Review of Literature on Software Quality

Article · October 2018

CITATIONS
9

READS
12,915

4 authors, including:

Farhan Alebeisat
Tafila Technical University
19 PUBLICATIONS **83** CITATIONS
SEE PROFILE

Zaid Alhalhouli
Tafila Technical University
2 PUBLICATIONS **14** CITATIONS
SEE PROFILE

Tamara E Alshabatat
Mu'tah University
2 PUBLICATIONS **9** CITATIONS
SEE PROFILE

World of Computer Science and Information Technology Journal (WCSIT)
ISSN: 2221-0741
Vol. 8, No. 5, 32-42, 2018

Review of Literature on Software Quality

Farhan M Al Obisat, Zaid T. Alhalhouli, Tamador I. Alrawashdeh, Tamara E. Alshabatat
Computer and Information Technology Department
Tafila Technical University, TTU
Tafila, Jordan

Abstract— the software development industry considers quality a crucial factor in its development. Applying a certain level of standard to Software Quality (SQ) can help ensure customer satisfaction. This study primarily aims to define the different dimensions of SQ, identify the requirements for enhancing SQ, and present the challenges when SQ is restricted. The study also provides a review on the impact of quality and its measurement in the life cycle of software development. It examines the need for a quality standard to measure the increasing quality requirements and size of software. The findings of this study indicate an increasing need for high-quality software. Moreover, it provides a reference for other scholars regarding SQ testing and SQ in fuzzy logic.

Keywords- quality challenges; Software Development Life Cycle (SDLC); software measurement; software quality.

I. INTRODUCTION

In the software development industry, software developers and engineers are primarily concerned with designing software that meet delivery, cost, and quality requirements, a property referred to as software quality (SQ). Customer requirements are collected in the initial stages of a software project. User expectations, required services, and software specifications are seriously considered in these stages because of their effect on SQ. In the early design and development stage, SQ evaluation must be conducted to minimize the effort, time, and cost input into a software product [1].

SQ can be categorized into functional and nonfunctional SQ. Functional SQ encompasses the software features and specifications identified in the early phases, whereas nonfunctional SQ involves features that support the functional requirements (FRs) of the software, i.e., software services.

The success of an overall software system is significantly based on SQ, which is considered a critical design component by developers, users, and project managers [2]. SQ is likewise a crucial factor in evaluating the global competitiveness of any software enterprise [3]. Thus, it is essential in sensitive systems, including control systems and real-time systems, among many others because poor quality may result in financial loss, failed missions, and even loss of human life [4]. SQ may be described as a product attribute that meets the stringent performance and FRs, specific development criteria, and inherent functions that all professionally designed software must have [4]. Since the arrival of computer programs, achieving SQ has been difficult, and various definitions of SQ have been proposed. Some definitions have been standardized, but the majority of definitions are deemed too vague and theoretical. For instance, the International Standard Organization (ISO) defines SQ as an array of product attributes by which the quality of a software is illustrated and appraised, ANSI Standard defines it as the sum of all the characteristics of a software product or service that represent its capability to satisfy customer requirements, while IEEE Standards defines SQ as the array of features of a software product that represent its capability to satisfy specific needs [2]. SQ is the extent to which a process, component, or system fulfills a specific requirement, that is, how much it fulfills customer needs or expectations through product or service features, thus providing customer satisfaction [5].

SQ is the extent to which characteristics such as reliability, maintainability, efficiency, portability, usability, and reusability are designed into a software product or service [6]. Numerous scholars from different fields have attempted to develop suitable models to define SQ, including ISO/IEC 9126 model [7], Boehm's model [8], Dromey's model [9], and FURPS Model [10]. The most famous among these models is ISO/IEC 9126 [7] because it incorporates the features of almost all the other models.

SQ is traditionally composed of software reliability, accuracy, maintainability, and usability [11]. Owing to its multidimensionality, every organization is obliged to identify which aspects of quality are important to them. Two techniques for ensuring the quality of a software product include (1) ensuring the development process of the product and (2) evaluating of the quality of the end product [12].

SQ is dependent on the access of designers to the ideal materials, devices, processes, management strategies, and latest technological developments [13]. Numerous authors have

highlighted that the success or failure of a software product in a competitive market is reliant on its quality [14, 15].

ISO 9126 standard defines quality as "the totality of features and characteristics of a product or service that bears on its ability to satisfy given needs" [16]. The development and improvement of software and quality, respectively, are essentially organizational in nature, not technical [17, 18]. The ISO 9126 model (ISO/IEC 9126:2001) consists of a four-part standard for "Software engineering – product quality," which includes quality model, external metrics, internal metrics, and quality-in-use metrics, respectively [19]. A recently proposed model, McCall's model, describes SQ as the characteristics of a software product that represent its capability to satisfy both explicit and implicit requirements. It proposes six high-level, independent quality measures, namely, Reuse based on Object-oriented Technology, Dromey's Quality Model, Software Assurance Technology Center Quality Model, Quality Model for Object-oriented Design, Metric-based Quality Model for Object-oriented Design, and Software Metrics. These measures comprise a set of software features by which the product quality is depicted and appraised [20]. SQ has also been depicted [21] according to its product characteristics: (i) internal quality (i.e., mode of product development), such as software complexity and configuration; and (ii) external quality (i.e., product functionality), such as serviceability and reliability [22]. The three most common SQ definitions are as follows:

1. Software quality is determined by a set of quality factors [23, 24]
2. Software quality is determined by user satisfaction [25].
3. Software quality is determined by unexpected software performance or errors [26, 27].

Several definitions of software assurance have been put forward. In the current study, we define the SQ assurance (SQA) as crucial to the success of any software company. SQA guarantees product quality by monitoring its optimum functionality and documenting its performance for maintenance. Apart from assessing the application, it monitors and manages the development processes and condition of all software products [16]. SQA is a strategic and methodical evaluation of the quality of a software product and its conformance to specified processes, practices, and criteria [28].

SQA comprises activities for assessing the process applied to developed or manufactured software products. Its primary objective is to ensure that a product meets the minimum acceptable level of confidence and satisfies the functional technical requirements. SQA check that standard steps are followed in evaluating a software product. It covers the entire development process, such as defined requirements, software design, coding, source code control, code reviews, change management, configuration management, testing, release management, and product integration [29]. SQA mainly serves to preserve the product quality [30]. SQA measurements are metrics-based and designed to help enterprises achieve high of SQ [28]. A metric is defined as "a standard of measurement, a mathematical function that associates real nonnegative numbers" [31]. The lack of an effective SQA is a major factor in the failure of many software projects. Hence, SQA is vital to the software development life cycle (SDLC) because of its capability to markedly diminish potential risks and enhance the success of a project [32].

SQA offers users and designers a guarantee that a software product is defect-free (both intentional and accidental defects) during its life cycle and that it performs as expected [33].

This paper is organized as follows. Section two discusses the concept of SQ, its definitions, and purpose, as well as the common models employed to describe SQ. Section three provides a literature review on the measurements and challenges of SQ, its application in SDLC, and use in assessing software risk and fuzzy logic (FL). It also presents the two types of SQ requirements and software testing.

II. Measurement of Software Quality

A method to gauge the quality, cost, and effectiveness of a project and its processes is essential in any software project. A project without the capability to measure such factors cannot be completed successfully [34]. SQ is assesses on the basis of its capability to meet user requirements and realize the purpose it was designed for [3]. A key element in controlling, managing, and refining the software development process is software measurement [3].

The outputs of product development throughout the analysis, design, and coding stages must be measured, observed, and managed so they can be verified against pre-specified criteria. Moreover, product development efforts must be upgraded at each stage to reduce costs and maintain or improve market competitiveness [35]

The importance of SQA measurements is highlighted in international standards such as ISO 9000-3, which is a guideline for software development, and ISO 9001 [36]. In addition, the capability maturity model [37] depicts the need for measurements during software development. Although such standards and models highlight the significance of software measurements, detailed guidelines on carrying out SQA measurements and the objectives of such measurement programs are lacking [38].

Product failures can be prevented by conducting software measurements. These measurements alert developers and engineers about development mistakes and thus avert errors and flaws both before and during the early stages of product release, as well as assist in monitoring software development [38].

ISO/IEC 9126 model [7] is provides a broad definition of SQ in terms of six characteristics for software assessment, namely, functionality, efficiency, maintainability, portability, reliability, and usability. This model covers nearly all the aspects mentioned in early models, such as Boehm's model [8], Dromey's model [9], and McCall's model [39], and includes the implicit and explicit quality attributes of a software product. However, how such characteristics and sub-characteristics can be measured is not specified in the ISO/IEC 9126 model.

TABLE 1 SUMMARIZES THE CHARACTERISTICS AND SUB-CHARACTERISTICS OF THIS MODEL [3, 7].

Characteristics	Subcharacteristics	Definitions
Functionality	Suitability	Software functionality characteristic refers to the appropriateness of the functions of the software.
	Accurateness	Correctness of the functions.
	Interoperability	The ability of a software component to interact with other components or systems.
	Compliance	Compliant capability of software.
	Security	Unauthorized access to the software or software functions.
	Maturity	Frequency of failure of the software.
Reliability	Fault tolerance	Ability of software to recover from component, or environmental, failure.
	Recoverability	Bring back a failed software/system to full operation, including data and network connections.
	Understandability	Relates to understanding the software/ easy to understand (Human Computer Interaction methods).
Usability	Learnability	Effort for learning different users
	Operability	Easily to operate the software by a given user in a given environment.
Efficiency	Time behaviour	Response times for a given thru put, i.e. transaction rate.
	Resource behaviour	resources used characterized, i.e. memory, cpu, disk and network usage.
	Analyzability	The ability to identify the root cause of a failure within the software.
Maintainability	Changeability	The ability to change a software/ system.
	Stability	The sensitivity to change of a given system that is the negative impact that may be caused by system changes.
	Testability	Testing a software/ system (change).
	Adaptability	Change the system to new specifications or operating environments.
Portability	Installability	Install the software.
	Conformance	Relates to portability. One example would be Open SQL conformance which relates to portability of database used.
	Replaceability	The ability to exchange a given software component within a specified environment.

Table 1 summarizes the characteristics and sub-characteristics of this model [3, 7].

In the ISO 9126 standard [40], SQ has six major quality characteristics, namely, functionality, reliability, usability, efficiency, portability, and maintainability, all of which have sub-characteristics. These quality characteristics are sorted into external and internal quality sets. Functionality, reliability, usability, efficiency, flexibility, friendliness, and simplicity are some of the external quality characteristics that customers expect from software products because these can be easily observed. By contrast, maintainability, portability, reusability, and testability are some of the internal quality characteristics that developers consider in a software product because these are linked to their efforts during the development stage [38].

Two types of measurement methods have been developed to handle the external and internal quality characteristics of software quality. These methods apply both external and internal measurements and complement each other. Nonetheless, owing to the different objectives of each method, they cannot completely substitute for the other [38].

The ISO 9126 quality model has three perspectives on quality that distinguish the characteristics and sub-characteristics of software quality [21]. The first two perspectives, namely, external and internal, comprise the same 6 characteristics and 26 sub-characteristics (Figure 1). The third perspective, quality in use, has four unique characteristics. All three perspectives complement one another. Internal quality affects external quality, which in turn influences quality in use. Internal quality measures serve as an early gauge of external quality.

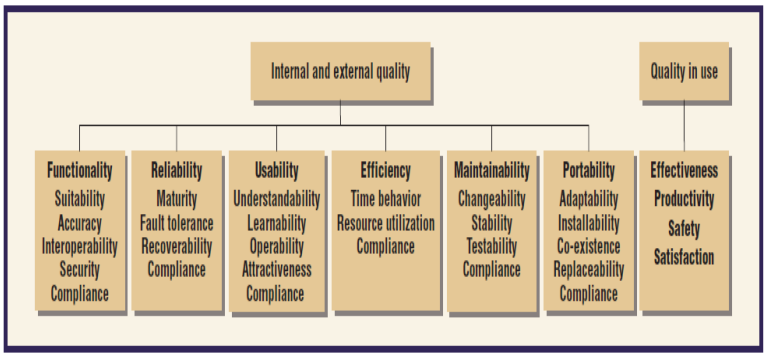

Figure 1. ISO/IEC 9126 quality model [21].

The related properties of SQ present researchers an effective means to understand the quality being measure, the meaningful operations on the measured values, and the interpreted results [41]. Software methodologies such as V-Model, waterfall method, and RUP are classified as traditional software development methodologies or heavyweight methodologies [42]. A software development methodology is the framework utilized to design, manage, and monitor the development process of an information system [43]. An SQ model serves as the framework for evaluating the quality attributes of a software product [44]

ISO/IEC IS 9126-1 [7] defines a quality model as “the set of characteristics, and the relationships between them that provides the basis for specifying quality requirements” and International Journal of evaluation". The fundamental factors (i.e., characteristics) are defined through the models constructed specifically to evaluate SQ [7].

Considerable efforts have been exerted to develop models of software product quality. Many authors have carried out

literature reviews on quality models and included some benchmarking, including Al-Badareen [45], Dubey [46], Al-Qutaish [47], Ghayathri [48], and Samadhiya [49]. These cited works are all concerned with basic quality models.

External measurements are those that require the end product and, in the majority of cases, users' participation, whereas internal measurements are automatically performed on the program code via internal software metrics. Furthermore, external measurements can regulate internal measurement tools and provide the perceived SQ measurements of a user, whereas internal measurements can help prevent errors and defects in the early stages of development [50].

The ultimate purpose of QA is to design and manufacture software capable of reducing vulnerabilities and satisfying specific standards of function, consistency, and performance. Quality assurance (QA) aims to complete a project according to previously agreed on conditions, criteria, and functionalities without flaws and potentials for failure [51]. Software metrics and SQ models are regarded as primary references in the SQ assessment, though specific methods for assessing SQ via software metrics have yet to be established [52].

Scarpino and Kovacs [53] investigated the negative effects on an organization when an SQA tool is applied without first setting up an SQ process. In their research, an organization that applied an SQA tool was used selected. Data were collected through open observational analysis and interviews by an internal QA expert and an external specialist, respectively. They author found that team members were not provided proper guidance and training in using the SQA tool and that documentation on how the system would align with the company's SDLC was nonexistent. The brief period and lack of prior communication with team members also resulted in high user resistance toward the application of the tool. Additionally, the capability of the tool to meet company requirements was not properly assessed, and inconsistencies in the reported progress of tool implementation were observed.

IEEE, ISO, and other organizations have attempted to standardize SQ by forwarding models that incorporate related SQ characteristics and sub-characteristics [52].

Challenges in Software Quality

Software firms inevitable encounter numerous challenges in their efforts to provide high-quality software and achieve user satisfaction [54]. The factors that may impair SQ management include bureaucracy, inefficient management, tight deadlines, developer ego, conflicting opinions and principles, additional costs (e.g. for tool purchases), insufficient resources to automate the development process, absence of organizational training on quality standards, low familiarity with and understanding of the process, an organizational lack of quality management structure, disapproval from top management, and futility of an early version of the process [54]. The respondents confirmed that the initiatives to implement SQA practices are facing serious obstacles [32].

Several scholars have based the input to measure SQ on the perspectives of the developers, managers, and users regardless of the attribute's relevance [55]. A disadvantage of this approach is that even though the developer may be unaware of how the user assessed the SQ, he/she will still judge the qualities of the project manager and user. Likewise, the user may be unaware of how developers assess the SQ but will still judge the developer's quality. This deficiency may lead to erroneous results [3]. In the software industry, software must function, released quickly in the market, and offer competitive value, all of which must be accomplished with limited resources [56]

III. Software Quality in the Software Development Life Cycle

SDLC refers to the time period from conceiving the software to its release [57]. This life cycle is divided into various phases as follows: Feasibility study, Analysis and specification of requirements, Designing, Implementation or Coding, Testing, Operations, and Maintenance [57]. These phases can change depending on client demand and situations and on the SDLC model employed [57]. SDLC concerns the course of building or maintaining software systems [58]. It represents the entire development process that a software development organization must employ to successfully develop a software product. The modern SDLC has two main categories, traditional and agile [59]. Agile SDLC methods aim to shorten the life cycle, minimize bug rates, enhance customer satisfaction, and accommodate evolving business requirements during the development process [60]. SQA covers the entire SDLC, including software design, coding/implementation, source code control, code reviews, configuration, and testing at both the development and user end, as well as the management of changes and market release [61]. There are many reasons, and one is the improper choice of SDLC model [62]. The fundamental concepts of software development methodologies must be properly understood when evaluating the best SDLC methodology [63].

IV. Software Quality and Software Risk

Software projects confront various risks throughout their life cycle. Risk is defined as a potential condition or event that may adversely influence the success of a project; it refers to the possibility of loss or damage or a factor that involves a potential danger [64, 65, 66]. Risks affect the reliability, cost, timetable, and quality of a software product [67].

TABLE 2: TYPES OF SOFTWARE RISKS [67]

Risks in Software Requirements	1. Insufficient analysis of changes in requirements 2. Extended changes in requirements 3. Lack of documentation for requirements 4. Poor definition of requirements 5. Ambiguous changes in requirements
Risks in Software Cost	1. Erroneous estimation of project costs 2. Unrealistic schedule 3. Defective or malfunctioning hardware
Risks in Software Scheduling	1. Insufficient budget 2. Human errors 3. Limited knowledge on techniques and tools 4. Necessity of long-term personnel training
Risks in SQ	1. Inadequate documentation 2. Nonexistent project standards and estimation 3. Nonexistent design documentation 4. Insufficient budget 5. Human errors 6. Unrealistic schedule

Understanding the risks in SQ is important. Numerous risks in SQ have been represented in early studies, and relations between quality risk events have been observed. Some relations are based on tool and hardware failures, some pertain to human errors, limited knowledge, supply disagreements between developers and clients, and requirements and costs that can affect SQ [68].

Identifying the different risks in software engineering projects is a difficult or even impossible endeavor. The most important risks in such projects are categorized as software requirement risks, software cost risks, software scheduling risk, SQ risks, and software business risks [67].

SQ management involves a set of activities that delineate the process of controlling and managing the SQ, such as SQA, SQ plan, and SQ control [29]. SQA is the process that guarantees the quality of a software product through the application of different methods, knowledge, guidelines, and criteria in the course of its development life cycle [29].

Different SQ techniques, including code reviews, process improvements, software testing, risk management, change management, and configuration have been proposed, all of which can be implemented manually and automatically via specialized tools [32]. Software risks can be broken down to external and internal risks; the former originates from factors outside the organization and are difficult to manage, and the latter stems from factors inside the organization. These risks can likewise be categorized into process, project, and product risks [69].

Quality is the ability to implement permission, certification, and intentional denial-of-service attacks [70]. Numerous quality-related issues and management responsibilities have been identified in literature. Moreover, management is a key part of SQA [71].

This unsuccessful development is mainly attributed to the fact that by the time problems are identified, it is too late to rectify them. Developers and project managers must possess the foresight to ascertain potential risks to decrease cost and enhance quality [72].

V. SOFTWARE QUALITY AND SOFTWARE REQUIREMENT

Functional, structural, and process qualities are the three key aspects of SQ [73]. Functional quality refers to the capability of the software to properly perform its tasks according to user needs and intended objectives. Structural quality refers to the resilient structure of the code itself and is difficult to test compared to functional quality. These first two qualities are the most common aspects discussed in SQ literature. However, the last and most critical aspect is process quality.

The specification of FRs is the first stage of software development and is considered the most important in the software life cycle. Requirements designed in this stage affect the succeeding life cycle stages and, subsequently, the SQ [74]. Recognizing and specifying requirements are key components in the eventual success or failure of a software project [21]. Thus, prioritizing requirements is critical in software development [75].

Requirements are usually communicated via natural language [76, 77] and categorized into FRs and nonfunctional requirements (NFRs). FRs only state the required functionality, such as "the system must allow users to log in," whereas NFRs are statements of human needs (i.e., cognitive requirements), such as "the user login must be simple and efficient" [78].

Given that requirements frequently change throughout a project's life cycle, the resulting design modifications disrupts the organizational processes applied to manage requirements, thus causing a wide range of potential defects [79]. The typical difference between FRs and NFRs is in how the system carries out a task contrary to what the system is expected to perform [80, 81]

a. Functional Requirement Quality

The two most popular definitions of FRs are as follows: (1) a statement that expresses the objective of a product or process to achieve the expected performance and/or results, and (2) a requirement that stipulates the mandated function of a system or system component [82].

Scheduling and budgeting aspects should also be included in SQA activities in addition to the technical aspects of FRs. This expanded scope is attributed to the close association between scheduling and/or budget failure and the fulfillment of functional technical requirements. Projects with a tight

timetable are frequently burdened by professionally "dangerous" professional revisions in the project schedule that can damage the possibility of fulfilling FRs. Projects with limited budgets and resources allocated to its maintenance confront the same negative consequences as well [83]. The basic explanation for the similar effects is that NFRs also describe behavioral properties [84] and should thus be considered in the same manner as FRs in the development process [85].

FRs are expressed through their intended use and describes users' interactions with the software product. FRs comprise the scope, objective, perspective, tasks, user characteristics, detailed functionalities, software attributes, interface requirements, and database requirements [86].

FRs are defined by the expected inputs and outputs expected, otherwise known as Functionality (F), while NFRs are defined as Usability (U), Reliability (R), Performance (P), and Product Support (S) [87], as shown in Figure 2. The main problem in the table is that several key features (e.g., portability) are not included.

SQ is defined as "How well the software complies with or conforms to a given standard or requirements, based on FRs or specifications." This attribute can likewise be described as the aptness of a software product to satisfy a specific objective or how it compares as a competitive product in the marketplace [88].

b. Nonfunctional Requirement Quality

NFRs are defined as software requirements that designate how software products will accomplish the tasks they were designed to perform. They are also referred to as design constraints. Given that NFRs are difficult to test on occasion, they commonly undergo subjective assessments [82].

A universally acknowledged definition for NFR has yet to be proposed [89]. One definition states that "NFRs not only introduce quality factors but also represent global constraints under which a system must operate." Different from FRs which tackle specific problems, NFRs, also referred to as quality requirements [90, 91], are usually applied via precise localized modules or mechanisms.

Pohl [92] argues that the term "nonfunctional" is misleading and that the term "quality requirements" should be utilized for product-related and non-constraint NFRs. The value of NFRs in software and systems development cannot be repudiated, yet the majority of discussions and studies on how NFRs should be regarded are still focused on distinguishing them from FRs [93, 94]. NFRs offers support for design decisions and constraints by presenting how a required functionality may be achieved to fulfill the quality concerns of stakeholders [91, 95, 96, 97, 98, 99, 100]. NFR definitions remain vague [101,102] and unquantified [103], and as a result, studying and testing NFRs are continuing challenges [101,102, 103].

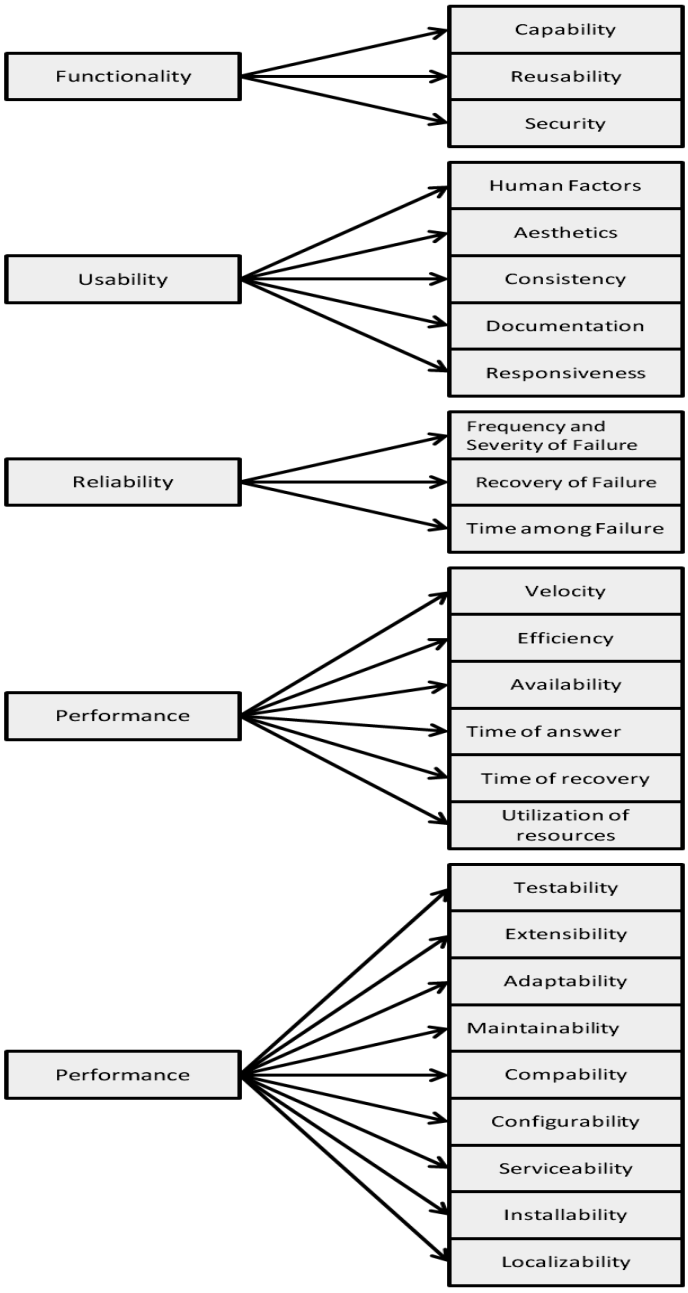

Figure 2: FURPS Model

VI. Software Quality and Fuzzy Logic Technique

In 1965, Zadeh introduced FL, motivated primarily by the inaccuracies in measurement methods [63]. FL technique is applied using heuristic information and indefinite inputs to achieve complicated functions in modeling complex systems. In a world full of ambiguities, FL has succeeded in diverse fields such as decision support, dynamic control, and other expert systems. Therefore, fuzzy systems are essential in risk estimation because it can handle crisp values [104].

In [70], an intelligent software early warning system based on FL was proposed. The warning system utilizes a combined set of software measurements to gauge the associated risks in lagging behind schedule, exceeding the allotted budget, and

producing poor quality in software development and maintenance. The measurements are derived from various perspectives to help address inaccurate, vague, and partial information, as well as settle conflicts in an uncertain environment in software risk assessment by utilizing fuzzy inference rules, fuzzy linguistic variables, and fuzzy sets.

At the onset of developing an SQ prediction model, the factors that greatly affect SQ and the number of residual errors must be identified. In [105] , an FL-based approach was proposed to calculate error-prone modules via inspection information. By representing ambiguous and insufficient information, FL enables machines to comprehend the world in the same manner as humans [106].

A fuzzy set S of a discourse universe U is indicated by a membership function associated with each element y of U in a number in the interval [0, 1], which denotes the membership grade of y in S [107]. FL-based reasoning offers novel perspectives in software development, including accumulative software life cycle and fuzzy artifacts [107]. Given the difficulties in implementing traditional model-based approaches, FL-based reasoning aids in gauging the reusability of software components. The growing complexity of modeling the problem with various components to measure reusability has given rise to another renowned technique called neuro-fuzzy approach [108].

FL is a potent technique for tackling problems with complex and vague phenomena, which can be evaluated only linguistically rather than numerically [109]. Furthermore, FL is useful in estimating multi-criteria decision problems [110, 111]. Numerous fields, such as artificial intelligence and control theory, have benefitted from FL because it explains a mathematical system that can be applied to model the inference framework that facilitates proper human reasoning capabilities [101]. The variable of FL may have a truth value that ranges from 0 and 1. FL provides an expedient means of generating precise mapping between output and input spaces owing to the natural expression of fuzzy rules [113].

In their work, "Software Quality Assessment Based on Fuzzy Logic Technique," Mittal et al. [114] proposed a detailed FL-based approach to measure SQ. A fuzzy system represents a mapping between linguistic terms (e.g., "very small") ascribed to variables [115]. Fuzzy sets are depicted by membership functions that associate real numbers in the interval [0, 1] with points in the fuzzy sets; this function is known as grade or degree of membership [6]. Attempts have been made to use FL with historical data to predict error-prone code modules [105]. FL techniques have been used to compute metric tree scores, which were then evaluated and experimented on using other methods. Appropriate components of the Mandeni fuzzy inference engine were utilized to satisfy customer demands, resulting in the technique being constructed according to user requirements [116].

VII. Software Quality Testing

Testing involves various measurement methods to improve SQ and is in the broad category of software management practices known as QA. Similar to other activities such as design and code inspections and defect tracking, software testing is oriented toward "detection" [117] and is a method to detect system errors. It helps find and debug system errors, mistakes, faults, and failures and guarantees the expected functionality of the system [118]. Unit testing is conducted only on small units. In integration testing, various integrated modules are assessed, whereas in system testing, the entire system is assessed. The method of software system testing affects the way SQ is evaluated [118].

Software that is user-friendly, error-free, and provides client satisfaction is considered to be of high quality. Appropriate software testing techniques are thus necessary to enhance and maintain quality [58, 119]. Software testing in SQA involves assessing the functionality, regression, load, performance, and security of a software product [65]. This process offers information on whether a software program or application fulfills the technical and business requirements that informed its design and development and performs as needed [120]. Software testing helps identify when problems arise and diagnose the root of such problems.

As an activity that implements software in a controlled manner, software testing answers the question "Does the software behave as specified?" Furthermore, it is frequently employed along with verification and certification [120]. It is a process that both verifies the results of SQA and achieves the intended quality [118]. Software testing developed along with the development of the software and is thus an essential element in SDLC. It provides a guarantee that the system will perform with the required functionality. Consequently, numerous software testing systems and strategies have been implemented, including White-box, Black-box, and Grey-box testing [118]. Software testing can also be conducted in three ways as follows: unit testing, in which testing is done only on small units; integration testing, in which different integrated modules are assessed; and system testing, in which the entire system is tested [118].

As we know Mobile Learning (mLearning) characterize a new trend of learning that uses innovations like wireless communication, personal digital assistants, digital content from traditional textbooks, and other sources to provide a dynamic learning environment [121]. Some studies have been done as a case study in Jordan universities [122].

On the other hand, E-learning involves the use of the Internet as a communications medium between instructors and students who are separated by physical distance. Wireless networks have become very common in this environment, often replacing wired networks, in order to provide mobile access to educational systems and the Internet for students and staff. But these networks must be secured [123].

In [124]), the study was devoted to describe the government of Jordan Initiative toward E-Government and to explain the blue print and roadmaps provided to the government of Jordan. the study has been investigated all the necessary information technology requirements that are vital to build an E-Government in Jordan and assess the status of E-Government initiative achievements in Jordan from many aspects; E-Connectivity and Infrastructure, E-Human Resources, E-payment, E-leadership and Information

Technology Industry to determine the problem and challenges that faces this project.

VIII. Conclusion

This study establishes and describes the current state of SQA. It focuses on SQ measurement, which can strengthen the quality of software products or processes. The challenges in SQ, SQ in SDLC, relation of SQ and software risk, two types of SQ requirements (i.e., NFRs and FRs), FL technique, and SQ testing models and methods are also discussed in terms of their implications in SQA. Additionally, this study analyzed early studies conducted on SQ. These studies depicted the development of the SQ models and the increase in SQ features throughout the years.

REFERENCES

[1] Azar, D., Harmanani, H., & Korkmaz, R. (2009). A hybrid heuristic approach to optimize rule-based software quality estimation models. Information and Software Technology, 51(9), 1365-1376.

[2] Youness, B., Abdelaziz, M., Habib, B., & Hicham, M. (2013). Comparative Study of Software Quality Models. IJCSI International Journal of Computer Science Issues, 10(6), 1694-0814.

[3] Challa, J. S., Paul, A., Dada, Y., Nerella, V., Srivastava, P. R., & Singh, A. P. (2011). Integrated Software Quality Evaluation: A Fuzzy Multi-Criteria Approach. JIPS, 7(3), 473-518.

[4] Suman, M. W., & Rohtak, M. D. U. (2014). A Comparative Study of Software Quality Models. International Journal of Computer Science and Information Technologies, 5(4), 5634-5638.

[5] Hossain, A., Kashem, M. A., & Sultana, S. (2013). Enhancing software quality using agile techniques. IOSR Journal of Computer Engineering, 10(2), 87-93.

[6] Gupta, D., Goyal, V. K., & Mittal, H. (2011). Comparative study of soft computing techniques for software quality model. International Journal of Software Engineering Research & Practices, 1(1), 33-37.

[7] International Organization for Standardization, & International Electrotechnical Commission. (2001). Software Engineering--Product Quality: Quality model (Vol. 1). ISO/IEC Available at www.iso.org

[8] Boehm, B. W., Brown, J. R., & Lipow, M. (1976, October). Quantitative evaluation of software quality. In Proceedings of the 2nd international conference on Software engineering (pp. 592-605). IEEE Computer Society Press.

[9] Dromey, R. G. (1995). A model for software product quality. IEEE Transactions on software engineering, 21(2), 146-162.

[10] Singh, I. (2013). Different Software Quality Model. International Journal on Recent and Innovation Trands in Computing and Communication, 1(5), 438-442.

[11] R.C. Dromey, A model of software product quality, IEEE Transactions on software Engineering (February) (1995) 146–162.

[12] Kannabiran, G., & Sankaran, K. (2011). Determinants of software quality in offshore development–An empirical study of an Indian vendor. Information and Software Technology, 53(11), 1199-1208.

[13] Li, E. Y., Chen, H. G., & Cheung, W. (2000). Total quality management in software development process. The Journal of Quality Assurance Institute, 14(1), 4-6.

[14] Luftmann, J., & Kempaiah, R. (2008). Key issues for IT executives 2007. MIS Quarterly Executive, 7(2).

[15] Tian, J. (2004). Quality-evaluation models and measurements. IEEE software, 21(3), 84-91.

[16] Agrawal, M., & Chari, K. (2007). Software effort, quality, and cycle time: A study of CMM level 5 projects. IEEE Transactions on software engineering, 33(3)..

[17] Issac, G., Rajendran, C., & Anantharaman, R. N. (2006). An instrument for the measurement of customer perceptions of quality management in the software industry: An empirical study in India. Software Quality Journal, 14(4), 291-308.

[18] Gopal, A., & Koka, B. R. (2009). Determinants of service quality in offshore software development outsourcing. In Information Systems Outsourcing (pp. 497-523). Springer, Berlin, Heidelberg.

[19] S. Birla, M. Johansson, "Quality Requirements for Software dependent Safety-critical Systems History current status and future needs", 26.01. 2014, [online] Available: http://pbadupws.nrc.gov/docs/ML1424/ML14247A205.pdf.

[20] Ortega, M., Pérez, M., & Rojas, T. (2003). Construction of a systemic quality model for evaluating a software product. Software Quality Journal, 11(3), 219-242.

[21] Boegh, J. (2008). A new standard for quality requirements. IEEE Software, 25(2), 57.

[22] Gorla, N., & Ramakrishnan, R. (1997). Effect of software structure attributes on software development productivity. Journal of Systems and Software, 36(2), 191-199.

[23] Iee, E. (1990). IEEE standard glossary of software engineering terminology.

[24] International Organization for Standardization. (1994). ISO 8402: 1994: Quality Management and Quality Assurance-Vocabulary. International Organization for Standardization.

[25] Deephouse, C., Goldenson, D., Kellner, M., & Mukhopadhyay, T. (1995, January). The effects of software processes on meeting targets and quality. In System Sciences, 1995. Proceedings of the Twenty-Eighth Hawaii International Conference on(Vol. 4, pp. 710-719). IEEE.

[26] Carey, D. (1996). Is software quality intrinsic, subjective, or relational?. ACM SIGSOFT Software Engineering Notes, 21(1), 74-75.

[27] Lanubile, F., & Visaggio, G. (1997). Evaluating predictive quality models derived from software measures: lessons learned. Journal of Systems and Software, 38(3), 225-234.

[28] Agarwal, R., Nayak, P., Malarvizhi, M., Suresh, P., & Modi, N. (2007, August). Virtual quality assurance facilitation model. In Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference on (pp. 51-59). IEEE.

[29] Sharma, E. A., Padda, E. S., & Kaur, E. J. "New Approach Towards Ensuring Software Quality.", International Journal of Engineering Research and Applications (IJERA), Vol. 2, Issue 1, Jan-Feb 2012, pp. 452-454.

[30] Boehm, B., Chulani, S., Verner, J., & Wong, B. (2007, May). Fifth workshop on software quality. In Software Engineering-Companion, 2007. ICSE 2007 Companion. 29th International Conference on (pp. 131-132). IEEE.

[31] Dictionary, M. W. (2013). 2013 Merriam-Webster, Incorporated. An Encyclopedia Britannica Company.

[32] Sowunmi, O. Y., Misra, S., Fernandez-Sanz, L., Crawford, B., & Soto, R. (2016). An empirical evaluation of software quality assurance practices and challenges in a developing country: a comparison of Nigeria and Turkey. SpringerPlus, 5(1), 1921.

[33] Committee on National Security Systems. National Information Assurance Glossary (April 26, 2010).

[34] http://www.softwaretestinghelp.com/software-test-metrics-and-measurements/ accessed 29 January 2018.

[35] Kevitt, M. (2010). Best Software Test & Quality Assurance practices in the project life-cycle. An approach to the creation of a process for improved test & quality assurance practices in the project life-cycle of an SME (Doctoral dissertation, Dublin City University).

[36] ISO, B. (2000). 9001: 2008 Quality management systems. Requirements. International Organization for Standardization.

[37] Paulk, M. C., Curtis, B., Chrissis, M. B., & Weber, C. V. (1993). Capability maturity model, version 1.1. IEEE software, 10(4), 18-27.

[38] Stavrinoudis, D., & Xenos, M. N. (2008, June). Comparing internal and external software quality measurements. In JCKBSE (pp. 115-124).

[39] McCall, J. A., Richards, P. K., & Walters, G. F. Factors in Software Quality, 1977, Vol. I, II, and III, US Rome Air Development Center Reports-NTIS AD/A-049 014. NTIS AD/A-049 015 and NTIS AD/A-049 016, US Department of Commerce.

[40] Iso, I., & Std, I. E. C. (2001). 9126 Software product evaluation–quality characteristics and guidelines for their use. ISO/IEC Standard, 9126.

[41] Jorgensen, M., & Shepperd, M. (2007). A systematic review of software development cost estimation studies. IEEE Transactions on software engineering, 33(1).

[42] Ņikiforova, O., Sukovskis, U., & Ņikuļšins, V. (2008). Integration of MDA framework into the model of traditional software development. publication. editionName, 229-239.

[43] Sommerville, I., & Prechelt, L. (2010). Software testing. Software Engineering, 9th edn. Addison-Wesley.

[44] Behkamal, B., Kahani, M., & Akbari, M. K. (2009). Customizing ISO 9126 quality model for evaluation of B2B applications. Information and software technology, 51(3), 599-609.

[45] Bassam, A. B. A. (2011). Software Quality Evaluation: User's View. International Journal of Applied Mathematics and Informatics, (3), 200-207.

[46] Dubey, S. K., Ghosh, S., & Rana, A. (2012). Comparison of software quality models: an analytical approach. International Journal of Emerging Technology and Advanced Engineering, 2(2), 111-119.

[47] Al-Qutaish, R. E. (2010). Quality models in software engineering literature: an analytical and comparative study. Journal of American Science, 6(3), 166-175.

[48] Ghayathri, J., & Priya, E. M. (2013). Software Quality Models: A Comparative Study. International Journal of Advanced Research in Computer Science and Electronics Engineering (IJARCSEE), 2(1), pp-042.

[49] Samadhiya, D., Wang, S. H., & Chen, D. (2010, October). Quality models: Role and value in software engineering. In Software Technology and Engineering (ICSTE), 2010 2nd International Conference on (Vol. 1, pp. V1-320). IEEE.

[50] Shepperd, M. (1993). Software engineering metrics I: measures and validations. McGraw-Hill, Inc..

[51] Iacob, I. M., & Constantinescu, R. (2008). Testing: First Step Towards Software Quality. Journal of Applied Quantitative Methods, 3(3).

[52] Khosravi, K., & Guéhéneuc, Y. G. (2004, November). On issues with software quality models. In Proceedings of the 11th Working Conference on Reverse Engineering (pp. 172-181).

[53] Scarpino, J., & Kovacs, P. (2008). An Analysis of a Software Quality Assurance Tool's Implementation: A Case Study. Journal of the International Association for Computer Information Systems, 9(2), 9.

[54] Elgebeely AR (2013) Software quality challenges and practice recommendations. In: IBM. http://www.ibm.com/developerworks/rational/library/software-quality-challenges-practice- recommendations/. Accessed 13 March 2017.

[55] Srivastava, P. R., Singh, A. P., & Vageesh, K. V. (2010). Assessment of software quality: A fuzzy multi-criteria approach. Evolution of Computation and Optimization Algorithms in Software Engineering Applications and Techniques, IGI Global USA, 200-219.

[56] Srivastava, P. R., Jain, P., Singh, A. P., & Raghurama, G. (2009, December). Software quality factor evaluation using Fuzzy multi-criteria approach. In IICAI (pp. 1012-1029).

[57] Snöbohm, J. (2015). A Study of Creativity and Innovation Support Within an Agile Context: Applied on a Scrum Team.

[58] Singh, S., Singh, S., & Singh, G. (2010). Reusability of the Software. International Journal of Computer Applications (0975–8887) Volume, 7, 38-41

[59] Leau, Y. B., Loo, W. K., Tham, W. Y., & Tan, S. F. (2012). Software development life cycle AGILE vs traditional approaches. In International Conference on Information and Network Technology (Vol. 37, No. 1, pp. 162-167).

[60] Cho, J. (2008). Issues and Challenges of agile software development with SCRUM. Issues in Information Systems, 9(2), 188-195.

[61] Scarpino, J. J., & Chicone, R. G. (2014). The Quality of Agile-Transforming A Software Development Company's Process: A Follow-Up Case Study. Issues in Information Systems, 15(2).

[62] Clara, V. T. (2013). SDLC and model selection: a study International Journal of Application or Innovation in Engineering & Management (IJAIEM) , Volume 2, Issue 1, January 2013.

[63] Seema, S. (2012). Analysis and tabular comparison of popular SDLC models. International Journal of Advances in Computing and Information Technology.

[64] Antonov, A., Nikolov, V., & Yanakieva, Y. (2006). Risk Simulation in Project Management System. In International Conference on Computer Systems and Technologies-Compsystech.

[65] Galorath, D. D., & Evans, M. W. (2006). Software sizing, estimation, and risk management: when performance is measured performance improves. CRC Press.

[66] Kruchten, P. (2004). The rational unified process: an introduction. Addison-Wesley Professional. chapter 7.

[67] Agrawal, A., & Maurya, L. S. Implementing Fuzzy Logic for Software's Risk and Quality Estimation. note : Published in National Conference in SRMS CET, Publication Date: Jan 24, 2014 Implementing Fuzzy Logic for Software's Risk and Quality Estimation

[68] Hoodat, H., & Rashidi, H. (2009). Classification and analysis of risks in software engineering. World Academy of Science, Engineering and Technology, 56(32), 446-452.

[69] Yong, H., Juhua, C., Zhenbang, R., Liu, M., & Kang, X. (2006, December). A neural networks approach for software risk analysis. In Data Mining Workshops, 2006. ICDM Workshops 2006. Sixth IEEE International Conference on (pp. 722-725). IEEE.

[70] Rahman, W. N. W. A., Talha, H., Josiah, B., Adamu, L., Liming, W., & Rosli, N. S. M. (2015). Software Quality Assurance–E-commerce Customers Satisfaction in Requirements Engineering Process. International Journal of Software Engineering and Its Applications, 9(3), 57-70.

[71] Hribar, L., Burilovic, A., & Huljenic, D. (2009, June). Implementation of the Software Quality Ranks method in the legacy product development environment. In Telecommunications, 2009. ConTEL 2009. 10th International Conference on(pp. 141-145). IEEE.

[72] Liu, X. F., Kane, G., & Bambroo, M. (2006). An intelligent early warning system for software quality improvement and project management. Journal of Systems and Software, 79(11), 1552-1564.

[73] Dash, R., & Dash, R. (2010). Risk assessment techniques for software development. European journal of scientific research, 42(4), 629-636.

[74] Eckroth, J., & Amoussou, G. A. (2007, March). Improving software quality from the requirements specification. In Proceedings of the 2007 Symposium on Science of Design (pp. 38-39). ACM.

[75] Giesen, J., & Volker, A. (2002). Requirements interdependencies and stakeholders preferences. In Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on (pp. 206-209). IEEE.

[76] Luisa, M., Mariangela, F., & Pierluigi, N. I. (2004). Market research for requirements analysis using linguistic tools. Requirements Engineering, 9(1), 40-56.

[77] Neill, C. J., & Laplante, P. A. (2003). Requirements engineering: the state of the practice. IEEE software, 20(6), 40-45.

[78] Felici, M., Sujan, M. A., & Wimmer, M. (2000). Integration of functional, cognitive and quality requirements. A railways case study. Information and Software Technology, 42(14), 993-1000.

[79] Pressman, R. S. (2005). Software engineering: a practitioner's approach. Palgrave Macmillan.

[80] Robertson, S., & Robertson, J. (2012). Mastering the requirements process: Getting requirements right. Addison-wesley.

[81] Sommerville, I., & Sawyer, P. (1997). Requirements engineering: a good practice guide. John Wiley & Sons, Inc..

[82] ISO, I. (2010). IEEE, Systems and Software Engineering--Vocabulary. IEEE computer society, Piscataway, NJ.

[83] Galin, D. (2004). Software quality assurance: from theory to implementation. Pearson Education India.

[84] Glinz, M. (2007, October). On non-functional requirements. In Requirements Engineering Conference, 2007. RE'07. 15th IEEE International(pp. 21-26). IEEE.

[85] Broy, M. (2015). Rethinking nonfunctional software requirements. Computer, IEEE Computer 48(5), 96-99.
[86] Bassil, Y. (2012). A Simulation Model for the Waterfall Software Development Life Cycle. International Journal of Engineering and Technology, 2(5).
[87] Grady, R. B. (1992). Practical software metrics for project management and process improvement. Prentice-Hall, Inc.
[88] Parasuraman, A., Zeithaml, V. A., & Berry, L. L. (1988). Servqual: A multiple-item scale for measuring consumer perc. Journal of retailing, 64(1), 12.
[89] Chung, L., & do Prado Leite, J. C. S. (2009). On non-functional requirements in software engineering. In Conceptual modeling: Foundations and applications (pp. 363-379). Springer, Berlin, Heidelberg.
[90] Barbacci, M., Klein, M. H., Longstaff, T. A., & Weinstock, C. B. (1995). Quality Attributes (No. Cmu/Sei-95-Tr-021). Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.
[91] Cysneiros, L. M., & do Prado Leite, J. C. S. (2004). Nonfunctional requirements: From elicitation to conceptual models. IEEE transactions on Software engineering, 30(5), 328-350.
[92] Pohl, K. (2010). Requirements engineering: fundamentals, principles, and techniques. Springer Publishing Company, Incorporated.
[93] Broy, M. Rethinking nonfunctional software requirements: A novel approach categorizing system and software requirements. Software Technology, 10.
[94] Glinz, M. (2007, October). On non-functional requirements. In Requirements Engineering Conference, 2007. RE'07. 15th IEEE International(pp. 21-26). IEEE.
[95] Cysneiros, L. M., do Prado Leite, J. C. S., & Neto, J. D. M. S. (2001). A framework for integrating non-functional requirements into conceptual models. Requirements Engineering, 6(2), 97-115.
[96] Chung, L., Nixon, B. A., Yu, E., & Mylopoulos, J. (2012). Non-functional requirements in software engineering (Vol. 5). Springer Science & Business Media.
[97] Matoussi, A., & Laleau, R. (2008). A survey of non-functional requirements in software development process. LACL.
[98] Sommerville, I. Software Engineering (Tenth edition,2016).
[99] El-Far, I. K., & Whittaker, J. A. (2001). Model-Based Software Testing. Encyclopedia of Software Engineering.
[100] Chung, L., & do Prado Leite, J. C. S. (2009). On non-functional requirements in software engineering. In Conceptual modeling: Foundations and applications (pp. 363-379). Springer, Berlin, Heidelberg.
[101] Borg, A., Yong, A., Carlshamre, P., & Sandahl, K. (2003). The bad conscience of requirements engineering: an investigation in real-world treatment of non-functional requirements.
[102] Ameller, D., Ayala, C., Cabot, J., & Franch, X. (2012, September). How do software architects consider non-functional requirements: An exploratory study. In Requirements Engineering Conference (RE), 2012 20th IEEE International (pp. 41-50). IEEE.
[103] R. B. Svensson, T. Gorschek, and B. Regnell. Quality requirements in practice: An interview study in requirements engineering for embedded systems. In Requirements Engineering: Foundation for Software Quality, volume 5512 of Lecture Notes in Computer Science. Springer, 2009.
[104] Svensson, R. B., Gorschek, T., & Regnell, B. (2009, June). Quality requirements in practice: An interview study in requirements engineering for embedded systems. In International Working Conference on Requirements Engineering: Foundation for Software Quality (pp. 218-232). Springer, Berlin, Heidelberg.
[105] So, S. S., Cha, S. D., & Kwon, Y. R. (2002). Empirical evaluation of a fuzzy logic-based software quality prediction model. Fuzzy Sets and Systems, 127(2), 199-208.
[106] Michael, N. (2005). Artificial intelligence a guide to intelligent systems, Pearson Education Limited, 2005.
[107] Marcelloni, F., & Aksit, M. (1997, June). Applying fuzzy logic techniques in object-oriented software development. In European Conference on Object-Oriented Programming (pp. 295-298). Springer, Berlin, Heidelberg.
[108] Al-Jamimi, H. A., & Ahmed, M. (2012, June). Prediction of software maintainability using fuzzy logic. In Software Engineering and Service Science (ICSESS), 2012 IEEE 3rd International Conference on (pp. 702-705). IEEE.
[109] Lin, C. T. (2007, December). New product portfolio selection using fuzzy logic. In Industrial Engineering and Engineering Management, 2007 IEEE International Conference on (pp. 114-118). IEEE.
[110] Lin, C. T. (2000) 'A knowledge-based method for bid/no-bid decision making in project management', Proceedings of PMI Research Conference 2000 Paris, France: pp. 347-355.
[111] Chen, L. H., & Chiou, T. W. (1999). A fuzzy credit-rating approach for commercial loans: a Taiwan case. Omega, 27(4), 407-419.
[112] Martin, C. L., Pasquier, J. L., Yanez, C. M., & Tornes, A. G. (2005, September). Software development effort estimation using fuzzy logic: a case study. In Computer Science, 2005. ENC 2005. Sixth Mexican International Conference on(pp. 113-120). IEEE.
[113] Zadeh, L. A. (2002). From computing with numbers to computing with words: From manipulation of measurements to manipulation of perceptions. In The Dynamics of Judicial Proof(pp. 81-117). Physica, Heidelberg.
[114] Mittal, H., & Bhatia, P. (2009). Software maintainability assessment based on fuzzy logic technique. ACM SIGSOFT Software Engineering Notes, 34(3), 1-5.
[115] Munakata, T., & Jani, Y. (1994). Fuzzy systems: An overview. Communications of the ACM, 37(3), 69-77.
[116] Senior, J., Allison, I. K., & Tepper, J. A. (2007). Automated software quality visualisation using fuzzy logic techniques. Communications of the IIMA, Volume 7, Number 1.
[117] Kaner, C., Bach, J., & Pettichord, B. (2008). Lessons learned in software testing. John Wiley & Sons.
[118] Jan, S. R., Shah, S. T. U., Johar, Z. U., Shah, Y., & Khan, F. (2016). An Innovative Approach to Investigate Various Software Testing Techniques and Strategies. International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET), Print ISSN, 2395-1990.
[119] Pressman, R. S. (2005). Software engineering: a practitioner's approach. Palgrave Macmillan.
[120] El-Sofany, H. F., Taj-Eddin, I. A., El-Hoimal, H., Al-Tourki, T., & Al-Sadoon, A. (2013). Enhancing Software Quality by an SPL Testing based Software Testing. International Journal of Computer Applications, 69(6).
[121] Al Obisat, FM; HS Alrawashdeh; H Altarawneh and M Altarawneh. 2013. "Factors Affecting the Adoption of E-Learning: Jordanian Universities Case Study." Computer Engineering and Intelligent Systems, 4(3), 32-39.
[122] Alzboun, Faried; Haroon Alatarwneh; Mohammad Altarawneh and Farhan M Al Obisat. 2013. "An Assessment for Jordan's E-Government Initiative Projects: A Conceptual Framework." Computer Engineering & Intelligent Systems, 4(3), 1-11.
[123] Masadeh, Shadi R; Nedal Turab and Farhan Obisat. 2012. "A Secure Model for Building E-Learning Systems." Network Security, 2012(1), 17-20.
[124] Obisat, Farhan and Ezz Hattab. 2009. "A Proposed Model for Individualized Learning through Mobile Technologies.", International Journal of Communications 3, no. 1 (2009): 125-132.

AUTHORS PROFILE

Farhan Alebeisat:

(PhD) is an Associate Prof. working in Tafila Technical University (TTU) in the Department of Computer and Information Technology since 2012, my research interests are E-technology, web application and software engineering. I carried out my PhD degree in 2009. I have many published papers in different areas.

Zaid T. Alhalhouli:

(PhD) is an Assistant Professor of computer and Information System at the Department of Computer and Information Technology of the University of Tafila Technical University (Jordan). He was awarded a PhD from Tenaga National University (UNITEN), Malaysia in the beginning of 2015. His research interests include E-learning systems, mobile computing technology, big data and machine learning, Information and knowledge sharing,

Information technology and healthcare, Social Networks, and human-computer interaction.

Tamador I. Alrawashdeh:

She is graduated from Tafila technical university department of Computer and Information Technology and she was worked as a research assistant in 2016-2017.

Tamara E. Alshabatat:

She is graduated from Tafila technical university department of computer and information technology and she was worked as a research assistant in 2016-2017. Now she is studying master of computer science in Mutah university.