Review on Functional and Non-functional requirement using Machine Learning Approaches

Neha Dhiman, Er Mohan Singh
Deptt.Of Computer Science & amp; Engg. HPTU Hamirpur
E-mail- nehadhimanneha333555@gmail.com
Assistant Professor HPTU Hamirpur
E-mail- mcamohanchoudhary@gmail.com

Abstract- Agile software development is defined as rapid development and delivery of the software in the requirement changing environment which is well suited for present day business scenario. So requirement analysis and classification is important task for the agile method in this paper review on different supervised and unsupervised learning on requirement classification and reusability.

Keywords- Machine learning, Supervised learning, Clustering

I. INTRODUCTION

Agile software development is defined as rapid development and delivery of the software in the requirement changing environment which is well suited for present day business scenario. Working software measures the progress. Basically, Agile method involves interleaving the specification, implementation, design and testing. Series of versions are developed with the involvement of and evaluation by the stake holders in each version. Agile methods aim at reducing the software process overheads (like documentation) and concentrate more on code rather than the design. Customer involvement, incremental delivery, freedom of developers to evolve new working methods, change management, and last but not the least simplicity is the basic essence of Agile development. Agile methodologies are well suited for small as well as medium sized projects. However, uniform customer involvement throughout the project, appointing appropriate team to adapt to Agile methodology, ranking of changes to be accommodated in software, maintaining simplicity, difficulty in scaling Agile procedures to larger projects and deciding upon the contract terms account for the major disadvantages involved in the Agile development.

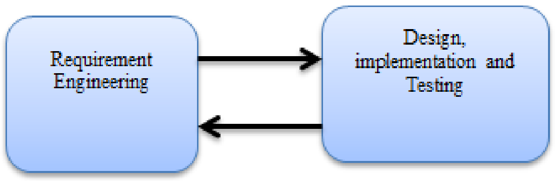

Fig 1**:** Agile development essence Agile Methods

Several agile methods have been developed till date like Extreme Programming, Scrum, Dynamic Systems Development Method, and Adaptive Software

Development, Feature-Driven Development, Crystal, Lean Software Development, Kanban, Agile Modeling, Agile Unified Process, etc.

Text Processing:

General text preprocessing is nothing but preparing the available text for computer analysis. In involves steps that prepare the text for computer understandable representation and extracts the necessary useful matter from the text. There are mainly three steps involved in text preprocessing, namely:

- Tokenization: This involves the process of converting a stream of characters into tokens (generally word tokens). Delimiters like spaces, punctuations etc. are used for separating one word token from another.
- Stop-word Removal: In this step, words that carry negligible or little meaning for the sentence like ‘is’, ‘of’, ‘and’, ‘the’, etc. are removed.
- Stemming: It is the process of reducing words in the word stem to its root form like ‘writes’, ‘writing’, ‘wrote’, ‘written’ these all correspond to a single root “write” [37].

Clustering: As discussed by [38], Clustering is nothing but partition of data into sets of similar items. Each of the sets is called a cluster. Document clustering aims at increasing cohesion in a single cluster and minimizing coupling between two or more clusters i.e., trying to reduce the intra-cluster distance and increase inter-cluster distance. Clustering is considered to be a part of unsupervised learning. Three most popular clustering methods are described below:

- K-Means: It is the simplest flat and hard clustering algorithm. This algorithm’s objective function tries to minimize average squared distance of items from the cluster centers (which is mean of items in a cluster). This method is best known for its simplicity and efficiency.
- Expectation Minimization: It is a flat model-based clustering technique which assumes data to be generated by a model and tends to recover that original model from data. This original model further describes clusters and cluster membership of data. It is considered to be

generalization of K-means technique. It has alternating expectation step and maximization step.

- Hierarchical Clustering: As described by [39], hierarchical technique creates a nested structure of partitions with an all-inclusive single cluster at root and singleton clusters of singular points at bottom. Every intermediate level is built combining the two clusters from lower level or splitting the top level cluster into two. Two main approaches of hierarchical clustering are:
    - Agglomerative: It is a bottom up approach. Points are considered as individual clusters at the starting. At each step, most similar clusters are merged according the cluster similarity/distance definition. These are known for their quality. Examples of agglomerative techniques are Intra-cluster Similarity Technique (IST), Centroid Similarity Technique (CST) and Unweighted Pair Group Method with Arithmetic Mean (UPGMA). F-measure for UPGMA is better than the other two.
    - Divisive: It is a top down approach. Process starts with root cluster and is split until singleton clusters of individual points are formed. At each step, decision of which and how cluster should be split is made.

II. LITERATURE REVIEW

John Mylopoulos et al [1]: To propose a comprehensive process oriented qualitative framework that integrates non-functional requirements into the process of software development. To illustrate the application of proposed methodology by taking examples of accuracy requirements in design phase and performance requirements in implementation phase for information systems. Evidence for the power of the framework is provided through the study of accuracy and performance requirements for information systems.

A. Eberlein and J. C. Leite [2]: Agile methods are an attractive alternative for those pressured to produce code fast. Many programmers like the hands-on strategy of these approaches which also help them avoid some of the less exciting tasks, such as specification. On the other hand, some people appear to welcome agile methods as an excuse to throw overboard everything that requirement engineering has been teaching. This position paper looks at numerous aspects of requirements engineering and argues about their suitability for agile approaches. The aim is to elicit lessons from requirements engineering that agile methods might consider, if quality is a major concern.

F. Paetsch, A. Eberlein and F. Maurer [3]: This article compares traditional requirements engineering approaches and agile software development. Their paper analyzes commonalities and differences of both approaches and determines possible ways how agile software development can benefit from requirements engineering methods.

N. S. Rosa et al[4]: Non-functional requirements (NFRs) are rarely taken in account in most software development processes. There are some reasons that can help us to understand why these requirements are not explicitly dealt with: their complexity, NFRs are usually stated only informally, their high abstraction level and the rare support of languages, methodologies and tools. In this paper, they concentrate on defining how to reason and how to refine NFRs during the software development. Their approach is based on software architecture principles that guide the definition of the proposed refinement rules. In order to illustrate their approach, they adopt it to an appointment system.

L. Chung and J. C. S. do Prado Leite [5]: Essentially a software system's utility is determined by both its functionality and its non-functional characteristics, such as usability, flexibility, performance, interoperability and security. Nonetheless, there has been a lop-sided emphasis in the functionality of the software, even though the functionality is not useful or usable without the necessary non-functional characteristics. In this chapter, they review the state of the art on the treatment of non-functional requirements (hereafter, NFRs), while providing some prospects for future directions.

S. Farhat et.al [6] This work recognizes four NFR sorts and gives the philosophy for creating space particular NFR by utilizing procedures for changing over the necessities into outline ancient rarities per NFR sort. The commitment is four NFR sorts: Functionally Restrictive, Additive Restrictive, Policy Restrictive, and Architecture Restrictive and the software engineering process that gives particular refinements that outcome in one of a kind compositional and plan curios. By applying the same utilitarian prerequisite center to the distinctive NFR areas it upgrades the improvement process and advances software quality characteristics, for example, compensability, viability, resolvability, and traceability.

Taehoon Um et.al.[7]They proposed a lightweight quality evaluation method for an lithe way to deal with reflect non-functional aspects. Their approach bolsters early distinguishing proof of non-functionality, and makes a difference members reliably continue focusing on quality qualities. Members get inputs for the following discharge by the evaluation consequences of demonstrating unsatisfied quality traits in each discharge. Besides, members can make anticipates quality change. Be that as it may, members have their claim criteria when they lead evaluation with prototypes.Accordingly, the proposed evaluation method could be subjective. Accordingly, it is expected to make a quantitative evaluation show, utilizing estimation measurements to enable members to have more trust in the outcomes.

Weam M. Farid et.al.[8] This examination exhibits a lightweight building of NFRs for agile processes. The proposed Non-functional Requirements Modeling for Agile

Processes (NORMAP) Strategy recognizes, connections, and models Agile Loose Cases (ALCs) with Agile Use Cases (AUCs) and Agile Choose Cases (ACCs). A lightweight adjusted adaptation of the NFR System was created including 25 critical NFRs. Further, a hazard driven agile requirements usage arrangement and a visual tree-like view were created. The procedure was approved through building up a Java-based modeling reproduction apparatus and two contextual investigations. W. M. Farid and F. J. Mitropoulos [10]: This research proposes NORMATIC, a Java-based simulation tool for modeling non-functional requirements for semi-automatic agile processes. NORMATIC is the semi-automatic tool that supports the more general Non-Functional Requirements Modeling for Agile Processes (NORMAP) Methodology. Early results show that the tool can potentially help agile software development teams in reasoning about and visually modeling NFRs as first-class artifacts early on during requirements gathering and analysis phases. The tool can also aid project managers and Scrum teams in user story estimate and risk calculations as well as risk-driven planning and visualization of the proposed plans.

Review Table

Author Name	**YEAR**	**TECHNOLOGY USED**	**DESCRIPTION**
John Mylopoulos et al	1992	A process-oriented approach	To propose a comprehensive process oriented qualitative framework that integrates non-functional requirements into the process of software development.To illustrate the application of proposed methodology by taking examples of accuracy requirements in design phase and performance requirements in implementation phase for information systems. Evidence for the power of the framework is provided through the study of accuracy and performance requirements for information systems.
A. Eberlein and J. C. Leite	2002	Agile method	This position paper looks at numerous aspects of requirements engineering and argues about their suitability for agile approaches. The aim is to elicit lessons from requirements engineering that agile methods might consider, if quality is a major concern. On the other hand, some people appear to welcome agile methods as an excuse to throw overboard everything that requirement engineering has been teaching. This position paper looks at numerous aspects of requirements engineering and argues about their suitability for agile approaches. The aim is to elicit lessons from requirements engineering that agile methods might consider, if quality is a major concern.
F. Paetsch, A. Eberlein and F. Maurer	2003	Agile software	This article compares traditional requirements engineering approaches and agile software development. Their paper analyzes commonalities and differences of both approaches and determines possible ways how agile software development can benefit from requirements engineering methods.
N. S. Rosa et al	2004	Non-functional requirements	In this paper, they concentrate on defining how to reason and how to refine NFRs during the software development. Our approach is based on software architecture principles that guide the definition of the proposed refinement rules. In order to illustrate their approach, they adopt it to an appointment system.
L. Chung and J. C. S. do Prado Leite	2009	Non-functional requirements	Essentially a software system's utility is determined by both its functionality and its non-functional characteristics, such as usability, flexibility, performance, interoperability and security. Nonetheless, there has been a lop-sided emphasis in the functionality of the software, even though the functionality is not useful or usable without the necessary non-functional characteristics. In this chapter, they review the state of the

			art on the treatment of non-functional requirements (hereafter, NFRs), while providing some prospects for future directions.
S. Farhat et.al.	2009	Non-functional requirements	This work recognizes four NFR sorts and gives the philosophy for creating space particular NFR by utilizing procedures for changing over the necessities into outline ancient rarities per NFR sort. The commitment is four NFR sorts: Functionally Restrictive, Additive Restrictive, Policy Restrictive, and Architecture Restrictive and the software engineering process that gives particular refinements that outcome in one of a kind compositional and plan curios. By applying the same utilitarian prerequisite center to the distinctive NFR areas it upgrades the improvement process and advances software quality characteristics, for example, compensability, viability, resolvability, and traceability.
Taehoon Um et.al.	2011	Attributes Evaluation Method	They proposed a lightweight quality evaluation method for anlithe way to deal with reflect non-functional aspects. Their approach bolsters early distinguishing proof of non-functionality, and makes a difference members reliably continue focusing on quality qualities. Members get inputs for the following discharge by the evaluation consequences of demonstrating unsatisfied quality traits in each discharge. Besides, members can make anticipates quality change. Be that as it may, members have their claim criteria when they lead evaluation with prototypes.Accordingly, the proposed evaluation method could be subjective. Accordingly, it is expected to make a quantitative evaluation show, utilizing estimation measurements to enable members to have more trust in the outcomes.
Weam M. Farid et.al.	2012	NORMAP Methodology	This examination exhibits a lightweight building of NFRs for agile processes. The proposed Non-functional Requirements Modeling for Agile Processes (NORMAP) Strategy recognizes, connections, and models Agile Loose Cases (ALCs) with Agile Use Cases (AUCs) and Agile Choose Cases (ACCs). A lightweight adjusted adaptation of the NFR System was created including 25 critical NFRs. Further, a hazard driven agile requirements usage arrangement and a visual tree-like view were created. The procedure was approved through building up a Java-based modeling reproduction apparatus and two contextual investigations.
W. M. Farid and F. J. Mitropoulos	2012	NORMATIC	This research proposes NORMATIC, a Java-based simulation tool for modeling non-functional requirements for semi-automatic agile processes. NORMATIC is the semi-automatic tool that supports the more general Non-Functional Requirements Modeling for Agile Processes (NORMAP) Methodology. Early results show that the tool can potentially help agile software development teams in reasoning about and visually modeling NFRs as first-class artifacts early on during requirements gathering and analysis phases.

III. REFERENCES

[1]. J. Mylopoulos, L. Chung and B. Nixon, "Representing and using nonfunctional requirements: A process-oriented approach," *IEEE Transactions on Software Engineering,* vol. 18, no. 6, pp. 483-497, 1992.

[2]. A. Eberlein and J. C. Leite, "Agile requirements definition: A view from requirements engineering," in *Proceedings of the International Workshop on Time-Constrained Requirements Engineering (TCRE'02)*, 2002.

[3]. F. Paetsch, A. Eberlein and F. Maurer, " Requirements engineering and agile software development," in *Proceedings of twelfth IEEE international workshops on enabling technologies: Infrastructure for collaborative enterprises (WETICE)*, 2003.

[4]. N. S. Rosa, P. R. Cunha and G. R. Justo, "An approach for reasoning and refining non-functional requirements," *Journal of the Brazilian Computer Society,* vol. 10, no. 1, pp. 59-81, 2004.

[5]. L. Chung and J. C. S. do Prado Leite, "On non-functional requirements in software engineering," *Conceptual modeling: Foundations and applications,* pp. 363-379, 2009.

[6]. S. Farhat, G. Simco and F. J. Mitropoulos, "Refining and reasoning about nonfunctionalrequirements," in *Proceedings of the 47th Annual Southeast Regional Conference.ACM*, 2009.

[7]. T. Um, N. Kim, D. Lee and H. P. In, "A Quality Attributes Evaluation Method for an Agile Approach," in *2011 First ACIS/JNU International Conference on Computers, Networks, Systems, and Industrial Engineering*, 2011.

[8]. W. M. Farid and F. J. Mitropoulos, "NORMATIC: A visual tool for modeling non-functional requirements in agile processes," in *Proceedings of the IEEE SoutheastCon 2012*, 2012.