By all programmers the right to change the code, there is the risk of errors by programmers, they know what they are doing, thinking introduced, but no specific dependencies. Unit tests sufficiently well defined in order to solve this problem: If the dependencies to create unexpected errors when the unit tests are running, they show losses.

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

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

5.3.1 Switching pair

Agile teams, in frequent, regular, high-quality production, are striving to find ways to enhance productivity, maintain the short and long term, the highest possible. 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.

5.3.2 Truck Factor

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

An informal action (if you can call it that) is the "Truck Factor" team. Truck-factor measures the amount of spread of knowledge within the team.

Formally, the truck is the number of people that need to be run over by a truck before. 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

5.4 Continues analysis

In agile processes, the continuous analysis is important.

Both staff and participants should keep a watchful eye on the progress of the project, especially when it comes to application functionality and performance. To them the perspective they need should be permanent both performance analysis and comprehensive. The current analysis is used to identify problem areas in the application. The analysis is done during the day and mixed IT infrastructure and performance Tester as participants and actors in the application.

The employees are involved are active members of the team sprint and have everyone look at the current state of development effort. If all team members on the performance of each sprint are concerned, they are in a better position to maintain the quality of the application. More problems are found, the sooner they can be solved.

5.5 Trends

In an agile environment, it is important for the application owner to see continuous improvement in demand during successive sprints. You want a positive trend; the iteration of the application is seen well than the last time. It is therefore even more important to monitor trends in the application performance in relation to the needs. Trends reports you can stakeholders. Regular performance snapshots that should ideally point that performance are getting better or at least not worsen.

5.6 The difference Expertise

As in many other professions in the field of IT projects, there are a number of different silos expertise, the development team for example, a typical software development team of programmers, administrators, database, network administrators, security experts, testers, UI designers, and others. While the diversity of expertise of a team in software development joins the team as a whole, this diversity is often the cause of a cultural impact quality.

6 CONCLUSION AND FUTURE WORK

In particular, the metrics listed by the accompanying technical reports can at best establish the degree of maintainability and usability of a system after the fact. The vast literature on software metrics, on the other hand, proposes numerous ways of measuring software without providing a traceable and actionable translation to the multi-faceted notion of quality.

In particular, the Maintainability and Usability Index suffers from severe limitations regarding root-cause analysis, ease of computation, language independence, understand ability, explain ability, and control.

A well-chosen selection of measures and guidelines for aggregating and rating them can, in fact, provide a useful bridge between source code metrics and the quality characteristics.

The maintainability of agile approach is constrained by several factors such as project size and type, experience of project personnel, and the availability of knowledgeable and committed customers.

Agile is beneficial in case of software Maintainability and usability as it is possible to deliver the Working software

within the shortest possible time by using the agile. As well as it increases the customer satisfaction and confidence in the respective company.

This research uses literature to reason about the relationship between agile development methods and maintainability or usability. Future work can be done in order to validate the findings presented in this research, by setting up an experiment to specifically test the impact of development methods on maintainability or usability. Prior empirical research has not dealt explicitly with this relationship. Instead, most empirical research has focused on other specific aspects such as programmer productivity and error count, measured mainly for the short term. It would be interesting to measure the amount of hours required for maintaining a program developed using agile methods when compared to a program developed using a traditional plan-driven approach over a long time.

In agile, there exists continuously contact with customer, so according to the need of customer, the new features can be introduced to satisfy customer requirement and which will make us to come on the track to reduce the cost and time if there is any kind of lacking from the planned cost and schedule.

7 ACKNOWLEDGMENTS

I would like to thank Professor Rana Majumdar for all the help he offered in the course of my studies. His encouragement and guidance have been of great value to me Personally, I would also like to thank my parents and my partner for their support in my studies.

8 REFERENCES

[1] P. Antonellis, D. Antoniou, Y. Kanellopoulos, C. Makris, E. Theodoridis, C. Tjortjis, and N.Tsirakis, "A data mining methodology for evaluating maintainability according to ISO/IEC-9126 software engineering – product quality standard," in Special Session on System Quality and Maintainability - SQM2007, 2007.

[2] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design." IEEE Trans. Software Eng., vol. 20, no. 6, pp. 476–493, 1994.

[3] M. Broy, F. Deissenboeck, and M. Pizka, "Demystifying Maintainability," in Fourth International Workshop on Software Quality Assurance (SOQUA 2007). ACM, 2007.

[4] P. Oman and J. Hagemeister, "Metrics for assessing a software system's maintainability," in Proceedings of Conference on Software Maintenance, 1992., Nov. 1992, pp. 337–344.

[5] www.elsevier.com/locate/infsof Empirical studies of agile software development: A systematic review Tore Dyba, Torgeir Dingsøyr SINTEF ICT, S.P. Andersensv. 15B, NO-7465 Trondheim, Norway

[6] e-Informatica Software Engineering Journal, Volume 1, Issue 1, 2007.

[7] Agile Methods and CMMI: Compatibility or Conflict? Martin Fritsch_, Patrick Keil_ _Technische Universit¨at M¨unchen.

[8] The Agile Business AnalystBy: Mike Cottmeyer, V. Lee Henson.

[9] How software process improvement standards and agile methods co-exist in software organizations? Ngoc Tuan Nguyen, University of Twente. n.t.nguyen-1@student.utwente.nl Enschede, August 2010.

[10] Process Improvement, the Agile Way! Ben Linders, Senior Consultant, www.benlinders.com

[11] The Journal of American Science, 4(1), 2008, ISSN 1545-1003, americansciencej@gmail.com A Framework for Agile Methodologies for Development of Bioinformatics SYED Ahsan, Abad SHAH R & D Center of Computer Science University of Engineering and Technology , Lahore, Pakistan ,Corresponding author: Syed Ahsan

[12] Capturing the Requirements,Shari L. Pfleeger,Joanne M. Atlee .

[13] C. G. O'Regan, A Practical Approach to Software Quality, Springer, New York, NY, USA,

[14] Agile assessment Framework © Copyright Agile VTT Minna Pikkarainen Version_1.0

[15] Agile Software Development of Embedded Systems Version : 1. 0 Date : 2005.04.04 Pages 44 Authors Minna Pikkarainen Tua Huomo

[16] The Challenges to the Safety Process When Using Agile Development Models, Master Thesis Hanne-Gro Jamissen June 29, 2012 Halden, Norway.

[17] Observe-mine-adopt (OMA): an agile way to enhance software maintainability Jane Huffman Hayes , Naresh Mohamed and Tina Hong Gao . Journal of Software Aintenance and Evolution: Research and Practice

[18] Surveying the Factors that Influence Maintainability Research Design Wiebe Hordijk Roel Wieringa Faculty of Electrical Engineering Mathematics and Computer Science University of Twente www.cs.utwente.nl/roelw roelw@cs.utwente.nl