International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Associated Requirements Classification Model and Standardization for Enhancing Quality of Software

[*1]Mr. Thakur Ritesh Bankat Singh, [2]Dr. S. V. A.V. Prasad, [3]Dr. Malla Reddy Jogannagari

Submitted:10/05/2024 **Revised:** 24/06/2024 **Accepted:** 01/07/2024

Abstract: All software's, systems, goods, services, and businesses are built upon requirements. All parties involved in the requirement classification should receive information that is essential, concise, verifiable, traceable, and comprehensive from a well-crafted demand. System functions and objectives are defined using a variety of requirements, including design, quality, certification, non-functional and functional, which are derived from the domain of interest and the system under design. A great quality software project always begins with gathering requirements. The primary responsibility for collecting requirements is Requirement Engineering (RE). Obtaining relevant data is essential for developing high-quality software. With the help of big data and machine learning, software engineering has recently become data centric. As time goes on as technology, social media, and other sources continue to advance, more and more data is collected from a variety of sources. When gathering the necessary components to manufacture a high-quality product, there are numerous aspects to consider. The software development life cycle includes the requirement engineering step, which is crucial. The goal of requirement engineering is to facilitate communication between developers and clients for accurate classification. The quality of the software product and its ability to meet user requirements are both impacted by the extent to which requirements are comprehensive and consistent. Taking into account the needs of the product from a variety of perspectives, roles, and responsibilities is a challenging aspect of requirement classification. If requirement classification is done correctly, the software product quality will be affected. In this study, requirement classification is considered and its processes contribute to the creation of high-quality software. A lack of process consistency throughout the primary development phases, including requirements analysis, has a negative impact on the development of agent-based systems. Because of this problem, agent technology investors have a far more difficult time understanding and evaluating the intricacy of these system requirements. This research presents an Associated Requirement Classification Model (ARCM) for standardization in the process of implementing high quality software. The proposed model when contrasted with the traditional requirement classification models performs better in classification accuracy.

Keywords: *Software Development, Software Lifecycle, Requirement Gathering, Requirement Engineering, Standardization, Quality of Software.*

1. Introduction

A software engineer's job is to create high-quality software that meets all of the client's needs and then keep it running smoothly after deployment by fixing bugs and updating code [1]. Utilizing engineering standards allows for the development of cost-effective software that is both dependable and capable of running on actual hardware. The utilization of the systematic, disciplined as well as quantifiable methods to the creation, maintenance [2], and operation of software to the system; essentially, applying the principles of engineering methods to software is the definition given by the IEEE standard for software engineering [3]. Boehm characterizes software engineering as the practical use of scientific concepts in the design as well as construction of software applications and the associated documenting needed for creating, operate, and maintain them. Software engineering is all about the theories, methods, as well as tools that are necessary to create software products efficiently and affordably [4].

Software engineers employ a wide variety of techniques, processes, tools, and standards while creating new software [5]. An alternative definition of software engineering as a layered approach that takes into account an organization's dedication to product quality, a method that incorporates a number of activities necessary for software

Research Scholar, Dept. of Computer Science & Engineering, Lingaya's Vidyapeeth, Faridabad, Haryana, India , ritraj.t76@gmail.com
Professor , Dept. of Computer Science & Engineering, Lingaya's Vidyapeeth, Faridabad, Haryana, India, svavprasad@lingayasvidyapeeth.edu.in
Professor , Dept. of Computer Science & Engineering, Mahaveer Institute of Science and Technology,Hyderabad, India , jmrsdpt06@gmail.com

development such as requirement elicitation, design, code implementation, testing, maintenance, and the use of automated tools to ensure timely completion of the process. Both procedures are designed to aid in the creation of software. The first section distinguishes itself by outlining the development process in detail. And the second grouping is concerned with how to better put high-quality software into action [6]. The quality of the final software product is directly proportional to the degree to which the software development process adheres to established standards for quality [7]. Adopting the right process for building software products is, thus, beneficial.

Various software development procedures can be employed to create software, depending on a multitude of factors such as the complexity of the program, the resources that are available, and the requirements that clients or end users offer [8]. An integral part of every software development process is the life cycle of software development. One graphical and emotive way to represent product development is via a software procedural model [9]. Every stage of creating a software programming product may be seen in a software product model. Activities in various software product life cycle models are mapped out in relation to the software development standards model, which in turn develops software products [10].

When it comes to creating software, the Software Development Life Cycle (SDLC) is paramount. The SDLC is a useful tool for estimating how long it will take to create software. In addition, it's useful for tracking the several stages that software goes through as it's being built or used. There are numerous steps in the software development life cycle, and each one is defined and responsible for a distinct set of activities [11]. There is a prescribed order for completing each step of the process because each procedure builds upon the one before it. Figure 1 represents the software development life cycle, which consists of two distinct phases: product engineering and process management.

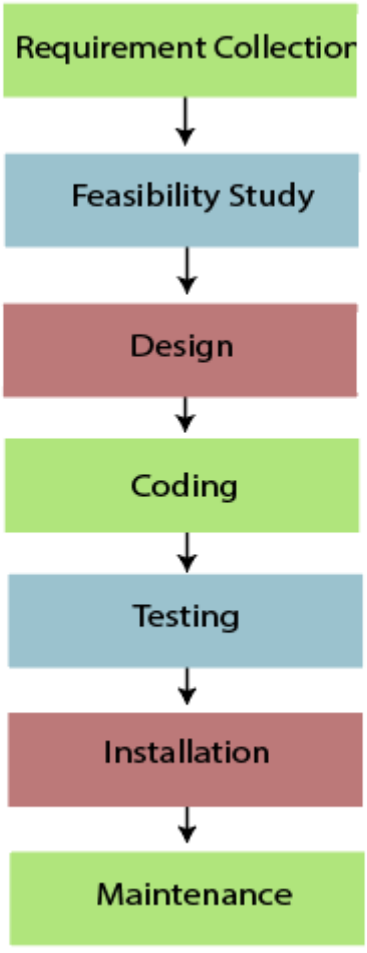

Fig 1: Software Development Life Cycle

System lifecycle models were developed to emphasize the significance of adhering to a structured approach to improved system construction, as stated. Rapid creation of an application prototype, and V-shaped were among the waterfall designs that were suggested [12]. As many companies have evolved, so has the urge to automate tasks that were formerly done by hand. Standardized, structural procedures that substantially simplify the system are essential if the industry wants to ease the shift from human to automated processes [13]. SDLC used a number of models for software development. It is common practice to incorporate software development-related data into SDLC models. A systematic approach to software development that guarantees optimal performance and timely delivery is essential, and SDLC models play a key role in this. When project managers employ the right SDLC, they get overall control over the software development strategy. When choosing an SDLC model, it is important to weigh the benefits and drawbacks of each option. Software engineering is a systematic and quantitative process for developing, deploying, and maintaining software. There have been many attempts to improve software or systems over the last many decades, with varying degrees of success [14]. The software requirement types is shown in Figure 2.

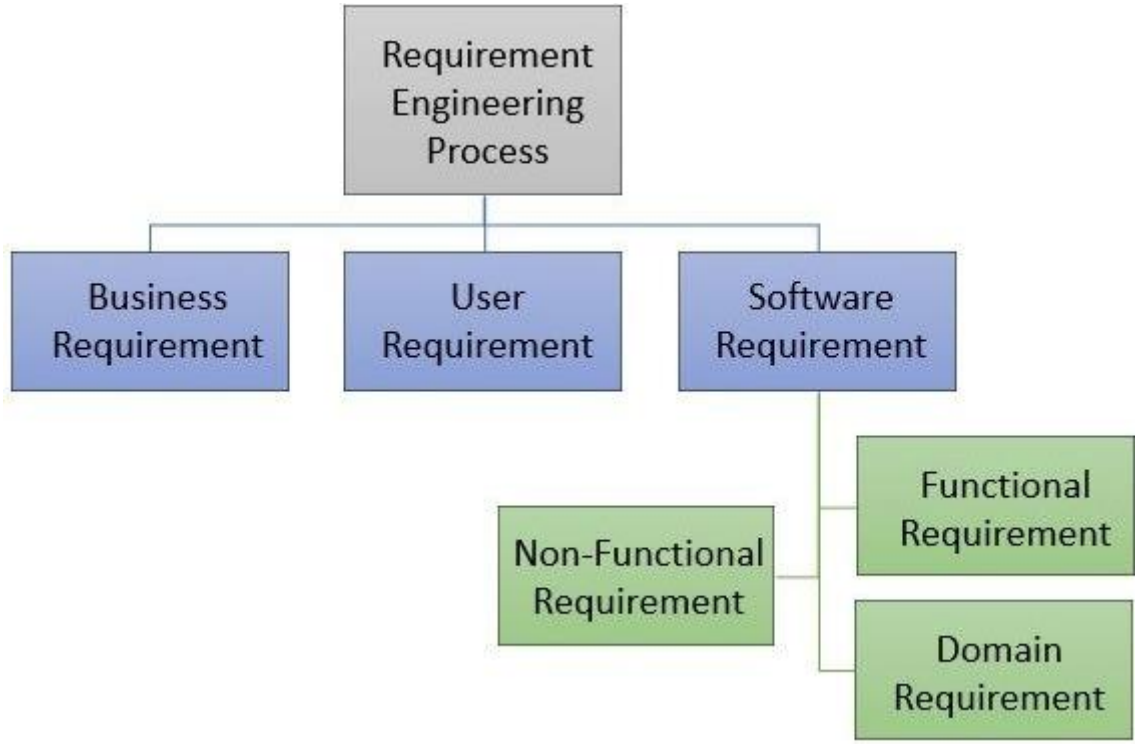

Fig 2: Software Requirement Types

The initial set of 55 quality criteria, or variables, identified by had a substantial impact on quality. To make things easier, McCall narrowed the list of qualities down to eleven: accuracy, usability, reliability, efficiency, flexibility, interface facility, adaptability, as well as transferability [15]. Boehm defined a second set of quality factors, which included nineteen characteristics including: simplicity, readability, efficiency, dependability, adaptability, resilience, correctness, maintainability, flexibility, interoperability, clarity, validity, economy, and generalizability [16]. Users are solely concerned with the software's outward appearance, while developers focus on the software's internal structure to get these outward appearances [17]. Features that are external to the product, such as trust, dependability, practicality, and precision are considered. Functional Requirement (FRs) is one that specifies the outcome of the system's behavior and how that function will be implemented. Functional requirements are defined as requests that outline the capabilities that an entire system or its parts must have [18]. A functional requirement is a description of a feature that the system's users will be able to access; it partially explains the system's behavior in response to a stimulus. In a perfect world, functional requirements wouldn't have anything to do with design or implementation; in

other words, they wouldn't bring up any technical issues [19].

Unlike functional requirements, Non-Functional Requirements (NFRs) specify the attributes of the produced system and have a greater impact on its architecture. It offers an alternative interpretation, stating that: a requirement that is not functional is a collection of constraints placed on the system that need to be created, which determine its attractiveness, usefulness, speed, and reliability, among other things [20]. NFRs are a set of required overall characteristics of the system, such as flexibility, reliability, efficiency, human design, testability, understandability, along with modifiability. NFRs represents the non-behavioral features of a system, gathering the properties as well as limitations under which a system must operate. It is possible to classify NFRs into subgroup [21]. Product requirements, organizational and process-related requirements, and external needs are some ways to categorize non-functional requirements. In software engineering, the distinction between functional and non-functional needs should center on the how and what of system performance or resource offerings [22].

Project analysts or requirement engineers work with the demand to try to determine the desired system requirements during the software requirement analysis phase. Then, using the requirements as a guide, they create the software requirement document and distribute it to everyone who needs to see it. The requirements document is then thoroughly reviewed by requirements engineers or business analysts. So, depending on the system's intended purpose, they classify them as either FR or NFRs [23]. FRs are the requirements that a product must meet in order to be considered complete. In addition, FR is the software details specified by the stakeholders, the system's services, and the system's required limitations. Features including response time, performance, security, and usability are examples of NFR, which is sometimes called software quality characteristics [24]. Because other phases of the software life cycle, such design and coding, are based on the software requirements classification, the project's success is directly related to the accuracy of the FR and NFR classifications [25]. It could be difficult to manually distinguish between FR and NFR since they are both natural language texts included in the same requirement document. Ignoring NFR leads to project failure, loss of system integrity, or cost increase. Without FR, the built software system fails. This research presents an Associated Requirement Classification Model for standardization in the process of implementing high quality software.

2. Literature Survey

Software analytics tools have been developed in the last ten years, made possible by advances in data analytics. These tools provide real-time visualization of many elements of software development as well as usage. Businesses engaging in agile software development may find these products especially appealing. Unfortunately, there is no way to combine or link the data offered by the current technologies to better quality objectives. Simultaneously, the software engineering industry has focused on evaluating and enhancing software quality, leading to several suggestions for models and standards in this area. The gap might be filled by connecting software analytics tools with such quality models, which would lead to greater quality targets. S. Martínez-Fernández et al. [1] investigated whether practitioners plan to use the information on process or product quality that is provided by software analytics tools that incorporate quality models in a way that is intelligible, dependable, practical, and relevant. Included in this case study are four businesses who, for over a year, used this kind of tool to evaluate and enhance software quality across several projects.

Commercial open-source software (COSS) firms have recently seen a spike in their number, which indicates their increasing importance in the software market. The success of COSS companies, which are knowledge-based, is highly dependent on the interaction of intangible resources including software quality, human resources, relational capital, and structural capital. Garomssa [2] surveyed 200 software development specialists and professionals from 60 different international COSS organizations using a questionnaire-type approach to explore hypotheses about these linkages. The research confirmed that intellectual capital affects COSS company success in two distinct but complementary ways: directly and indirectly. The success of the COSS company is influenced by relational capital, which is one component of intellectual capital. However, under a sequential mediation model, software quality mediates the relationship between relational as well as structural

capital and human capital, which in turn influences the profitability of COSS companies. Since software quality is the single most important factor determining COSS companies' success, it follows that COSS companies may have to make software quality a top priority.

The demand for reliable and secure software solutions is growing as software becomes more integrated into every aspect of life and every type of organization. The goal of software development techniques is to enhance software quality through the incorporation of practices that foster software quality. Nowadays, the majority of software is made available for use via the Internet, hence security for software is a crucial aspect of software quality. The majority of studies addressing the topic of safe and high-quality software development have neglected individual developers in favor of teams. In this study, Moyo et al. [3] offered an agile secure-software method of development to address this gap. This methodology is designed to help solitary developers create software that is both secure and of high quality. Agile safely developed software procedures are the result of our integration of quality practices with portable security practices. The author used security measures taken from pre existing lightweight techniques and quality practices taken from a standalone software development framework that the author built in earlier research.

In order to evaluate and identify outliers that impact the quality of a computer program, Software quality assurance approaches are extensively utilized during software development. Over the past few years, numerous software quality control (SQC) methods have been suggested for ensuring software systems' privacy. Nevertheless, study has been conducted from several angles, leading to an expanding corpus of information dispersed across various fields. Guamán et al. [4] conducted a thorough mapping study to fill this knowledge gap and give researchers and practitioners a bird's-eye perspective of the most cutting-edge methods for privacy-focused software quality control in information systems. According to the findings, there has been an increase in research in this area. Since the assessment criteria for 37% of compliance-focused procedures are based on the European Data Protection Regulation, this legal framework appears to be significantly impacting this expansion. Different types of approaches have different levels of maturity: While combination approaches have proven effective in real-world circumstances, formal verification techniques are still in their early stages of development.

Due to the emphasis on rapid delivery and minimum documentation in the agile software development (ASD), quality requirements (QRs) are frequently left undocumented or underspecified. There is a lack of guidelines to assist with the QR documenting work. In an effort to bolster QR documentation in ASD, Behutiye et al. [5] established a set of Agile QR-Doc QR documents guidelines. In order to construct the Agile QR-Doc, the author used a DSRM, or design science research methodology. In order to verify the accuracy of the Agile QR-Doc, the author polled ten software professionals from two ASD firms and had open discussions with them. The rules were assessed by the practitioners for their practicality, applicability, clarity, and breadth of coverage in relation to QR documentation and its effect on software development agility. Agile QR-Doc provides a list of twelve suggestions, divided into two groups. The first grouping presents three suggestions with an emphasis on getting the word out regarding the value of QRs, their documentation, and the difficulties associated with them. Nine suggestions introducing artifacts, techniques, and critical components for QR documentation are presented in the second category. The rules to enable QR documenting in ASD are relevant, understandable, and valuable, as revealed by the validation.

Since the dawn of the digital age, software systems have undergone tremendous transformation and are now fundamental to human civilization. The massive amounts of sensitive data generated by software systems' widespread use necessitate their protection. Ensuring the safety of these computer systems is just as critical as making sure they meet the needs or functional demands of the users. But new studies reveal that as software development approaches go beyond demand architecture to their last stages, many of these approaches fail to incorporate software security safeguards. There is now a critical requirement to incorporate software security into the SDLC at every level. Many approaches, concepts, and methodologies have been proposed and implemented to address software security; nevertheless, just a handful offer substantial proof for developing secure software applications. Khan et al. [6] primarily aimed to examine security measures within the framework of

safe software development (SSD) as it pertains to systematic mapping (SMS).

As a first step in requirements engineering, software requirement validation checks that the acquirer's goals and the target system's capabilities are a good fit. Its primary objective is to identify and rectify mistakes that predominate within the defined parameters. Some software may fail despite the abundance of requirements validation methodologies due to inadequate or nonexistent techniques, as well as the unreliability of the requirements themselves. A comprehensive literature analysis on requirements validation is carried out in this study by Atoum et al. [7]. This research looks at the most popular validation methods, details the qualities of high-quality requirements, and finds major obstacles to validation. Trends in requirements validation techniques, including the strengths and weaknesses of their subtechniques, categories of requirements quality characteristics, and the tools and datasets used in these techniques were analyzed in depth from 66 primary studies that were deemed relevant to the review. The author classified validation methods as either formal models, knowledge-oriented, test-oriented, prototyping, or inspection. A total of twenty-seven tools, nineteen different validation methods, several new features for requirements validation, and a number of difficulties were detailed in the study.

Activities involving knowledge and collaboration make up requirements elicitation. Although many methods for gathering requirements and eliciting implicit understanding from stakeholders have been suggested in requirements engineering literature, very few have actually implemented such tactics. One of the most obvious issues with requirements elicitation is the challenge of intentionally capitalizing on crucial stakeholders' tacit knowledge. To better understand and access the implicit information that has been developed throughout requirements elicitation, H. Al-Alshaikh et al. [8] presented a strategy for doing just that. The model is built upon the principles of reasoning knowledge elicitation within the context of the requirements elicitation process, which has been adopted and expanded upon. In addition, a representation code for expressing implicit understanding in this setting is provided in this study. Lastly, in order to assess the model's practicability, a survey was administered to domain experts to collect their feedback on the suggested model's capacity to facilitate the elicitation of tacit knowledge. The suggested model was also tested in a controlled environment.

Companies use Global Software Development (GSD) to create affordable, high-quality software. For a GSD project to be a success, Requirement Change Management (RCM) is crucial. M. A. Akbar et al. [9] aimed to validate the identified limitations of the RCM process through the use of a questionnaire survey in real-world practices. It does this by adopting a systematic literature review (SLR). The combination of SLR with empirical research yielded a list of twenty-five obstacles. To further understand the RCM problems in the context of both kinds of GSD enterprises, the author grouped the highlighted challenges into client organizations and vendor organizations. Additionally, the identified difficulties were grouped into three primary types of organization size: small, medium, and large. This categorization helps to emphasize the importance of every obstacle for each level of organization.

Software requirements greening refers to the practice of incorporating sustainability ideas into the requirements engineering stage in the development life cycle. This integration has the potential to significantly impact the software architecture used by state-of-the-art IT systems. Priorities in software design can shift to improve the use of resources and energy, flexibility, maintainability, adaptability, and sustainability when requirements engineering incorporates sustainability principles. It is necessary to conduct more research into the connection between the development of software and the applicable green sustainability principles during requirements engineering, in contrast to other environmentally friendly methods that take sustainable development into account. One step involves mapping NFRs to sustainability dimensions, and the other involves mapping sustainability dimensions to two groups of green IT features that are defined in this work by Subahi et al. [10]. This new mechanism maps software NFRs to specified aspects of green software sustainability.

3. Proposed Method

In order to ensure that the system meets all stakeholders' needs throughout its lifetime, requirements engineering must collect requirements from those stakeholders, document them appropriately, validate and verify them, and manage

them throughout the development process. The method known as requirements engineering relies on requirements elicitation, an essential activity that must be carried out in order to find the product's functionality by converting stakeholder desires and needs into software specifications. An essential part of requirements engineering is gathering needs for the future system. Knowledge acquired during requirements engineering regarding the system context that needs to be developed, which includes the sources of requirements to be evaluated and queried, is the basis for gathering requirements. The requirements are typically categorized into two primary types: functional requirements as well as non-functional requirements. This helps in making a thorough documentation of the requirements.

One tool that may be used to help with security requirements analysis is security requirements elicitation, which is the process of gathering the SR. Conventional statistical methods are not well suited to this analysis because the requirements are expressed in normal language. Most software engineers also don't know much about security, which makes the problem worse. Security needs categorization is a part of analysis as a process. This involves placing each security demand into a specific model for the software's security services. Even though security is typically not given much attention in most studies, it is one of the functional and non-functional needs that has been studied. Researchers are discouraged from focusing on this crucial area because even models with more granularity do not come with a comparable dataset. On top of that, maintainability is seen as more of a non-functional criterion that has little to do with security. Software requirement categorization becomes a non-trivial effort in light of all these possibilities. Software requirement classification and other tasks, such as dependability prediction, are made easier and faster by a number of machine learning as well as natural language processing algorithms. However, suitable datasets for training, validating, and testing machine learning models are required to extract and categorize software requirements. Replicable study of security requirements categorization is hindered by the skewed datasets that are currently available for software requirements, which focus on non-functional or functional needs rather than security. Also, the classification results are affected by the datasets' imbalance, which is highly noticeable. The feature selection and classification is shown in Figure 3.

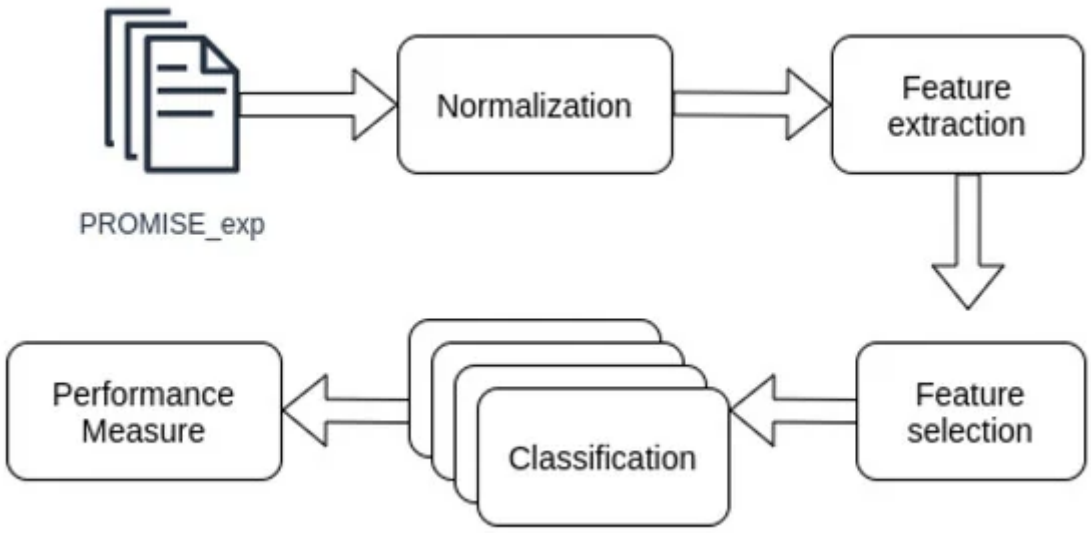

Fig 3: Feature Selection and Classification Process

Data mining and natural language processing are two examples of the artificial intelligence techniques that have been developed to manage the spread of data and extract reliable information from it. To enhance data mining performance and produce clean, understandable data, feature selection seeks to build models that are simpler and easier to understand. Each column in the dataset may include an option that describes the data; these options are called features. The choices defining text can be used to classify it. For a classification process to be successful, it is ideal to consider the features that

best characterize the data. Results in text classification tasks should therefore be as close to reality as possible, hence it is crucial to think about the features that describe the data the best. By excluding features that are either duplicated across the dataset or do not contribute significantly to its overall meaning, feature selection allows for the management of options that are more reflective of the data. The goal of feature selection strategies is to decrease dimensionality and speed up learning in order to improve prediction performance.

Machine learning algorithms frequently use feature selection approaches in data pre-processing to build a feature subset with high-quality features that contribute to computation from the data feature space and improve performance. Gain ratio (GR) and correlation-based feature selection (CFS) were also employed in the study. Using a multivariate filter technique, CFS selects groups of features that are unrelated to each other yet have a strong correlation with the class. When using correlation-based feature selection, a heuristic evaluation function is employed to rank the feature subsets. The approach disregards features with low correlation while defining more significant features as highly correlated during the training and testing process of the prediction model. In addition, the prediction model gets rid of all the extra choices. The GR method determines the information gain for each feature. Therefore, features are chosen based on their performance and gain ratio, with the criteria of performing at least as well as the average information gain. When compared to the information gain metric, GR performs better in terms of classifier complexity and accuracy. The proposed model workflow is shown in Figure 4.

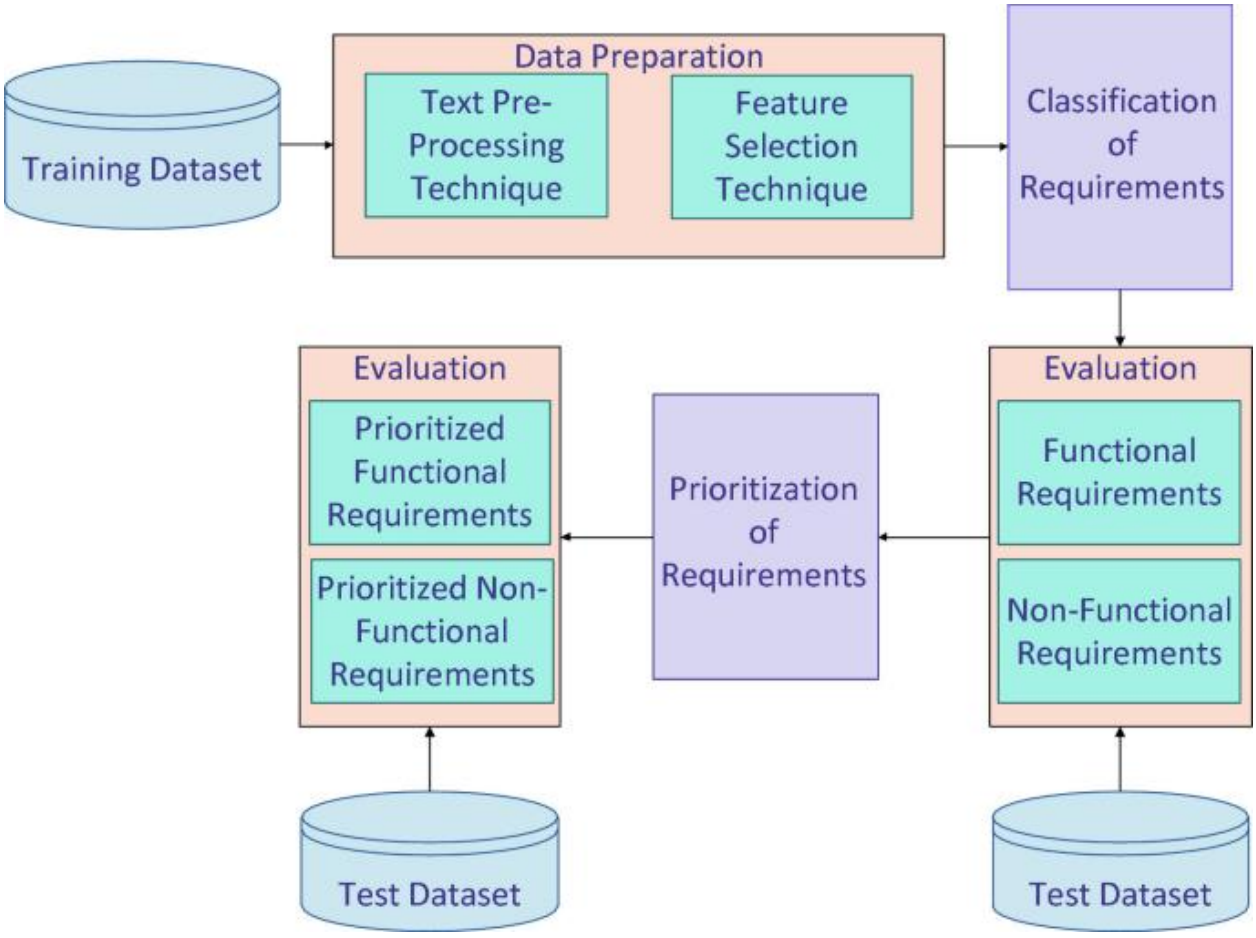

Fig 4: Proposed Model Workflow

The most important part of software engineering, it turned out, was requirements analysis. There is a wide variety of approaches, techniques, and tools that have been created, suggested, and used thus far to aid in requirements elicitation, definition, and validation for maintain standardization. Functionality representation and organization are at the heart of most requirements modelling methodologies. The development of software, however, is about more than just functionality. Unconsidered criteria can lead to project cancellations, unprofitable products, dissatisfied

users, and schedule and budget overruns. The literature often refers to a dimension of requirements known as non-functional requirements or quality requirements as the source of those problems. A generic taxonomy is suggested that places an emphasis on quality standards and how they are realized are performed for software standardization. The fact that a comprehensive and final description of quality requirements is now impossible is taken into consideration by this categorization approach. It guarantees that the characterisation framework's extensibility won't be hampered from the start, therefore it won't have the same effect on quality needs as earlier characterization efforts. Also, with the right data system in place, the suggested categorization and standardization scheme can help with elicitation, communication, traceability, and control tasks, all of which lead to better documented requirements. This research presents an Associated Requirement Classification Model (ARCM) for standardization in the process of implementing high quality software.

The software requirement dataset is considered from public dataset service provider available at the link https://www.kaggle.com/datasets/iamvaibhav100/software-requirements-dataset. Data pre-processing is the process of preparing data for analysis by cleaning and altering it. Accurate, consistent, and analyzable data is what data preparation is all about. As a result, data mining becomes more effective and efficient. The data pre-processing is performed as

In order for a software system to fulfil its customers' demands, it must first be identified and defined through requirements gathering. It entails consulting with relevant parties, gathering information about user requirements, and outlining precise requirements for developers to follow when creating the system. The requirement gathering is performed as

$$PrePro[L] = \prod_{i=1}^{L} \square\, getattr(i) + getmaxrange(i, i+1) + getminrange(i, i+1) + \frac{\omega(attr(i))}{L} + \frac{\delta(attr(i))}{L}$$

$$Proc[L] = \sum_{i=1}^{L} \square\, getattr\big(PrePro(i)\big) - \omega(i) - \delta(i)$$

Here ω is the model that considers null values in the dataset considered and δ is the model that considered unwanted symbols.

In a correlation analysis, the correlation coefficient is the metric that specifically measures the strength of the linear link between two variables. When writing up a correlation, the 'Corr' parameter stands for the coefficient. This formula tells us how well the relationship between two variables can be fit to an imaginary line formed across the data by comparing the distance of each datapoint from the variable mean. It is applicable when dealing with two features in the dataset iteratively. The idea that correlations only consider linear relationships is conveyed in this way. The correlation calculation is performed as

$$Corr[L] = \prod_{i=1}^{L} \square\, \frac{\sum \square\, (attr\big(Proc(i)\big) - \underline{Proc(i)}) * (attr\big(Proc(i+1)\big) - \underline{Proc(i+1)})}{\sqrt{\sum \square\, (attr\big(Proc(i)\big) - \underline{Proc(i)})^2 \sum \square\, (attr\big(Proc(i+1)\big) - \underline{Proc(i+1)})^2}}$$

By comparing feature attributes with a smaller set of attribute ranges, gain ratio can normalize the data using the entropy value of the variable, eliminating the bias associated with multi-variable data and variables with numerous ranges. The gain ratio is calculated as

$$GR[L] = \sum_{i=1}^{L} \square\, \frac{GR(i)}{max\ (Corr(i, i+1))}$$

When building a requirement classification model, feature selection is necessary to isolate the most important, consistent, and non-redundant features. As both the quantity and diversity of datasets increase, it is crucial to systematically decrease their sizes. By eliminating superfluous, unimportant, or distracting elements, feature selection narrows down the original set of features to just the most useful ones. The feature selection process is performed as

$$Fsel[L] = \sum_{i=1}^{L} \frac{getattr(i)}{L} + \big(Corr(i,i+1)\big) + max\,(GR(i,i+1)) \{ Fsel \leftarrow i \;\; if\; corr(i) < Th\; and\; GR(i) > Gth\; continue \quad Otherwise$$

FRs and NFRs are the two primary ways in which software needs are usually categorized. A variety of quality criteria, including security, availability, and usability, are subsets of non-functional requirements. The software requirements classification is performed as

$$Rclass[L] = \sum_{i=1}^{L} getmax(Fsel(i,i+1)) + \big(Corr(i,i+1)\big) + \frac{max\,(attr(i,i+1))}{L} - min\,(Fsel(i))$$

4. Results

Every person and every product is believed to place a major emphasis on quality. Many different types of people have offered their own definitions. In a nutshell, there are two types of product quality: one is for built hardware and the other is for developed software. Even though the exact definition of quality is different for the two products, the statement if the product is deemed acceptable by the clients therefore the standard of excellence of the item in question is high still applies. Customers, clients, end users, and development teams all have a fundamental impact on software quality. Therefore, it is critical to focus more on the process with the necessary effort in order to build high-quality software. Customers and product developers are also affected by the process quality.

Software engineering projects can be categorized using a model that is built on top of big data technologies. Gathering data, cleaning it up, selecting features, training the model, evaluating it, making adjustments, and finally applying it are the fundamental steps of this model. Information about the project's scope, programming language, development methodologies, and technical details must be gathered and organized during the data gathering stage of a software engineering project. The data pre-processing phase involves cleaning and processing the obtained data, which includes tasks such as de-duplication, missing value processing, discretization, and standardization, among others. If users want the model to be better at generalizing and have lower dimensionality, they need to pick features that are meaningful and useful during the feature selection stage.

The PROMISE repository's is considered. The dataset is used from the link http://promise.site.uottawa.ca/SERepository/datasets/jm1.arff. The proposed model is implemented in python and executed in Google Colab.This information is labelled according to whether the software need is functional or non-functional. The data was subsequently pre-processed by doing things like changing the case of all characters to lowercase, removing symbols and non-alphanumeric ones, and removing commonly used words like the and a as well as others with lengths of two or less because they don't matter much for classification. After that, each sentence was converted to a word using the tokenization process. This research presents an Associated Requirement Classification Model (ARCM) for standardization in the process of implementing high quality software. The proposed model is compared with the traditional Novel Lightweight Solo Software Development Methodology With Optimum Security Practices (NLSSD-OSP) and BERT-Based Approach for Greening Software Requirements Engineering Through Non-Functional Requirements (BERT-GSRE) model. The results represent that the proposed model performance in classification is high than the traditional models.

Improving the accuracy and effectiveness of data mining is the purpose of data pre-processing, which entails cleaning and modifying data to make it acceptable for analysis. The data must be correct, consistent, and fit for analysis. The Pre-Processing Accuracy Levels of the proposed and existing models are shown in Table 1 and Figure 5.

Table 1: Pre-Processing Accuracy Levels

Records Considered	Models Considered		
	ARCM Model	NLSSD-OSP Model	BERT-GSRE Model
10000	97.6	93.1	94.5
20000	97.8	93.3	94.6
30000	98.0	93.6	94.8
40000	98.3	93.9	95.0
50000	98.5	94.0	95.2
60000	98.7	94.2	95.4

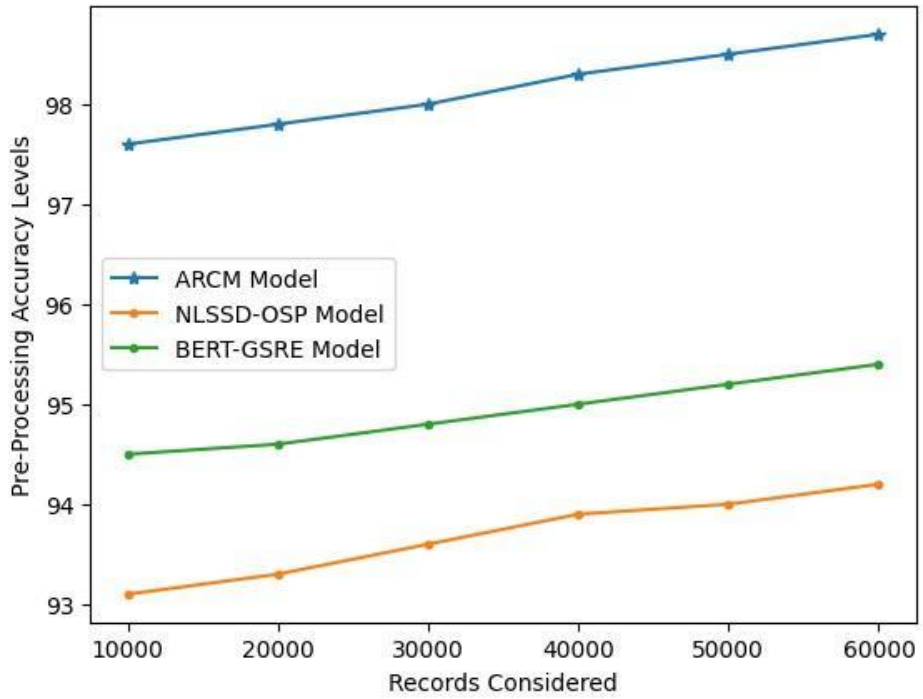

Fig 5: Pre-Processing Accuracy Levels

In order for a software system to ful fill its customers' demands, it must first be identified and defined through requirements gathering. It entails consulting with relevant parties, gathering information about user requirements, and outlining precise requirements for developers to follow when creating the system. To properly scope out a project, users must first identify its requirements. Software requirements play a crucial role in defining the final product's features, development timeline, and budget. As a result of poorly stated project goals, scope creep can occur. The Requirement Gathering Time Levels of the existing and proposed models are indicated in Table 2 and Figure 6.

Table 2: Requirement Gathering Time Levels

Records Considered	Models Considered		
	ARCM Model	NLSSD-OSP Model	BERT-GSRE Model
10000	17.0	25.0	28.0
20000	17.3	25.2	28.2

30000	17.5	25.4	28.4
40000	17.6	25.6	28.5
50000	17.8	25.8	28.8
60000	18	26	29

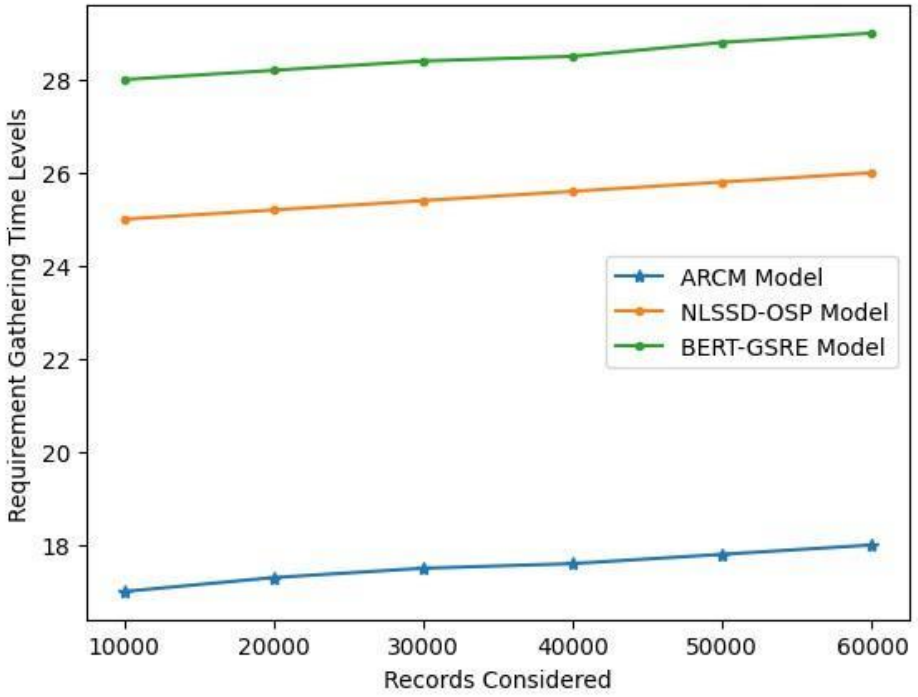

Fig 6: Requirement Gathering Time Levels

For interval-type variables, the correlation ratio is just the square root of the sum of squares, divided by the entire sum of squares. One way to measure the strength and direction of linear relationships between two variables is via a correlation coefficient. The Table 3 and Figure 7 shows the Correlation Calculation Accuracy Levels of the proposed and existing models.

Table 3: Correlation Calculation Accuracy Levels

Records Considered	Models Considered		
	ARCM Model	NLSSD-OSP Model	BERT-GSRE Model
10000	97.6	93.7	94.1
20000	97.8	93.9	94.3
30000	98.0	94.1	94.6
40000	98.1	94.3	94.8
50000	98.3	94.6	95.0
60000	98.5	94.8	95.2

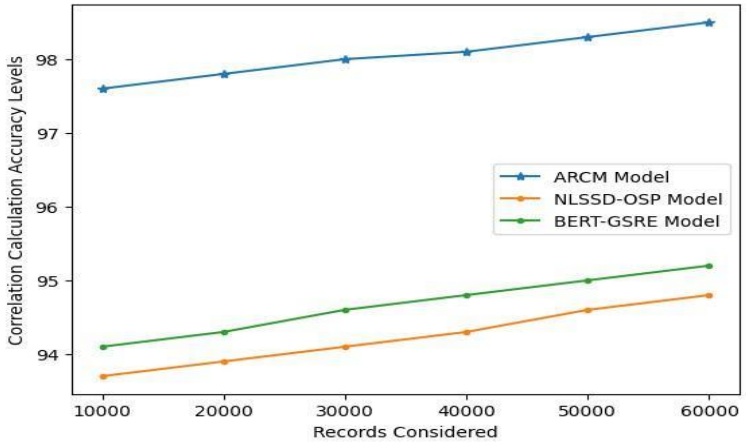

Fig 7: Correlation Calculation Accuracy Levels

A tweak to the information gain that decreases its bias is the gain ratio. The number and size of branches are considered by gain ratio while picking an attribute. By factoring in the intrinsic information of a split, it fixes the information gain. When working with datasets that include attributes with varying quantities of unique values, Gain Ratio increases. Here, the gain ratio considers the attribute's intrinsic information, which aids in preventing biases towards characteristics having many distinct values. The Gain Ratio Calculation Time Levels of the proposed and existing models are indicated in Table 4 and Figure 8.

Table 4: Gain Ratio Calculation Time Levels

Records Considered	Models Considered		
	ARCM Model	NLSSD-OSP Model	BERT-GSRE Model
10000	11.6	18.0	20.7
20000	11.8	18.2	21.0
30000	12.0	18.4	21.2
40000	12.2	18.6	21.4
50000	12.4	18.7	21.6
60000	12.6	18.9	21.8

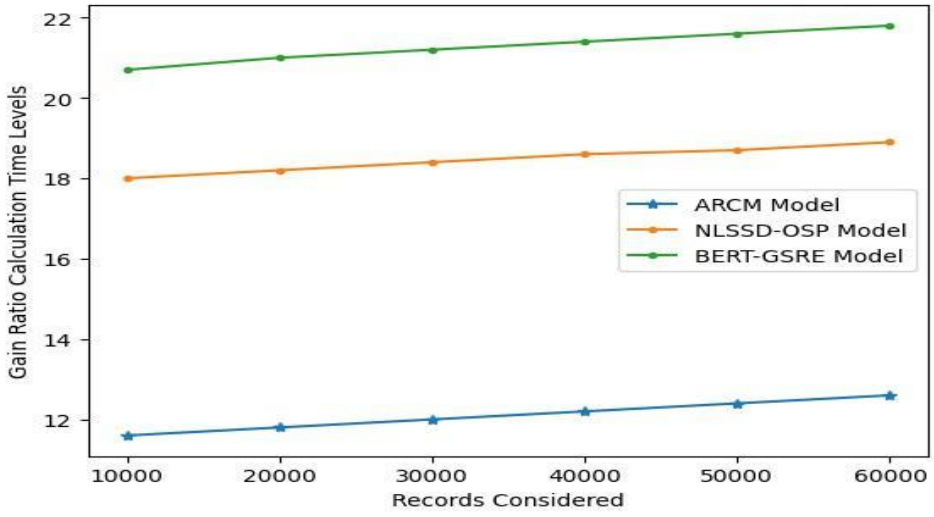

Fig 8: Gain Ratio Calculation Time Levels

When building a model, feature selection is necessary to isolate the most important, consistent, and non-redundant features. As both the quantity and diversity of datasets increase, it is crucial to systematically decrease their sizes. Improving a predictive model's effectiveness while decreasing the computational cost of modelling is the primary objective of feature selection. The Table 5 and Figure 9 represents the Feature Selection Accuracy Levels of the existing and proposed models.

Table 5: Feature Selection Accuracy Levels

Records Considered	Models Considered		
	ARCM Model	NLSSD-OSP Model	BERT-GSRE Model
10000	97.6	94.5	93.7
20000	97.8	94.7	93.9
30000	98.0	94.8	94.1
40000	98.1	95.0	94.3
50000	98.3	95.2	94.5
60000	98.6	95.4	94.7

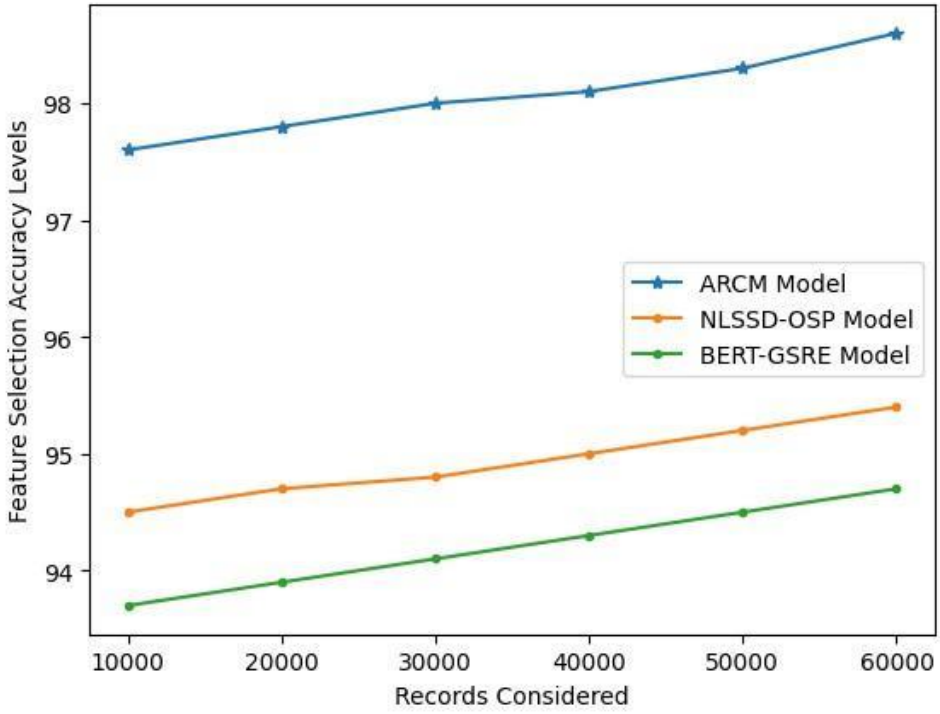

Fig 9: Feature Selection Accuracy Levels

FRs and NFRs are the two primary ways in which needs are usually categorized. A variety of quality criteria, including security, availability, and usability, are subsets of non-functional requirements. There are two main kinds of requirements: FRs, which describe the actual services, behaviors, or functions provided by a system, and NFRs, which pertain to the qualities like quality, usability, security, privacy, etc. or limitations of the application or software development process as a whole. The Requirements Classification Accuracy Levels of the proposed and existing models are shown in Table 6 and Figure 10.

Table 6: Requirements Classification Accuracy Levels

Records Considered	Models Considered		
	ARCM Model	NLSSD-OSP Model	BERT-GSRE Model
10000	97.5	93.3	94.3
20000	97.8	93.5	94.5
30000	98.1	93.7	94.7
40000	98.3	93.9	94.9
50000	98.5	94.1	95.0
60000	98.7	94.4	95.2

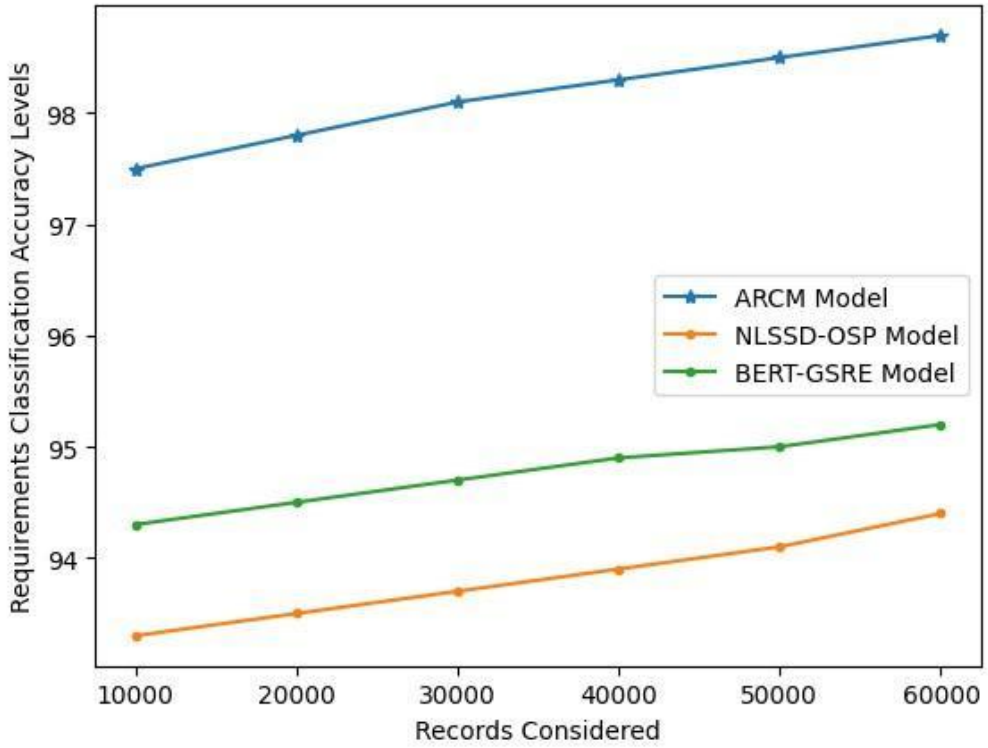

Fig 10: Requirements Classification Accuracy Levels

5. Conclusion

When it comes to creating top-notch software, the combination of known as RE as well as Quality Assurance (QA) has become an essential component of contemporary software development. Through this extensive investigation, the significance of traceability as a connecting factor between stakeholder demands and the quality attributes of the end product has been highlighted. At every stage of the development process, agile techniques place an emphasis on input from the client. Satisfaction with customers and software quality are both improved by making sure the program meets the needs and expectations of the end users through this customer-centric approach. There are a lot of good effects of SQA. Strict SQA procedures directly lead to more reliable products, lower failure rates, and happier customers. The flexibility of SQA to meet the ever-changing requirements of contemporary software development is demonstrated by the time-to-market acceleration that results from continuous testing and automation. Organizations face challenges such as high implementation costs, lengthy testing procedures, and the necessity to strike a balance between technology and human intuition. It is important to have a plan for implementation because of things like resistance to alteration and the difficulty of maintaining automation test suites. A new age of intelligent testing is dawning with the incorporation of AI and ML into SQA processes,

made possible by ever-improving technologies. One way to make sure to get a good software product is to think about security requirements early on in the SDLC. Accurately identifying and categorizing the protection needs for each software throughout development is crucial for knowing the amount of protection that is necessary. Successful software projects rely on accurately identifying business needs. To meet these needs, requirements must be defined and addressed in the analysis documents in a way that is clear, consistent, concise, and summarizing. This ensures that all stakeholders understand and there is no room for disagreement. In addition, in order to create trustworthy software, it is essential to analyze and categorize these needs thoroughly. Among the most important things to keep in mind when doing requirements analysis is making sure the requirements still have enough depth, are consistent with each other, and satisfy the demands of the organization. Since functional and non-functional criteria are prone to being confounded when expressed in normal language, manually identifying them is an incredibly tough undertaking. A system would fail during development if it lacked the necessary functional requirements. Problems like project failure, compromised system integrity, or increased costs could also result from disregarding non-functional criteria. This research presents an Associated Requirement Classification Model for standardization in the process of implementing high quality software. The proposed model achieved 98.6% accuracy in feature selection and 98.7% accuracy in requirements classification. In future, optimization techniques can be applied and classical and more sensitive requirements can also be analyzed for software quality.

DECLARATION CONFLICT OF INTEREST:

The authors declare that this manuscript has no conflict of interest with any other published source and has not been published previously (partly or in full). No data have been fabricated or manipulated to support our conclusions.

No funding is applicable and declaration for no financial Interest.

Acknowledge

Acknowledgment The authors declare that they have no conflict of interest. The manuscript was written through contributions of all authors. All authors have given approval to the final version of the manuscript. The article has no research involving Human Participants and/or Animals. The author has no financial or proprietary interests in any material discussed in this article.

COMPLIANCE WITH ETHICAL STANDARDS:

Conflicts of Interest:

The authors declare that they have no conflict of interest. The manuscript was written through contributions of all authors. All authors have given approval to the final version of the manuscript.

Availability of data and material:

Not data and materials are available for this paper. Data sharing not applicable to this article as no datasets were generated or analyzed during the current study'

Ethical Approval:

The article has no research involving Human Participants and/or Animals

Competing Interest:

The author has no financial or proprietary interests in any material discussed in this article.

DECLARATIONS:

Funding:

No Funding is applicable.

Code availability:

The data and code can be given based on the request

Consent to Participate:

The manuscript was written through contributions of all authors. All authors have given approval to the final version of the manuscript.

Consent to Publish:

All authors have given approval to the final version of the manuscript for publication.

References

[1] S. Martínez-Fernández et al., "Continuously Assessing and Improving Software Quality With Software Analytics Tools: A Case Study," in IEEE Access, vol. 7, pp. 68219-68239, 2019, doi: 10.1109/ACCESS.2019.2917403.

[2] S. D. Garomssa, R. Kannan, I. Chai and D. Riehle, "How Software Quality Mediates the

Impact of Intellectual Capital on Commercial Open-Source Software Company Success," in IEEE Access, vol. 10, pp. 46490-46503, 2022, doi: 10.1109/ACCESS.2022.3170058.

[3] S. Moyo and E. Mnkandla, "A Novel Lightweight Solo Software Development Methodology With Optimum Security Practices," in IEEE Access, vol. 8, pp. 33735-33747, 2020, doi: 10.1109/ACCESS.2020.2971000.

[4] D. S. Guamán, J. M. D. Alamo and J. C. Caiza, "A Systematic Mapping Study on Software Quality Control Techniques for Assessing Privacy in Information Systems," in IEEE Access, vol. 8, pp. 74808-74833, 2020, doi: 10.1109/ACCESS.2020.2988408.

[5] W. Behutiye, P. Rodríguez and M. Oivo, "Quality Requirement Documentation Guidelines for Agile Software Development," in IEEE Access, vol. 10, pp. 70154-70173, 2022, doi: 10.1109/ACCESS.2022.3187106.

[6] R. A. Khan, S. U. Khan, H. U. Khan and M. Ilyas, "Systematic Mapping Study on Security Approaches in Secure Software Engineering," in IEEE Access, vol. 9, pp. 19139-19160, 2021, doi: 10.1109/ACCESS.2021.3052311.

[7] Atoum et al., "Challenges of Software Requirements Quality Assurance and Validation: A Systematic Literature Review," in IEEE Access, vol. 9, pp. 137613-137634, 2021, doi: 10.1109/ACCESS.2021.3117989.

[8] H. A. Al-Alshaikh, A. A. Mirza and H. A. Alsalamah, "Extended Rationale-Based Model for Tacit Knowledge Elicitation in Requirements Elicitation Context," in IEEE Access, vol. 8, pp. 60801-60810, 2020, doi: 10.1109/ACCESS.2020.2982837.

[9] M. A. Akbar, S. Mahmood, A. Alsanad, M. Shafiq, A. Gumaei and A. A. -A. Alsanad, "Organization Type and Size Based Identification of Requirements Change Management Challenges in Global Software Development," in IEEE Access, vol. 8, pp. 94089-94111, 2020, doi: 10.1109/ACCESS.2020.2995238.

[10] F. Subahi, "BERT-Based Approach for Greening Software Requirements Engineering Through Non-Functional Requirements," in IEEE Access, vol. 11, pp. 103001-103013, 2023, doi: 10.1109/ACCESS.2023.3317798.

[11] J. A. Khan, L. Liu and L. Wen, "Requirements knowledge acquisition from online user forums", *IET Softw.*, vol. 14, no. 3, pp. 242-253, Jun. 2020.

[12] A. J. Gregory, J. P. Atkins, G. Midgley and A. M. Hodgson, "Stakeholder identification and engagement in problem structuring interventions", *Eur. J. Oper. Res.*, vol. 283, no. 1, pp. 321-340, May 2020.

[13] K. Marner, S. Wagner and G. Ruhe, "Stakeholder identification for a structured release planning approach in the automotive domain" in arXiv:2011.00227, 2020.

[14] R. Kumar, L. E. H. Son, M. Abdel-Basset, I. Priyadarshini, R. Sharma and H. V. Long,"Deep learning approach for software maintainability metrics prediction", IEEE Access, vol. 7, pp. 61840-61855, 2019.

[15] A. Ferrari and A. Esuli, "An NLP approach for cross-domain ambiguity detection in requirements engineering", *Automated Softw. Eng.*, vol. 26, no. 3, pp. 559-598, Sep. 2019.

[16] O. Malgonde and K. Chari, "An ensemblebased model for predicting agile software development effort", *Empirical Softw. Eng.*, vol. 24, no. 2, 2019.

[17] A.-E. H. Abd-Elrahma, A. A.-E. El-Borsaly, E. A.-E. Hafez and S. A. Hassan, "Intellectual capital and service quality within the mobile telecommunications sector of Egypt", *J. Intellectual Capital*, vol. 21, no. 6, pp. 1469-1930, Jul. 2020.

[18] H. T. Nhon, N. Van Phuong, N. Q. Trung and B. Q. Thong, "Exploring the mediating role of dynamic capabilities in the relationship between intellectual capital and performance of information and communications technology firms", *Cogent Bus. Manage.*, vol. 7, no. 1, Jan. 2020.

[19] J. Ahmad, A. W. Khan and H. U. Khan, "Role of critical success factors in offshore

quality requirement change management using SLR", *IEEE Access*, vol. 9, pp. 99680-99698, 2021.

[20] F. N. Colakoglu, A. Yazici and A. Mishra, "Software product quality metrics: A systematic mapping study", *IEEE Access*, vol. 9, pp. 44647-44670, 2021.

[21] A.-J. Molnar, A. Neamtu and S. Motogna, "Evaluation of software product quality metrics" in Evaluation of Novel Approaches to Software Engineering, Cham, Switzerland:Springer, pp. 163-187, 2020.

[22] A. Barcomb, A. Kaufmann, D. Riehle, K.-J. Stol and B. Fitzgerald, "Uncovering the periphery: A qualitative survey of episodic volunteering in free/libre and open source software communities", *IEEE Trans. Softw. Eng.*, vol. 46, no. 9, pp. 962-980, Sep. 2020.

[23] H. Y. Chiang and B. M. T. Lin, "A decision model for human resource allocation in project management of software development", *IEEE Access*, vol. 8, pp. 38073-38081, 2020.

[24] T. E. J. Vos, I. S. W. B. Prasetya, G. Fraser, I. Martinez-Ortiz, I. Perez-Colado, R. Prada, et al., "IMPRESS: Improving engagement in software engineering courses through gamification" in Product-Focused Software Process Improvement, Cham, Switzerland:Springer, 2019.

[25] B. Gezici, N. Ozdemir, N. Yilmaz, E. Coskun, A. Tarhan and O. Chouseinoglou, "Quality and success in open source software: A systematic mapping", *Proc. 45th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, pp. 363-370, Aug. 2019.