Lappeenranta University of Technology
School of Business and Management
Degree Program in Computer Science
Master's Programme in Software Engineering

Razaq Shonubi

SUSTAINABILITY ANALYSIS OF SOFTWARE QUALITY MODEL AND COLLECTION OF REAL TIME METRICS FOR SOFTWARE PROCESS IMPROVEMENT

Examiners: D.Sc. (Tech) Jussi Kasurinen
Professor Kari Smolander

Supervisors: D.Sc. (Tech) Jussi Kasurinen

ABSTRACT

Lappeenranta University of Technology
School of Engineering Science
Software Engineering
Master's Programme in Software Engineering

Razaq Shonubi

Sustainability Analysis of Software Quality Model and Collection of Real Time Metrics for Software Process Improvement

Master's Thesis

2019

51 pages, 8 figures, 5 tables

Examiners: D.Sc. (Tech) Jussi Kasurinen
Professor Kari Smolander

Keywords: Software Quality Model, Software Measurement, Software Sustainability,

Software quality is paramount for every software either private or public use. Over the years, several software quality models have been developed by different researchers. This study aimed at analysing the software quality models available. To this end the study made use of past literatures on software quality models using specific inclusion and exclusion criteria to select the needed literature for the study. This study selected five software quality models. All the five selected models were analysed based on their characteristics and frequency table and bar chart were used for depicting the attributes of each model then comparison was done. At the end of the analysis it was concluded that in as much as no model has all the qualities, the ISO 25010 has proven to cover wider attributes than other software quality models and can be more effective in measuring software quality than the other explored.

ACKNOWLEDGEMENTS

A very special thank you to my supervisor D.Sc. Jussi Kasurinen for his invaluable advice and feedback on my research. I am also very grateful to Prof. Kari Smolander for his support. I gratefully acknowledge the financial support received towards my MSc. (Tech) from Koulutusrahasto.

Nobody has been more important to me in the pursuit of my dream than the members of my family. I would like to thank my parents, whose love and guidance are with me in whatever I pursue. They are my role models.

Most importantly, I wish to thank my loving and supportive wife and my wonderful daughter, who provide unending inspiration.

"The degree to which a product or system can be used by specified goals with effectiveness, efficiency and satisfaction in a specified context."
The ISO 25010 standard

Espoo, 28 January, 2019
Razaq Shonubi

TABLE OF CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
ICT	Information and Communication Technology
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
ISO/IEC 25010	Software engineering –Software Product Quality Requirements and Evaluation standard
MTBF	Mean Time Between Failure
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair
PCMM	People Capability Maturity Model
PQF	Pragmatic Quality Factor
QIP	Quality Improvement Paradigm
SE	Software Engineering
SPI	Software Process Improvement
SQuaRE	Software product Quality Requirements and Evaluation model, ISO/IEC 25010 model
UcSoftC	User Centric Software Certification

1 INTRODUCTION

1.1 Background

Today, software development organizations are passionate about creating and keeping up their software items in a proficient and viable way. In this way, they endeavour towards having efficient and appropriate software development processes. While "in principle", efficient software processes can be accomplished by basically actualizing one of the generally known and acknowledged software development methods, it may not be so "in practice". Because of various organizational settings, certain software development methods can be reasonable for a few organizations yet for nobody else. In addition, software organizations frequently need mastery on and learning of how to distinguish the most reasonable software development methods, and above all, how to adjust them to the organizational setting in a right way.

It is generally realized that there is no "silver bullet" software development method that is reasonable for all (Sommerville, 2011). Software organizations have their very own societies, characters, personalities, custom levels and needs. Along these lines, they can't simply use any predefined existing software development method. They need to adjust it first to their individual needs and settings.

Adjusting a software method is, be that as it may, not an erratic movement. Since in the present exceptionally changing and aggressive business conditions, the organizations, their societies, characters and needs persistently change and advance thus do their software development processes.

Consequently, software development processes should be consistently refined and enhanced, which can be accomplished by executing software process improvement (SPI). SPI goes for the nonstop refinement of software process by methods for understanding the current software processes, distinguishing issues or shortcomings in them and by ceaselessly evolving them (Sommerville, 2011). Its general objective is to build process quality, and along these lines accomplish something like one of the accompanying: 1) improved product quality, 2) lessened development time, or 3) diminished development cost (Paulish & Carleton, 1994),

Software process change can be performed in three different ways, by for example: 1) receiving another software development method, 2) recognizing and disposing of the

shortcomings and defects of the current software processes, or 3) by benchmarking the process against and transforming it towards existing reference models. The reception of another software development method is regularly done in an organization particular manner since there are no standard procedural guidelines for it. The two different approaches to enhance software process, be that as it may, are bolstered by instrumental, procedural, and at times, by organizational SPI rules. Distinguishing and disposing of the shortcomings of the current software processes can be bolstered by inductive SPI methodologies, for example, quality improvement paradigm (QIP) (Basili, 1985), and six sigma (Bhote, 1989). Benchmarking the software process against existing reference models is driven by process institutionalization and dependent on the execution of endorsed processes and best practices upheld by a specific reference model. The best known reference models in software engineering industry are process capability and maturity models, for example, CMMI (capability maturity model integration) (CMMI product team, 2002) and spice (software process improvement and capability determination) (Dorling, 1993), and in addition a workforce maturity model, people capability maturity model (P-CMM) (Curtis, et al., 2001). Countless, measures and models is related with SPI. Be that as it may, in spite of the given rules on the most proficient method to institutionalize the processes and how to run SPI endeavours, just a couple of SPI activities succeed. The examination demonstrates that around 70% of the SPI activities come up short (Ferreira & Wazlawick, 2011). The examinations on the SPI achievement factors contend that accomplishment of the SPI activities is reliant on the organizational, social and administrative perspectives, for example, senior management commitment, staff contribution, preparing and coaching, just to specify a couple (Niazi, et al., 2006). The requirement for consolidation of the organizational culture and firmly incorporated in it social issues to the SPI activities has been perceived by various analysts (Heikkila, 2009), Be that as it may, those perspectives are not adequately secured by the current SPI methodologies and models.

1.2 Goals and delimitations

Despite the fact that research on SPI isn't new and countless have been devoted to process improvement, there is lack of studies that analysed software quality models. Many have just given an all encompassing outline of the current SPI domain that gives adequate inclusion

of organizational, social and managerial aspects of SPI (Mcfeeley, 1996), (ISO/IEC, 2004), (Basili, 1985). A couple of SPI models, for example, P-CMM (Curtis, et al., 2001) and six sigma (Bhote, 1989) address a portion of the organizational and social aspects, however don't cover them adequately. The little information on hierarchical, social and administrative parts of SPI that is secured by a couple of SPI approaches is for the most part scattered over the domain. To the information of the author of this postulation, there is only one model that gives a general outline of the SPI field and, to some degree, thinks about authoritative, social and administrative parts of SPI (Zahran, 1998). The model anyway is obsolete.

Research question: The main objective of this research is to conduct a sustainable analysis of software quality models. To achieve this objective, the study asks the following questions.

1. What are the different software quality models available?
2. What are the differences between the software quality models selected?
3. What are the recommendations for software process improvement based on the analysed software quality models?

The study covers only the models used for software development. In-depth study about software process improvement covers multiple fields such as software engineering, psychology, organizational science and more, which made it to be too big scope for this research to cover. As a result of this, the research does not go in-depth to have a holistic view of SPI but contribute to understanding the software process improvement. Hence, the focus of this research is to look into analysis of software process models.

1.3 Structure of the thesis

This research work is structured into five (5) chapters. Chapter one covers the background of research, objectives of research and research structure. Chapter two examines different research work relating to concept of software quality model, SPI and significant of software sustainability. Chapter three presents the research methodology, chapter four looks at data analysis, research investigation, discussion and findings and Chapter five draws conclusion, and provides recommendations, including the summary of the dissertation.

2 LITERATURE REVIEW

2.1 Concepts of software quality model

Software quality models are used to measure and manage the software quality. Software quality model possesses quality characteristics to measure the software processes involved during the software development as well as the software product. Quality of software aims at satisfying all stakeholders associated with the software. A software quality model provides basis for specifying quality requirements and assessing quality of software (Azuma, 1996; Fenton, 1994). Different research works have led to the development of various software quality models. Among them are McCall's quality model, Boehm's quality model, Dromey quality model, ISO 9126, ISO 25010, and UcSoftC (Yahaya et al., 2014; Suman and Wadhwa, 2014)

Jamwal (2010) assert that for any business to be successful through the use of technology, the quality of the software in use is an essential element to always consider. This consideration of the software quality is not for quality sake but because many business uses real time control system which any glitches in the system may cause financial loss, loss of life, work failure and delay to the company. Different authors have tried to defined software quality over the years, a recognised one by IEEE (1990) defined software quality as the degree to which a system, component or process meets specified requirement and customer needs. This means that before saying a software is of good or high quality, it must have met or satisfied the customers need, a software that keeps disappointing a customer cannot be said to be of good quality and it is not reliable enough.

Another definition by Pressman (2004) defined software quality as the "conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software." Also Petrash (1999) defined software quality as "the existence of characteristics of a product which can be assigned to requirements.". in al of the definitions, the quality of a software can be examined from its' characteristics, and the characteristics of a software reflect its quality.

2.1.1 McCall quality model (1977)

This is the first of the modern software product quality models introduced in 1977 by Jim McCall (Kitchenham and Pfleeger, 1996). It is to be used by system developer during the system development process. This model integrates user's opinion and system developers concern (Maryoly, 2003). McCall identify three main perspectives: Product Revision, Product Operation and Product Transition in classifying the software quality attribute (IEEE, 1993). The model uses a hierarchy of factors, criteria and metrics to address internal and external product quality (Khan et al., 2006)

2.1.2 Boehm's quality model (1978)

Boehm's quality model begins with the software general utility from various dimensions. Boehm's model presents product quality in a hierarchy with three high level characteristics linked to seven intermediate factors, which are in turn linked to 15 primitive characteristics (Khan et al., 2006). Relative to the McCall quality model, Boehm's model gives more emphasis on the cost-effectiveness of maintenance. Also, it has a wider scope than McCall.

2.1.3 Dromey Quality Model (1995)

This model consists of eight high-level quality attributes including reusability and process maturity. Dromey claims that quality process only exists if it is based on a product quality model (Kitchenham and Pfleeger, 1996)

2.1.4 FURPS quality model

This model was proposed by Grady B. R. and Hewlett Packard Co. The model categorized quality attributes into two different requirements such as Functional Requirements (F) which is defined by expected input and output and Non Functional Requirements. In the FURPS model, U sands for Usability (includes human factors, aesthetic, documentation of user and material of training), R stands for Reliability (includes frequency and severity of failure, recovery to failure, time among failure), P stands for Performance (includes functional requirements) and S stands for Supportability (includes backup, requisite of design, implementation, interface) (Calero et al., 2014)

2.1.5 Systemic quality model

The systemic quality model is developed by identifying the relationship between product-process, efficiency-effectiveness and user-customer to obtain global systemic quality (Calero et al., 2014). Process Effectiveness and Process Efficiency are essential elements of the model. In order for the quality evaluation to be systemic, The Process dimension is incorporated (Maryoly et al., 2003). The model serves as a benchmark that allows their products to evolve and be competitive. The disadvantages of this model is the absence of the user requirements and conformation aspects (Yahaya et al., 2014)

2.1.6 ISO 25010:2011

This model was derived from revised ISO/IEC 9126:1991. It defined quality characteristics and described a software product evaluation process. In ISO 25010 model, Product Quality is the static properties of the model concerns of computer software and dynamic properties systems.

The ISO/IEC 25000 family, is called the SE Software Product Quality Requirements and Evaluation which is abbreviated as (SQuaRE), is a replacement of the ISO/IEC 9126. The part ISO/IEC 25010 introduces a supposed Quality in Use Model. In contrast to the Quality Model obtainable by ISO 9126, this model was mostly prolonged by extra characteristics and modified to its terminology. Clearly, two high-level characteristics were additional to the new model: (1) compatibility aspect, this was added newly to the model and (2) security which previously served as sub-characteristic of functionality (ISO/IEC 25010:2011).

2.1.7 UcSoftC quality model

The user centric software certification (UcSoftC) model is claimed to fulfil the requirement of organization according to demands and constraints in software product quality and assessment. This model supports the user centric approach in assessing and certifying the software (Deraman et al., 2014). UcSoftC model enables software users to assess and certify their own products in their own environment with tailored and chosen attributes based on the organization's requirements and expectations.

2.1.8 PQF

Pragmatic quality factor (PQF) is a software quality model which can be used in assessment of software operating in certain environment involving the user. This model consists of four main components as behavioural attributes, impact attribute, responsibility and classification of attributes and weight factors. This model is an improvement of the earliest models of quality such as McCall, Boehm, FURPS. This model has shown that the immeasurable characteristics can be measured indirectly using measures and metrics approach. However, this model has been validated by research (Yahaya et al., 2014)

2.2 Concept of software quality

Different meanings have been reported in literature regarding software quality as it means different things to different people. Software quality is defined as the design or effort that ensures customer satisfaction by simply meeting customers' expectations. ISO 9126 defined quality as a set of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs. Also, software quality is defined as conformance to requirements, standards, and characteristics expected of all professionally developed software. Software quality can also be defined as the degree to which software possesses a desired combination of attributes which are well defined (IEEE, 1990). Quality is often considered as a property of a product in relation to attributes which can be designed into a product or to be evaluated to ensure quality (Bevan, 1995). Also, quality comprises all characteristics and substantial features of a product or an activity related to the satisfying of given requirements (Khan et al., 2006)

Software quality is categorized into attributes that satisfies operating requirements and customers' demands. Software quality attributes include maintainability, usability and portability readability and testability (Fenton and Neil, 1999). According to the International Standard Organization (ISO), software quality attributes are relative to participant perspectives and cannot be assessed by absolute value. This is because different users are dealing with the same software from different perspectives. In the opinion of (Shumskas 1992), it is hard to obtain software that can totally satisfy the quality requirements. However, satisfying the software quality requirement of different stakeholders can be achieved by studying customer wants and needs, gathering customer requirements and measuring

customer satisfaction (Lin and Shao, 2000). Also, Customer acceptance coupled with the time consumed and cost burden are factors to be considered while weighting software quality requirement (Gentleman, 1996). Investigating software quality features should not be left in the hands of requirements engineers and developers as the safer approach is to developing a questionnaire on the key elements in the software quality features which provides requirements' analysts with a knowledge repository to ask the right questions and capture precise software quality requirements information.

Customers determine product quality in terms of their interaction with the final product. Such interaction is represented through their perceived satisfaction that is obtained by a combination of product's functions whether present or absent and the product's non-functional qualities. In addition, the assessment of software quality is different among stakeholders (Atoum and Bong, 2014). Users are particular about the whole product while it is operational, while the developers are interested in developing quality software (Somerville, 2011).

Table 1. Different Factors Considered by Different Software. (Nermine & Mona, 2013)

Software application features	**Factors to be considered**
Software requiring long lifetime period	Portability, Maintainability
Real time system	Reliability, Efficiency
Software needed to be applied in several environments	Portability
Banking system	Reliability, Functionality

Table 1 shows the different factors to be considered in different software. Software are not of same kind, in as much as they all serve the purpose of easing work and solving problems, they follow different pattern and features. A software that is required to be used for a long time need to be portable and easy to maintain. This is because maintainability of the software

enables its smooth running without failure. A real time system which serves a companies' customers directly needs to be efficient and reliable as no room or time to waste correcting failure. A real time system represents the organization and failure in the system means failure by the organization. This has caused financial loss to many organizations in the past.

2.3 Significance of software quality

The quality of software product has an effect on the degree of success or failure of a software development program. Factors that determine software quality are human factor, software demand, shortage in quality management, testing control, traditional custom adopted through the software developers, the development specifications and inadequacy in development tools (Pedrycz and Succi, 2005). Poor software quality causes system failures, higher maintenance costs, longer cycle times, customer dissatisfaction, lower profits and loss of market share (Gopal et al, 2002). However, poor software quality may be associated with unfamiliarity with customer's quality requirement (Dongping and Youcheng, 2001) or by changes to the requirements. It has been reported that software quality has become the key competition for software product market. When software quality meets a customer's expectation, no negative consequences is produced and the customer is happy with the product (Denning, 1992). The quality model helps to provide a base for assessing and measuring characteristics like size, complexity, performance and quality. The development of software with goal to acquire quality can avoid the waste of time and effort and to avoid the negative assessment by users towards the application (Patton, 2007). Quality models serve as basis of modelling and coding standards or guidelines to provide direct recommendations on system implementation and thus constitute constructive approaches to achieve high software quality (Dromey, 1995)

To avoid issues associated with poor software quality, it is advocated that software engineers should equip themselves with the relevant skills. Also data should be collated from all stakeholders to ascertain their needs and expectations. Software quality assurance ensures IT-Business alignment and resolves the conflict of interest and preferences between the stakeholders. Unfortunately, it has been reported that many design organizations do not have the necessary human factors specialists to analyse the users' tasks and this might cause the organization to develop the wrong user interface. Users involvement is critical in software quality (Atoum and Bong, 2014). It has been reported that some software quality models do

not fit to measure software quality from user point of view (Fenton, 1994). It is therefore advocated that such quality models should be revised and extended to include users in the process (Denning, 1992). Software quality model should be utilized throughout the software lifecycle which incorporates all the perspectives of quality model and different stakeholders (Barney and Wohlin, 2009).

2.4 Concept of sustainability

Sustainability relates to environmental, social, and economic aspects of software development and the usage of software system (Calero et al., 2013; Penzenstadler and Femmer, 2012). Sustainable development involves meeting the needs of the present without compromising the ability of future generations to meet their own needs (UNWCED, 1987). Sustainable software also can be defined as software whose direct and indirect negative impacts on economy, society, human beings, and the environment resulting from development, deployment, and usage of the software is minimal and has a positive effect on sustainable development (Dick et al., 2000). From the perceptive of environmental, economic and social point of view, we can define sustainable software engineering as the art of developing sustainable software through a sustainable software engineering process. According to (Lami et al. 2012), sustainable software process is a software process that meets its sustainability objectives, expressed in terms of direct and indirect impacts on economy, society, human beings, and environment. (Amsel et al. 2011) opined that sustainable software engineering should develop software that meets the needs of users while reducing environmental impacts. In addition, (Koziolek 2011) define sustainability of software systems as long living system that should last for more than 15 years and can be cost-efficiently maintained and evolved over its entire life-cycle. According to (Seacord et al. 2013), software sustainability is the ability to modify a software system based on customer needs. This shows that sustainability is concerned with the quality of conforming to user specification. Sustainability as a system must achieve fairness in distribution and opportunity, adequate provision of social services, including health and education, gender equity, and political accountability and participation. This relates to how well a system can cater for different user needs irrespective of their condition. However, (Seacord et al. 2013) indicated that planning and management of software sustainment is impaired by a lack of consistently applied, practical measures. For a software product to attain the goal of

sustainability, it must be developed in such a way that the negative and positive impacts on sustainable development are continuously assessed, documented, and used for further optimization of the software product.

2.5 Sustainability analysis of software quality model

The global impacts of development, deployment and usage of the software on economy, society, human beings and environment have been reported (Dick et al., 2013). The major areas of concern are energy efficiency (Johann et al, 2011) and global carbon dioxide emissions. The most important objective of any engineering activity is to produce high quality product with limited resources and time. Overall, the information and communication technology (ICT) sector contributes around 2% of the global CO_2 emissions. It is also accountable for approximately 8% of the European Union's (EU) electricity consumption and 2% of the carbon emissions from ICT devices and services (Calero and Piattini, 2015). It is therefore important to look how to reduce the impact of ICT on the environment and how sustainability can be incorporated better into the software development lifecycle. Therefore, the meanings and integration of sustainability should cover the five dimensions of software sustainability (Blevis et al., 2017): economic, social, individual, environmental and technical (Penzenstadler and Femmer, 2013). The economic, technical and business dimensions are now core aspects of fundamental values in companies embracing sustainable development (Jeanette and Beloff, 2002). An area that has received less attention is the social dimension. It entails the well-being of the software users' community and developers and is about changing the human mind set and designing their perceptions and experiences of sustainability. Designers must learn to patch together a series of disparate sustainability understandings, and frameworks in order to address the different dimensions of sustainability (Shedroff, 2018). An alternative design solution is based on the sustainable design practices that use the least energy over ICT's life cycle (Bibri, 2018). (Käfer 2009) also presented conceptual and architectural issues concerning software energy consumption as well as ideas for incorporating energy efficiency issues into software projects. Designers of software technology are responsible for the long term consequences of software designs. (Agarwal et al. 2012) analyzed and discussed the possibilities and benefits of green software. One of their findings is that more efficient algorithms will support sustainability. The research presented methods to develop software in a sustainable way and listed some further

environmentally friendly best practices for the development and implementation of software systems. Based on the life cycle of software, (Taina 2010) proposed a method to calculate software's carbon footprint by analysing the impacts of each software development phase. The resulting carbon footprint is mainly influenced by the development phase, but also by the way it is delivered and how it will be used by the customers. (Erdélyi 2013) studied the lifecycle activities of software development with focus on environmental protection by proposing a formula to calculate software waste to encourage the development of green software. In addition, (Albertao et al. 2010) proposed software engineering metrics based on software quality like reusability, portability, supportability, performance as a way for measuring the sustainability performance of software projects. One of the most referenced model for developing and measuring sustainable software is the Greensoft Model by (Naumann et al. 2011). It is a conceptual reference model for "Green Software." The Greensoft model has the objective to support software developers, administrators, and software users in creating, maintaining, and using software in a more sustainable way. However, it lacks the clarity and practical examples of how this model can be implemented for software system development. The key to measuring sustainability of software system requires quantifiable variables that can be applied to all sustainability dimensions in relation to software system development.

Few tools are available to find out solutions to the Green and sustainable software engineering (Johann et al., 2011). Green Power Indicator visualizes when a webpage is hosted with Green electricity and depicts the power quality. This tool is an add-on for Mozilla Fire fox. To analyze the applications energy efficiency it is recommended to use a range of tools which are available to address power related frameworks, optimizations and measurement for example Intel Power Checker, Perfmon, PwrTest, Windows Event Viewer, Windows ETW (Event Tracing for Windows), Power Informer, Power Top, and Battery life toolkit etc. GREEN TRACKER measures the energy consumption of software. Soft Watt (Dick et al., 2013) is a tool to estimate power used by OS and applications. The Java tool called Workload Simulator can record a given flow of operations and execute it a given number of times for a given number of simultaneous users this tool will help to identify the performance of an application. See Figure 1 for GREENSOFT quality model for sustainable software

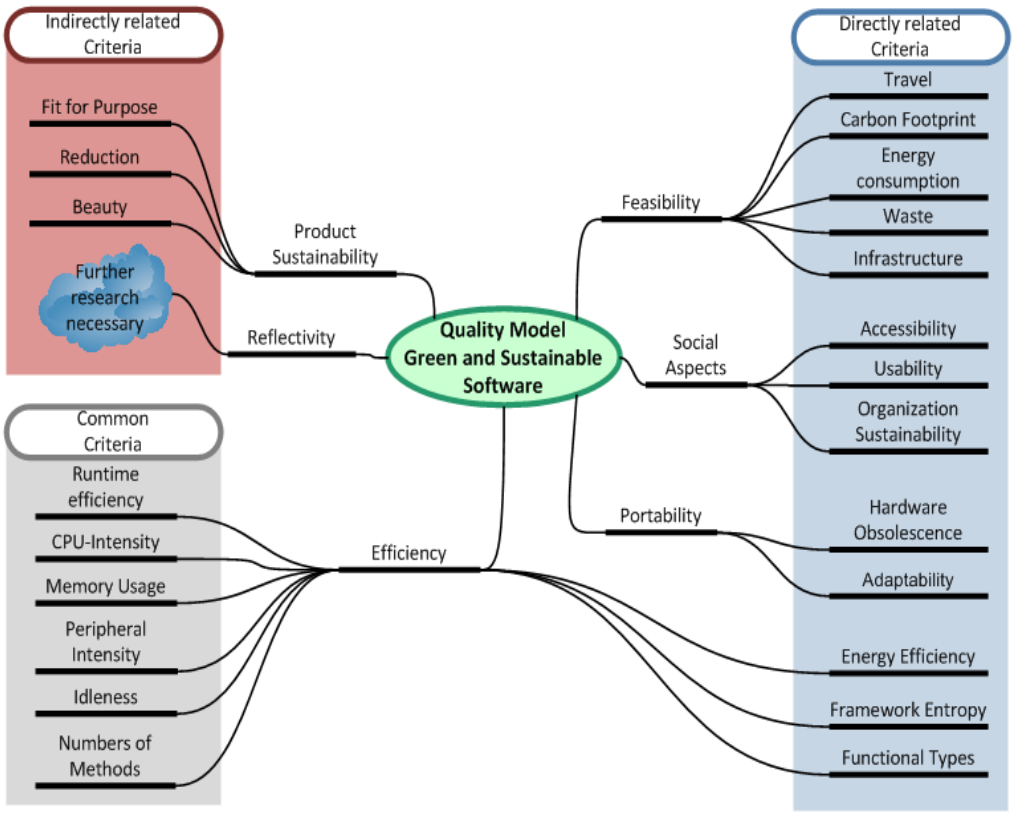

Figure 1. Quality model for sustainable software **(**GREENSOFT, 2014)

2.6 Concept of software quality metrics

Metrics have appeared for capturing software attributes in a quantitative way. Measuring the software quality characteristics defined in the quality model involves using software metrics. Software metrics are the units to measure the attributes of the software process and its developmental activities. For instance, metrics such as Line of Code (LOC), Token Count (TC) and Functional Process Count (FPC) measured and evaluated to determine the quality and sustainability of the software (Bourque and Fairley, 2014). The discipline of software metrics entails identifying various attributes that need to be measured and determining how to measure them in developing quality software. Quality metrics must be utilized and tightly coupled with the software development process. Software quality metric is a tool of measurement whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality (Akingbehin, 2005).

Metric can be defined as a quantitative measure of software or processes for a given attribute to assess quality (Yahaya et al., 2014)

Software metrics can be classified into three categories: product metrics, process metrics, and project metrics. Product metrics describe the characteristics of the product such as size, complexity, design features, performance, and quality level. Process metrics can be used to improve software development and maintenance. Project metrics describe the project characteristics and execution. Software metric is the most suitable methodology to assess the software quality characteristics. Software measurement is concerned with deriving a numeric value or profile for an attribute of a software component, system or process to draw conclusion about the software quality or asses the effectiveness of software processes, tools, and methods (Kitchenham and Pfleeger, 1996). Thus, optimal metrics should be: simple, objective, easily obtainable, valid and robust.

2.6.1 MTTF (mean time to failure)

The MTTF is the mean time for which a component is expected to be operational. MTTF is the average time between two successive failures, observed over a large number of failures. To measure MTTF, we can record the failure data for n failures. An MTTF of 500 means that one failure can be expected every 500 time units. The time units are totally dependent on the system and it can even be specified in the number of transactions, as is the case of database query systems. MTTF is relevant for systems with long transactions, i.e., where system processing takes a long time. We expect MTTF to be longer than average transaction length (Alaa & Mustafa, 2017).

2.6.2 MTTR (mean time to repair)

MTTR is a factor expressing the mean active corrective maintenance time required to restore an item to an expected performance level. This includes activities like troubleshooting, dismantling, replacement, restoration, functional testing, but shall not include waiting times for resources. In software, MTTR (Mean time to Repair) measures the average time it takes to track the errors causing the failure and then to fix them. Informally it also measures the down time of a particular system (Alaa & Mustafa, 2017).

2.6.3 MTBF (mean time between failure)

MTTF and MTTR can be combined to get the MTBF metric: MTBF = MTTF + MTTR. In this case, time measurements are real time and not the execution time as in MTTF. Thus, MTBF of 300 hours indicates that once a failure occurs, the next failure is expected after 300 hours (Alaa & Mustafa, 2017).

2.7 Method of measuring software quality

Software quality evaluation standard involves the following for software evaluation (Pressman, 1997): measuring software quality during development process, revealing current software condition, predicting the following-up as well as the provision of the powerful means to do so for the buyer, the developer and the evaluator. The principle of software metrics is to make assessments throughout the software life cycle, to measure whether the software quality needs are being met. Therefore, it is important to consider carefully what software engineering data to collect for the purpose of quality measurement and decision making. The data must be based on well-defined metrics and models. At the project level, the primary focus is to measure errors and defects and derive relevant metrics such as requirement or design errors per function point, errors uncovered per review hour, errors per thousand lines of code. The software engineering community has adopted an iterative approach to software development in form of Scrum (Schwaber, and Beedle, 2002), XP (Beck, 1999) and other agile (Cockburn, 2001) methods. These promote fast cycle user interaction and development process to keep the effort focused on customer needs based on fast customer feedback either interactively or through analysis of service use behaviour. The start-up community has adopted a similar approach and commonly uses the lean start-up cycle (Ries, 2011) to evaluate the hypothesis of customer needs using the build measure learn cycle, which is repeated to improve customer acceptability of the offering and the business value of the start-up. The common theme of these approaches is that instead of trying to estimate or predict the value in advance, try to shorten the cycle time from development to actual customer feedback, which indicates the value of the software in use. That is, from the software engineering (SE) perspective, the speed of feedback received from users is the best indicator of the value of the newly created software. This indicates that shortening the feedback cycle would drive the SE process towards faster reaction on

customer value and higher value creation. The flow of new features through a SE process can be measured at various points in time with an aim to reduce delay between points to reduce cycle time. Data collection and analysis, which yields intelligence about the project and the development process, is vital for business success. Validation should be performed concurrently with software development and data collection, based on interviews with those people supplying the data. The actual collection process can take several basic formats such as reporting forms, interviews, and automatic collection using the computer system. It is more important that the information extracted from the data be focused, accurate, and useful

Table 2. Software Metrics to Assess Software Quality (Sivagami et al., 2013)

Software Quality Characteristics	**Sub-Characteristics**	**Proposed Metrics**
Functionality	Suitability	No. of undetected functions during system testing / total no. of functions in the specification
	Accurateness	Total number of variations in the obtained results to the expected results for a set of given inputs
	Compliance	No. of satisfied requirements / Total no. of requirements
	Security	No. of attempts (succeeded /failed) / Total no. of attempts
2-Portability	Adaptability	No. of platforms to which the software is applicable / Total no. of platforms
	Installability	Time required to install the s/w in the specific environment / standard installation time
	Conformance	No of (Standards/ conventions) performed to which the software adheres / Total no. of (standards / conventions)
	Replaceability	Adaptability + Installability

2.8 Significance of software metrics

Metrics provide an insight into the effectiveness of the quality assurance activities. Software Metrics are essential in software engineering for measuring the software complexity,

estimating size, quality and project efforts (Amit and Sanjay, 2012). (Douglas 1993) believes that the software quality metrics can be used to detect and remove problems with the software processes. Application of software metrics acts as an improvement driver for Capability Maturity Model. (CMM). Over collection of software engineering data is quite common when people start to measure software without a specification of metrics and models. The interpretation of the desirable properties of a software product in quantitative terms is an important part of the engineering activity in the modern world. Because once the product is released, it is no longer in a controlled test situation but instead in-practice with different users. Measuring quality can formulate baselines, estimate quality and observe enhancement. The advantage of software metrics is it provides a quantitative basis in the assessment of software quality. Thus it reduces the subjectivity and makes the software quality more visible. Metrics plays significant role contributing to end product quality. Software quality assessment is expected to contribute and assist developer and tester to identify and correct the defect.

3 METHODOLOGY

This chapter presents the various methods that were used in the collection of data, processing and analysis. It also provides the descriptive information on the methods and instruments of data collections, processing and analysing.

3.1 Research design

There are three main research approaches in social science research namely: Quantitative, Qualitative and Mixed method (Tashakkori & Teddlie, 2003). In qualitative research approach, the researcher depends on the views and opinion of the participants by asking questions that are broad and general in nature, collecting of data that are mainly words or text, describe and analyse the words collected for theme and make inquiry in biased manner (Tashakkori & Teddlie, 2003). This research approach discovers and understands views and thoughts of the respondents which means it explores purpose and reality. This approach represents the world into interviews, photographs and recordings. Qualitative research engages an interpretive, naturalistic approach to the world. (Denzin and Lincoln, 2005) contends that qualitative researchers study things in their natural settings, trying to make sense of, or interpret phenomena in terms of the meanings people bring to them. The advantage of this approach is that it does not need a strict design plan to begin. Also qualitative research allows the researcher to gain more detailed and comprehensive written description or visual evidence such as photograph. This type of research approach looks at context and social meaning and how it affects individuals which are advantageous particularly in social science. The disadvantages of this approach are that the researcher interprets the research according to his or her own biased view. Also this research approach is time consuming and can last for months or years (Dillman, 2000; Wallen & Fraenkel, 2001).

The study will be a primarily a qualitative research. The reason for adopting such a methodology is based on the view that a qualitative research can be used “to study almost anything imaginable in the social world” (Kalof, Dan & Dietz, 2008). The preceding thus gives an indication that this study will not be based on an analysis of quantitative data. The research rather intends to analyse software quality models in other to collect real time metrics for software improvement. As (Kalof et al 2008) state that “the tradition of qualitative

research tends to focus on meaning and motivation that underlie understandings of processes in the social world". A similar assertion is found in (David Silverman's 2010) analysis as he suggests that "for qualitative research, detail is found in the precise particulars of such matters as people's understanding and interaction."

Central to this study is a need to understand how software development process can be improved. This essentially gives an indication that there will be a need for an examination of several SPI models and analyses and come out with how it can influence development. (Kalof and Kim 2008) lend their voice to the preceding position by suggesting that central to a qualitative research is an emphasis "on how people make sense of their setting" and "why people think and act as they do" (Kalof *et al* 2008).

3.2 Research Paradigm

Research paradigms are those that relate to the underlying "*knowledge*" or epistemology that guides the research towards the most suitable and appropriate research method for answering the research questions (Orlikowski & Bauroudi, 1991), (Myers, 2009). Therefore, in order to find the most appropriate method for this study, I chose a research paradigm suitable for the research problem and goal. Since the study aimed to add to the solution of an industrial research problem, and the phenomenon being studied was highly impacted by its context, as it is motivated below, the interpretivist paradigm was suggested to guide this study.

Interpretivism assumes that reality can be archived only by means of social constructions. It emphasizes the subjective meaning of the reality and highlights the impact of context and people involved, including the researchers (Orlikowski & Bauroudi, 1991).

According to the interpretivist paradigm, the organizations, groups and social systems cannot exist apart from humans and therefore cannot be defined, characterized and measured in some objective and universal way (Orlikowski & Bauroudi, 1991). By using interpretivism, one could achieve a better and comprehensive understanding/view of the phenomena that is being studied.

Due to high significance of the context and the influence of the organization and people involved in SPI process, the interpretivism was chosen as a leading paradigm for conducting this study. It aids to the understanding of social and contextual situations that influence the SPI process. Moreover, since software organizations are the primary target audience of this

research and the research problem is an existing industry problem, it is important that the research done within this field considers the role of context and real world setting. This is because it has been recognized that practitioners find more useful studies that provide detailed interpretation of a case in specific organizational and social contexts (Walsham, 1995). The lessons and recommendations given for a specific case can be easily applied by other practitioners, if the context allows it.

Interpretivism guides the research towards the methods which recognize the importance of the context and assume that the reality is subjective. In order to understand the meaning of human and social interactions, interpretivists need to engage in the social settings investigated and learn about the context from the participants' perspective (Porta & Keating, 2008), using mainly qualitative research methods such as field studies and action research, since these methods examine humans within their social settings (Orlikowski & Bauroudi, 1991). It, however, also makes use of quantitative data collection methods, if needed (Porta & Keating, 2008). Interpretivists mostly work inductively but can combine both inductive and deductive approaches (Porta & Keating, 2008). Driven by this research goal and objectives, I followed an interpretivist approach and conduct research in line with inductive reasoning.

Moreover, the research would be driven by qualitative research methods, to some extend incorporating quantitative data collection techniques, when required.

3.3 Data Collection and Procedure

Qualitative data for this study is sourced from books, journal articles, news bulletins and internet sources. A case study analysis is further used to interpret and analyse data gathered. This is because it will enable the research design to be more specific, credible, manageable, focused and practical. Essentially, through the adoption of a case study analysis, the researcher is able to glean through an array of materials, and identify substantial resource materials for the study.

Again, the suitability of the case study analysis hinges on the fact that the case study method examines the relationship among all variables in order to provide a thorough understanding of a situation, as much as possible, and avoid the fallacy of generalization (Gacenga et al 2012). The findings of a case study research such as this one, can also lead to the

specialization of the area under investigation, and a further extrapolation and generalization can be made in studies conducted under similar, or the same social, political and economic setting.

I studied three major source types: books and standards, scientific articles and white papers or industrial reports using mainly digital databases and libraries, such as LUT Finna, IEEE, Springer, Libris, and Google Scholar as well as simple internet-based searches. I have explored the state of the art on the software development processes, software development methods and methodologies, software method adoptions, software process improvements, as well as software process improvement tools and models. Finally, I came back to the *Definition of the research focus* step, I confirmed the research goal and objectives, and defined the research strategy to be followed in this research.

3.3.1 Search Strategy

In selecting the study materials needed for the study, articles were searched from the different selected databases using the keyword which returned 25 articles. Filtering the result with the article title and year of publication reduced the results to 16. Applying the inclusion and exclusion criteria limited the result to 7 studies which buttressed the points on the frequency of characteristics posed by the software quality models. Figure 2 shows the adopted search strategy for the research.

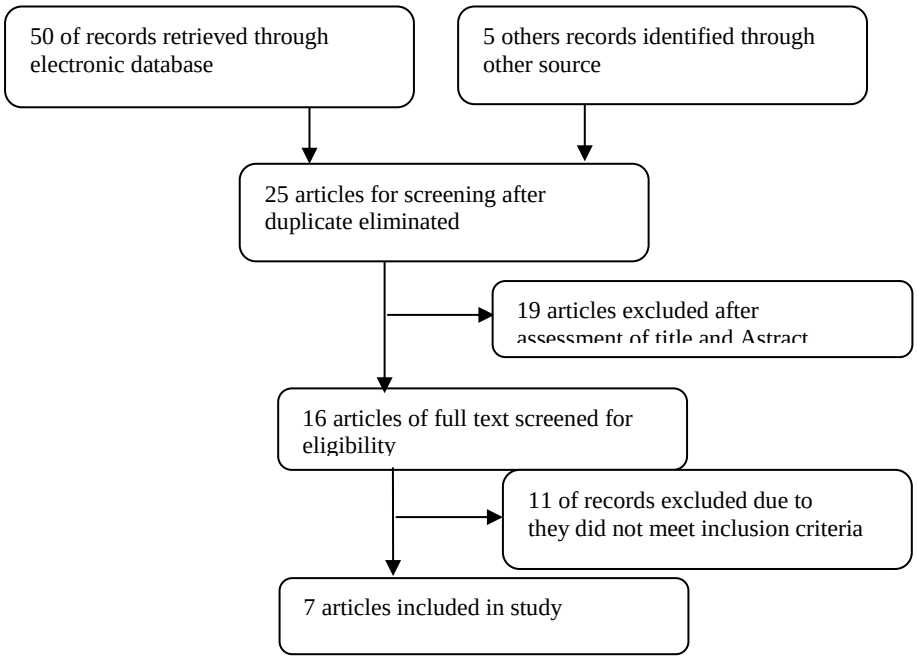

Figure 2. Search Strategy

Table 3. Search Criteria

Language	English
Period	2012 to 2018
Search Keyword	"Analysis of software quality model"
Region	All
Information Resources	IEEE Xplore, EBSCO, Science Direct and Google Scholar

3.3.2 Inclusion/Exclusion Criteria

The following inclusion and exclusion criteria was used to get the needed articles for the study. The inclusion criteria is the criteria used to select the relevant materials for the study. Any article that falls in the inclusion criteria is used while the exclusion criteria is the criteria for which material to be excluded as part of the study.

Table 4. Inclusion and Exclusion Criteria

	Inclusion Criteria	Exclusion Criteria
1	Main aim to examine software quality model	Not conducted to examine software quality model
2	Analyse not less than 5 quality models	Analyse fewer than 5 models
3	Makes comparison of the software quality models	Does not make comparison of the quality models

3.4 Data Analysis

The data gathered from different sources was analysed using a descriptive approach aided by Microsoft Excel. It enabled the research to come out with the final findings of the research. The results from the analysis are presented in frequency tables, bar graphs and pie charts. The data analysis started by collating the frequency of the characteristics of the

selected models. Each model was presented in a column using tally for each of its characteristics. The characteristics are then displayed under each model. When a model has a particular characteristic, it is ticked in its column against the characteristic at the end the total characteristics of each model is written below the table.

4 RESULTS

Several software quality models have been developed over the years; this chapter analyses five selected software quality models that have been developed over years and make comparison of them.

4.1 McCall's Quality Model (1977)

In the Jim McCall et al quality model, it was discovered that there are 11factors out of the associated 23 criteria of a software based on three perspectives which are 1; product revision (the ability to correct errors, system adaptation and undergo general changes) 2; product operation (understability of the product at once, easy to use and giving the required result) 3; product transition (the product being able to adapt to new environment in terms of hardware changes) the model is illustrated in Figure 3 below.

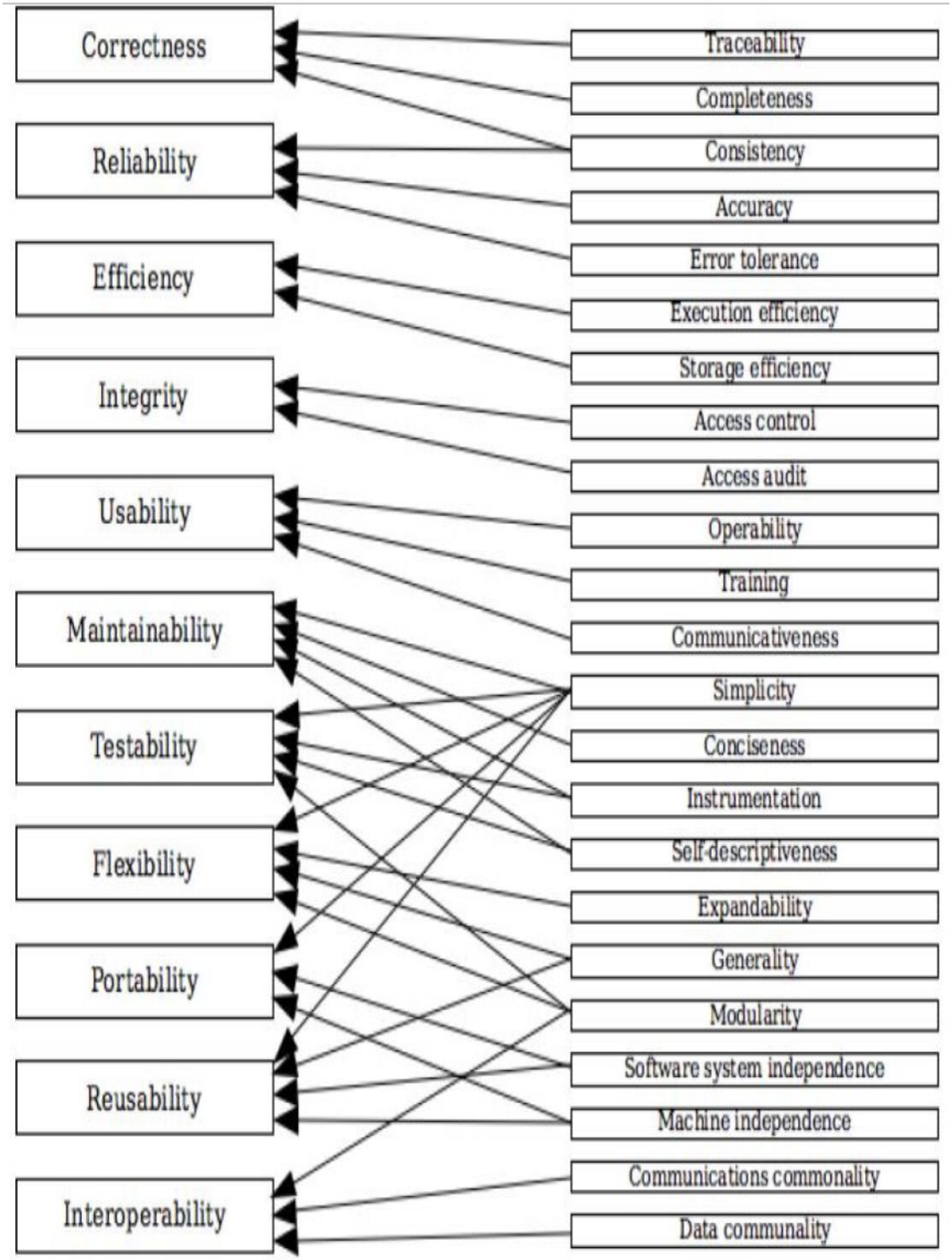

Figure 3. McCall's Quality Model Adapted from Pfleeger (2003) and McCall et al. (1977)

4.2 BOEHM's Quality Model (1978)

Further was the Boehm et al quality model in 1978 which defined the ultimate characteristic of a software as general utility. They arranged characteristics in a hierarchical structure each aiming at contributing to the quality of the entire system. The characteristics at the top level are the requirement of actual use of the software where quality evaluation is made. This include the as-is utility, portability and maintainability. The middle level characteristic is the seven qualities that form the required qualities of a good software which are Understandability, Reliability, Portability, Testability, Efficiency, Usability, and Flexibility. The Evans and Marciniak model is an alternative model have emerged after the McCall model, it defines twelve factors that are grouped into three categories: design, performance and adaptation. See Figure 4 for Boehm quality model adapted from Pfleeger (2003).

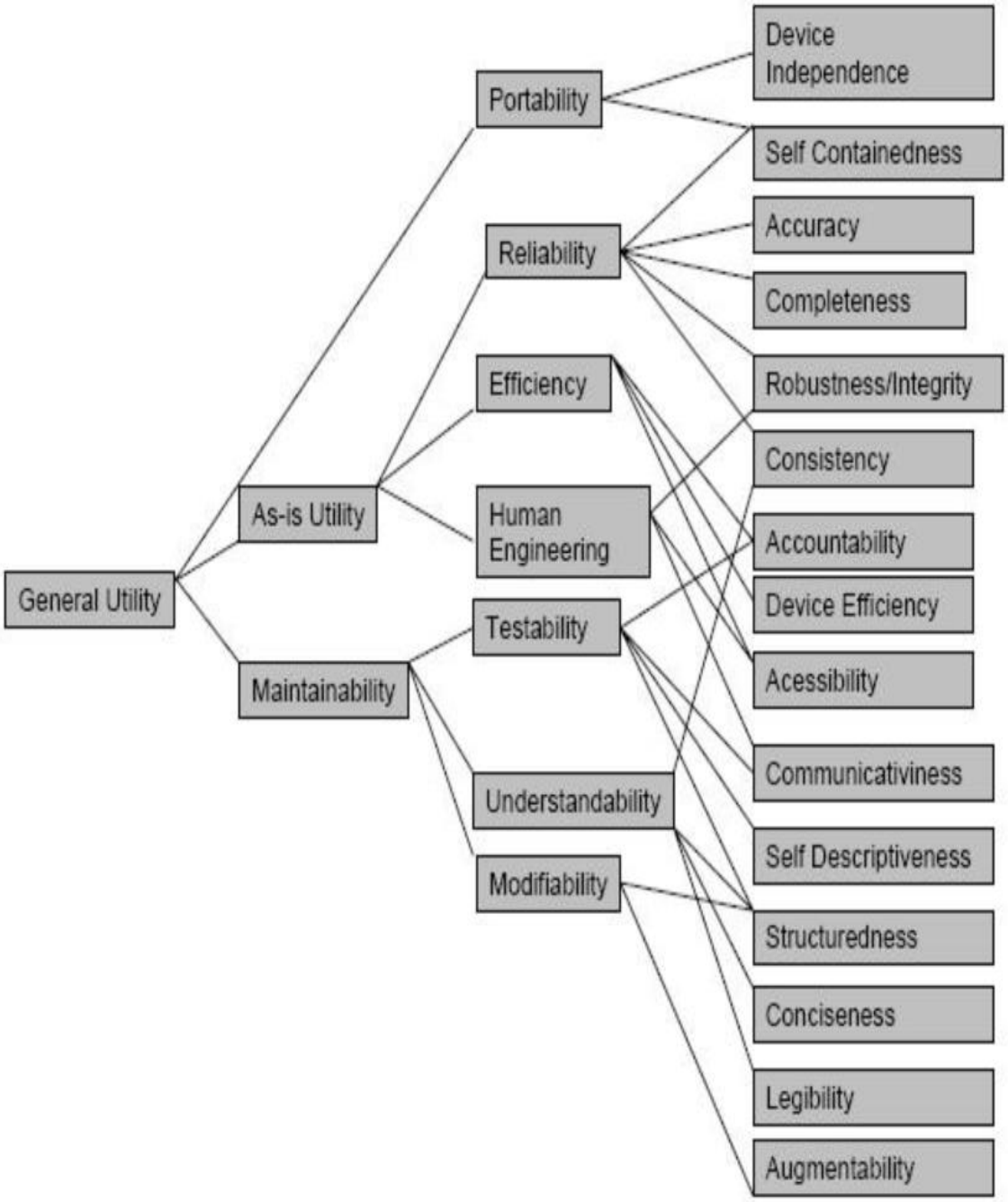

Figure 4. Boehm's quality model Adapted from Pfleeger (2003), Boehm et al. (1976; 1978)

4.3 DROMY's Quality Model (1992)

The third model to be analysed is the Dromy model which was developed in 1992. This model centered on the framework of analyzing requirement, design and implementation. It consists of 3 sub models which are the requirement quality model, the design quality model

and the implementation quality model. The top level characteristics of the implementation model are:

(i) Correctness (checking if some principles are neglected)

(ii) Internal measures (check if the system components are implemented as the intended use)

(iii) Contextual (examines external influence on the utilization of components)

(iv) Descriptive (check how descriptive a component is)

See Figure 5 for more details about Dromey's quality model.

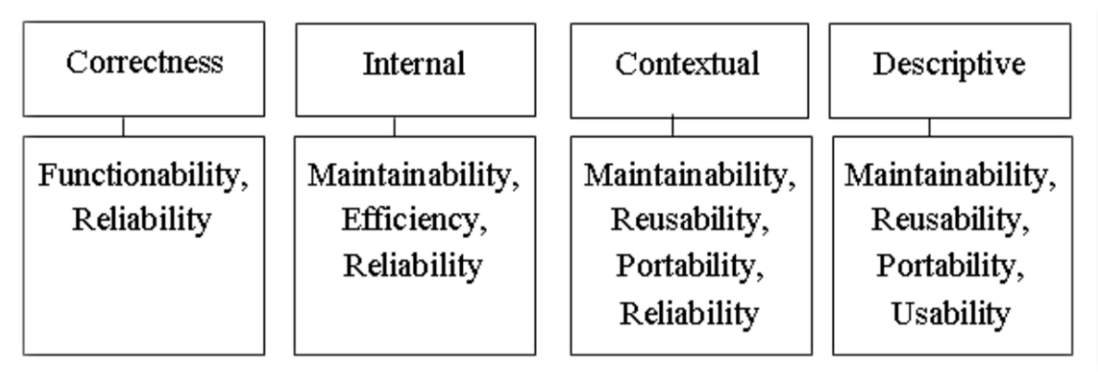

Figure 5. Dromey's Quality Model

4.4 FURPS Quality Model (1992)

This model was presented by Robert Grady at Hewlett Packard. He categorized it into two parts of requirements namely the functional requirements and non functional requirements. The functional requirement defined the input and output while the non functional requirement defined the usability, reliability, performance and supportability (Suman & Wadhwa). See Figure 6

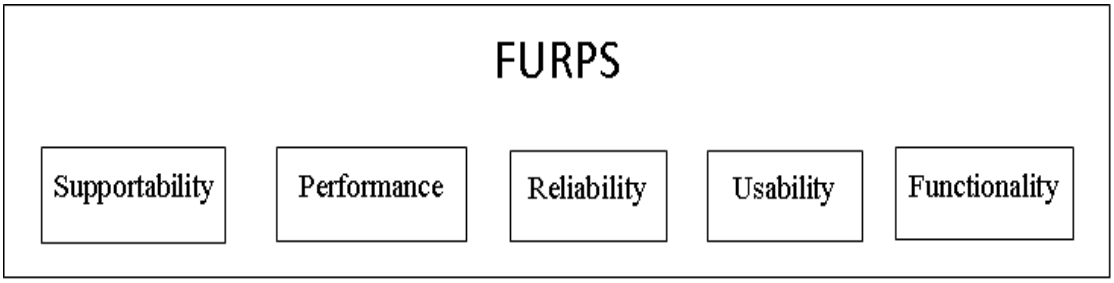

Figure 6. FURPS Model

4.5 ISO 25010 Quality Model (2011)

Another model identified by this study is the ISO 25010 quality model which served as an international standard for measuring quality. This model was coined from the first ISO model ISO 9216. The ISO 25010 is an extension of the ISO 9126 which was coined from the McCall model and it consist of the internal and external quality attributes and quality in-use attributes. The properties and qualities of the system refers to the internal attributes of the system that can be analysed without executing it while the external attributes of the system are the attributes that are examined while in execution.

Quality in-use means the properties the user of the system experience when the system is being used and during maintenance. All the characteristics of the internal and external are Portability, Maintainability, Reliability, Functionality, Efficiency and Usability. The ISO 25010 has additional two characteristics from the previous version 9126 namely; security and compatibility (Suman & Wadhwa, 2014). These new characteristics improves the quality of the ISO model and covers wider scope of characteristics. Figure 7 shows the product quality model according to ISO 25010:2011.

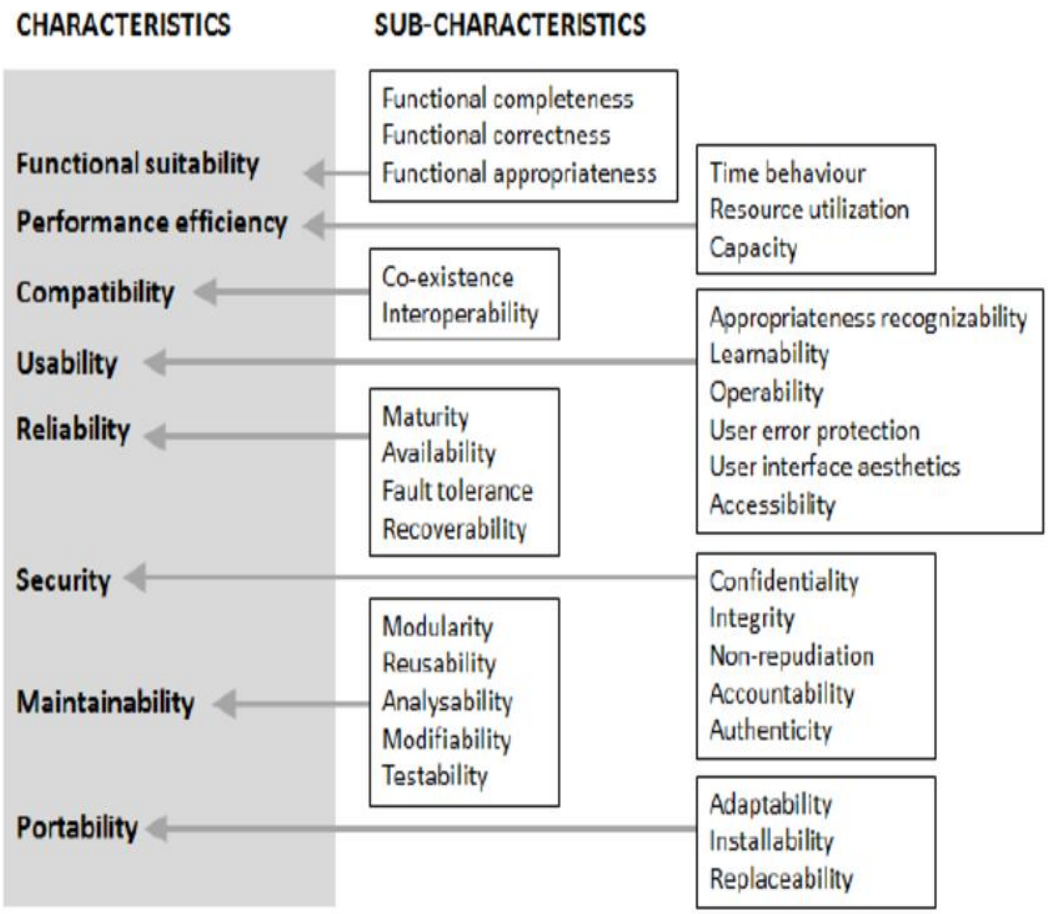

Figure 7. Product Quality Model According to ISO/IEC 25010:2011

Table 5. Comparison of Quality Models

Quality Models Attributes	Boehm	FURPS	Dromey	McCall	ISO 25010
Portability	✓		✓	✓	✓
Reliability	✓	✓	✓	✓	✓
Efficiency	✓		✓	✓	✓
Human engineering	✓				
Testability	✓			✓	✓
Understandability	✓				
Modifiability	✓				
Maintainability			✓	✓	✓

Flexibility				✓	
Reusability			✓	✓	
Interoperability				✓	
Correctness				✓	
Usability		✓	✓	✓	✓
Integrity				✓	
Supportability		✓			
Performance		✓			
Adaptability					✓
Installability					✓
Changeability					✓
Accuracy					✓
Maturity					✓
Suitability					✓
Resource utilization					✓
Analyzability					✓
Stability					✓
Attractiveness					✓
Operability					✓
Functionability		✓	✓		✓
Security					✓
Compatibility					✓
Total Number of Attributes	**7**	**5**	**7**	**11**	**20**

From table 5 above, the 5 quality models have been analysed based on 30 characteristics. Results from table 5 above shows that the first model Boehm posses 7 characteristics which are reliability, portability, efficiency, human engineering, testability, understandability and modifiability. The second model FURPS posses only 5 characteristics namely: reliability, usability, supportability, performance and functionability. The third model Dromey posses 7 characteristics namely; reliability, portability, efficiency, maintainability and reusability, usability and functionability. The fourth model posses 11 characteristics namely; reliability, portability, efficiency, testability, Maintainability, Flexibility, Reusability, Interoperability,

Correctness, Usability, Integrity. The last model ISO 25010 possess 20 characteristics namely; reliability, portability, efficiency, testability, maintainability, usability, Installability, Adaptability, Changeability, Accuracy, Maturity, Suitability, Resource utilization, Analyzability, Stability, Attractiveness, Operability, Functionability, security and compactibility.

In further comparison, the McCall model and the ISO 25010 model have some similarities and differences. The McCall and ISO model both have a quality factor, high level quality characteristics or factors such as usability, reliability, efficiency, maintainability and portability. The differences between these two models are that the ISO model stress more on the characteristics visible to the end users while the McCall consider qualities internal to the system. For example, the characteristics of reusability is internal to the system which the developer has to build this reusability component in the system and not visible to the user. Also, in the McCall model, a single quality criterion can have effect on several quality factors while in ISO model one characteristic affect one quality characteristics. Also, the McCall quality factors is more important to the developer while the ISO model focus on the product. The disadvantage of McCall model is that it does not consider directly on the functionality of software product.

The Boehm model has its strength from the angle where it addresses the contemporary shortcomings of models that automatically and quantitatively evaluate the quality of software. Also, the Boehm model represent the characteristics of the software product hierarchically in order to contribute to the total quality of the software. The drawback of this model is that, the model only contains a diagram without or little suggestion about measuring the quality characteristics.

The FURPS model has its strength is the angle that it considers only the user's requirement and disregard the developer's consideration while its' drawback is that the model fails to take into account some important characteristics such as portability and maintainability. The Dromey model is broad enough to accommodate different system development but suffer from the lack of criteria for measuring software quality.

Figure 8 shows the frequency of each characteristics appearance in the five quality models.

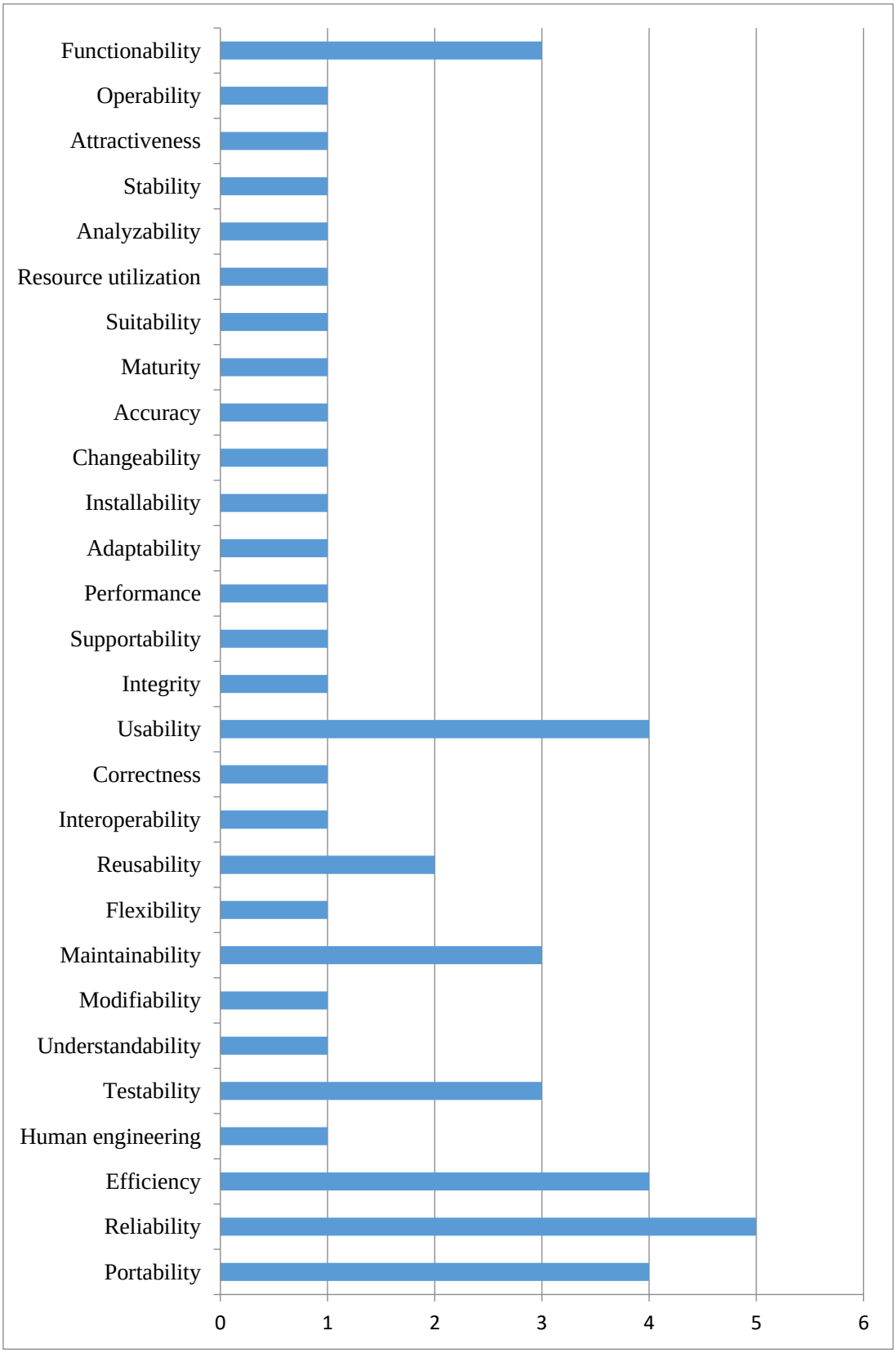

Figure 8. Frequency of Characteristics Appear in the Five Quality Models

From the 30 characteristics, only one characteristic is common to the 5 quality models that is the reliability. Also, there are only three characteristics (efficiency, usability and portability) which belong to 4 models. three characteristics is common only to three quality models that is testability, maintainability, and functionability characteristic. And only two models have reusability in common. And, twenty characteristics (Human engineering, understandability, modifiability, flexibility, interoperability, correctness, integrity, supportability, performance, adaptability, installability, changeability, accuracy, maturity, suitability, resource utilization, analyzability, stability, attractiveness, operability,) are defined in only one quality model.

5 DISCUSSION

This study examined the different software quality models and makes comparison on the selected models. In other for the study to examine the different quality models, the models were analysed by reviewing past literatures on software quality models.

Different software quality models have evolved over the years which has been used in software development and engineering. To get details about this, some selected software models were examined. These software quality models have different characteristics common to all which is reliability and other qualities which cut across some of the models. Out of the models, the ISO 25010 model appears to have the largest number of characteristics for quality software process comprising of 20 characteristics. However, the user has some impact on the quality of software and helpful in measuring the software quality as they interact with the system and spend longer time with the system. In fact, software performance is better rated or determined by the end users.

After the analysis, some of the qualities identified in the models are; reliability, testability, Human engineering, understandability, modifiability, flexibility, interoperability, correctness, integrity, supportability, performance, adaptability, installability, changeablibility, accuracy, maturity, suitability, resource utilization, analysability, stability, attractiveness, operability among others. In all, this study has shown the evolution of the software quality models till date. In as much as no model has all the qualities, the ISO 25010 has proven to cover wider attributes than other software quality models.

ISO/IEC 25010 categorizes the product quality properties into eight characteristics: functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability and portability. With each characteristics being composed of a set of related sub-characteristics. Quality in use attributes are categorized into five characteristics: effectiveness, efficiency, satisfaction, freedom from risk, context coverage (Suman & Wadhwa, 2014).

The definition of quality in use compared to ISO/IEC 9126 also includes the quality of hardware, operating environment and the characteristics of users, tasks and social environment as having an effect on stakeholders. The other models do not take into account quality in use the way ISO 25010 has taken into account of it. Boehm, Dromey, and McCall take into consideration only efficiency which is one of the attributes of quality in use. Also,

all the models take into consideration of another attribute of quality in use which is reliability (Nermine et al, 2013).

The reusability features of the model allows the software to be developed in a design that can be reused by using existing assets in some form within the software product development process; these assets are products and by-products of the software development life cycle and include code, software components, test suites, designs and documentation. Installability of the software is the ability of a software product to be installed in a specific environment. in the case of the ISO model, if the software is installed by the end users, installability will have effect on the suitability and operability of the system (Khan et al, 2006).

This may be due to incorrect installation by user but will operate better when installed by the developer. Another feature is the platform independence which means the software designed based on this model can run some code with little or no modification on multiple platforms. This reduces the stress of starting from scratch when moving to a new platform. Efficiency is the amount of resources required by a program to perform a specific function. This show the amount of effort put into the software development to measure its user satisfaction. Efficiency is important as the main purpose of developing a software is for the users to use it to solve an existing problem or to improve current operation. All these features mentioned improve the sustainability of software product. A software that perform independent is sustainable irrespective of any platform changes. Efficiency, reusability and installability are measures to ascertain if a software product is sustainable (Maryoly et al, 2003).

There is no risk associated to this research only limitations. This study is limited in scope as only five software quality models were examined. If more models are examined there is tendency of having a little shift in the findings. Also covering more models would go a long way in analysing the length and breadth of software quality models. In as much as the study used only five models, the researcher selected the major models and the result does not misrepresent what software quality models are. Also the study is limited to the use of secondary data for data collection if more time is available, meeting face to face with software development companies or software developers would go a long way in giving their opinion on the models they prefer based on their experience in the field. In as much as the study didn't employ interview with developers, literature reviewed took care of that by bringing together views of scholars on the research topic (Koziolek, 2011).

In all it can be concluded from the analysis that models by McCall, Boehm and Dromey are focused on the product perspective of the quality at the expense of other perspective. They can be said to be used in a down to top approach to quality that does not fit for software quality engineering. Also, the MCcall model support all quality perspective and its framework support both the bottom up and top down approaches.

5.1 Recommendation

Based on the results from the study, it is recommended that for software quality engineering, the ISO/25010 seems to be well suited for software engineering process. ISO/IEC 25010 standard isn't picking up the merited consideration from the industry. It is assumed that this is because the market does not request ISO/IEC 25010 certification and ISO/IEC 25010 does not offer any certification whatsoever. In the meantime, unique markets require following of different standards and regulations. ISO/IEC 25010 is predominantly utilized by the research institutions and still not widely utilized in the industry. Another purpose behind lack of support for ISO/IEC 25010 is that the standard does not depict the evaluation process, so the companies miss the data how to utilize this standard. We expect that this procedure issue will be settled with the up and coming generation of ISO/IEC 25010 quality standards SQuaRE.

Recommendation is that giving consulting and certification services to market requested for standards are preferred business opportunity over evaluating software quality utilizing ISO/IEC 25010. An elective business opportunity is utilizing ISO/IEC 25010 standard as a rule for evaluation of particular quality attributes i.e. Ease of use of application software, given that there is an interest from the software producers.

6 CONCLUSIONS

This study examines the software quality models and makes comparison on the selected models. This study has examined and analysed different software quality models. Past literatures on software quality models have been reviewed. All the objectives set for the studies were achieved and questions asked were answered. The first objective set for the study was to examine the different software quality models that exist. The literature reviewed in the chapter two revealed the software quality models that have been in existence till date. Results in chapter four also revealed some of these models. the second question asked in the study was to ascertain the characteristics of the selected software quality models. The results in chapter 4 revealed there exist in total 30 characteristics a quality model can have. Further revealed that no software quality model can have all the 30 characteristics. The last question seeks to find out how the models differ. Result showed that only one characteristic is common to all the 5 selected model and a model can have an attribute which another different model does not have.

REFERENCES

Agarwal M and Kaushal C. (2007), Software Efforts, Quality, and Cycle Time: A Study of CMM Level 5 Projects, IEEE Transaction on Software Engineering, Vol. 33. No.3

Akingbehin K. (2005). "Taguchi-based metrics for software quality," in Proc.Fourth Annual ACIS International Conference on Computer and Information Science, pp. 713–716.

Alaa E.S., & Mostafa E. A. (2017). Redundancy Level Impact of the Mean Time to Failure on Wireless Sensor Network (IJACSA) International Journal of Advanced Computer Science and Applications 8(10), 2017

Albertao F., Jing Xiao, Chunhua Tian, Yu Lu, Kun Qiu Zhang, and Cheng Liu (2010). Measuring the sustainability performance of software projects. In 2010 IEEE 7th International Conference on e-Business Engineering (ICEBE), pages 369{373.

Amit S., Sanjay K. D. (2012), Comparison of Software Quality Metrics for Object-Oriented System, International Journal of Computer Science & Management Studies, Special Issue of Vol. 12

Atoum I. and C. H. Bong (2014), "A framework to predict software 'quality inuse' from software reviews," in Proc. the First International Conference on Advanced Data and Information Engineering, pp. 429–436.

Azuma M. (1996), "Software products evaluation system: Quality models, metrics and processes — International Standards and Japanese practice," Information and Software Technology, vol. 38, no. 3, pp. 145–154

Azuma M. (2004), "Systems engineeringapplying ISO / IEC 9126-1 qualitymodel to quality requirements engineering on critical software department of industrial and management," in Proc. the 3rd International Workshop on Requirements Engineering for HighAssurance Systems, Kyoto, Japan

Barney S. and C. Wohlin (2009), "Software product quality: Ensuring a common goal," Lecture Notes in Computer Science, Springer, vol. 5543, pp. 256–267

Basili, V. R. (1985). Quantitative Evaluation of Software Methodology. Maryland: University of Maryland, TR-1519.

Beck K. (1999), Extreme Programming Explained: Embrace Change.Addison-Wesle

Bibri, M (2009). Sustaining ICT for Sustainability. Available online: https://www.diva-portal.org/smash/get/diva2:833352/FULLTEXT01.pdf (accessed on 8 December, 2017).

Blevis, E.; Preist, C.; Schien, D.; Ho, P (2017). Further Connecting Sustainable Interaction Design with Sustainable Digital Infrastructure Design. In Proceedings of the Workshop on ComputingWithin Limits (LIMITS'17),

Bourque P. and Fairley R.E (2014) editors. Guide to the Software EngineeringBody of Knowledge (SWEBOK(R)): Version 3.0. IEEE Computer Society Press, Los Alamitos, CA, USA, 3rd edition

Calero C., M. Bertoa, and M. Moraga (2013), "A systematic literature review for software sustainability measures," in Proc Green and Sustainable Software (GREENS) 2nd International Workshop, pp. 46–53

Calero C., M. Bertoa, and M. Moraga (2013), "Sustainability and quality: icing on the cake," in Proc 2nd International Workshop on Requirements Engineering for Sustainable Systems (RE4SuSy)

Calero C., M. Moraga, M. Bertoa, and Duboc (2014), "Quality in use and software greenability," CEUR Workshop Pcoceeding, vol. 1216, pp. 28-36

Calero, C.; Bertoa, M.F.; Moraga, M.A (2013). A Systematic Literature Review for Software Sustainability Measures. In Proceedings of the 2013 2nd International Workshop on Green and Sustainable Software (GREENS), San Francisco, CA, USA

Calero, C.; Piattini, M (2015). Introduction to Green in software engineering. Green Softw. Eng., 3–27.

Cockburn A. (2001), Agile Software Development, 1st edition, 256 p. Addison-Wesley Professional.

Curtis, W., Miller, S., & Hefley, W. (2001). People Capability Maturity Model (P-CMM) Version 2.0 (CMU/SEI-2001-MM-001). s.l.: Software Engineering Institute, Carnegie Mellon University.

Denning P. J. (1992), "What is software quality," Communications of the ACM, vol 35, no. 1, pp. 13–15,

Denzin, K., & Lincoln, S. (2005). The Sage Handbook of Qualitative Rese arch (3rd Ed.) London: Sage, 1288.

Deraman A., J. Yahaya, F, Baharom, and A. R. Hamdan (2010), "User-centred software product certification: Theory and practices," International Journal Of Digital Society (IJDS), vol. 1, no. 4, pp. 281–288

Dick M., S. Naumann, and N. Kuhn (2010), "A model and selected instances of green and sustainable software," What Kind of Information Society? Governance, Virtuality, Surveillance, Sustainability, Resilience, vol. 238, pp. 248-259

Dick, M.; Drangmeister, J.; Kern, E.; Naumann, S. (2013), "Green software engineering with agile methods," Green and Sustainable Software (GREENS), 2013 2nd International Workshop vol., no., pp.78,85

Dick, M.; Naumann, S (2010). Enhancing Software Engineering Processes Towards Sustainable Software Product Design. Available online: https://pdfs.semanticscholar.org/98d0/4de0318b252f273e3cf36cc385253542b924.pdf (accessed on 7 December 2018).

Dick M.D., J.Kern, E.Naumann, S. (2013), "Green Software Engineering withAgile Methods," Green and Sustainable Software (GREENS), 2013 2nd International Workshop on , Vol., no., pp.78,85, 20-20

Dilman, D. A. (2000). Mil and internet survey, the tailored design method (2nd ed.) new york;
willey 464.

Dongping F., and L. Youcheng (2001), "Structure Modeling and System Building of Self-adaptation Application Software System," Journal of Computer Engineering and Application, 37(12)

Dorling, A. (1993). SPICE: Software Process Improvement and Capability Determination. J. Software Quality, 2-4, 209-224.

Douglas H. (1993). The Role of the Quality Group in Software Development, Pacific Northwest Software Quality Conference

Dromey R. G. (1995), "A model for software product quality," IEEE Transactions on Software Engineering, Vol. 21, No. 2, pp. 146–162

Dromey R. G. (1996), "Cornering the chimera," IEEE Software, Vol. 13, No. 1, pp. 33–43,

Erdélyi, K (2013). Special Factors of Development of Green Software Supporting Eco Sustainability. In Proceedings of the IEEE 11th International Symposium on Intelligent Systems and Informatics (SISY), Subotica, Serbia, 26–28 September 2013.

Fenton N. (1994) "Software measurement: A necessary scientific basis,"IEEE Transaction on Software Engineering, vol. 20, no. 3, pp. 199–206

Fenton N. E. and M. Neil, (1999), "A critique of software defect prediction models," IEEE Trans. Softw. Eng., 25(5):675–689

Fenton N. E. and S. L. Pfleeger (1998), Software Metrics: A Rigorous and Practical Approach. PWS Publishing Co. Boston, MA, USA. R. S. Pressman, "Software engineering a practitioner's approach", 4th.ed, McGraw-Hill, New York - USA, 1997, pp. 852

Ferreira, M. G., & Wazlawick, R. S. (2011). Software Process Improvement: A organizational change that need to be managed and motivated. World Academy of Science, Engineering and Technology, 50, 296-304.

Gentleman W. M. (1996), “Software quality world-wide: What are the practices in a changing environment,” Proceeding of the sixth international conference on software quality (6ICSQ), Ottawa, Canada

Gopal A., T. Mukhopadhyay, M.S. Krishnan, and D. Goldenson (2002), “The Role of Communication and Processes in Offshore Software Development,” Communications of the ACM, Vol. 45, pp. 193-200

Heikkila, M., (2009). Learning and Organizational Change in SPI Initiatives. Berlin, Springer,
LNBIP 32.

IEEE Standard for a Software Quality Metrics Methodology (1993)

IEEE Standard Glossary of Software Engineering Terminology (1990)

International Organization for Standardization (2005): Software engineering—Software product Quality Requirements and Evaluation (SQuaRE)—Guide to SQuaRE 35.080(ISO/IEC 25000:2005 (E)).

ISO/IEC, (2004). Guidance on ue for process improvement and process capability determination, Geneva: ISO/IEC.

ISO/IEC, ISO/IEC 9126-1:2001(2001) Software Engineering: Product Quality - Quality Model, International Organisation for Standardisation/ International Electro-technical Commission

ISO/IEC25010 (2011) Systems and software engineering–Systems and software Quality Requirements and Evaluation (SQuaRE)–System and software quality models”. In: International Organization for Standardization.

Jamwal, D. (2010). Analysis of software quality models for organizations. International Journal of Latest Trends in Computing, 1(2).

Jeanette S, E.B.; Beloff, B (2002). Use Sustainability Metrics to Guide Decision Making. CEP, 98, 58–63.

Johann, T. Dick, M.; Kern, E.Naumann, S. (2011), "Sustainable Development, Sustainable Software, and Sustainable Software Engineering: An Integrated Approach," Humanities, Science & Engineering Research (SHUSER), 2011 International Symposium on , Vol., no., pp.34,39, 6-7

Kahn, R. Kat I., and Pratt C. (2009). Energy-Aware Storage Benchmarks. ERCIM News

Kandt, R. K. (2003). Ten Steps to Successful Software Process Improvement. Hong Kong, International Computer Software and Application Conference.

Kautz, K., Westergaard Hansen, H. &., Thaysen, K. (2001). Understanding and changing software organizations: An exploration of four perspectives on software process improvements. Scandinavian Journal of Information Systems, 13, 7-20.

Khan R.A, K. Mustafa, and S. I. Ahson (2006), Software Quality Concepts and Practices

Kitchenham B, and S. L. Pfleeger (1996), "Software quality: The elusive target," IEEE Software, 13(1):12–21

Kitchenham B., S. L. Pfleeger, and N. Fenton (1995), "Towards a framework for software validation," IEEE Transactions on Software Engineering, 21(12), 929–944.

Kitchenham B.A, and J. G. Walker (1989), "A quantitative approach to monitoring software development," Software Engineering Journal, 4 (1), 2-13

Koziolek, H. (2011). Sustainability Evaluation of Software Architectures: A Systematic Review. In: Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA and Architecting Critical Systems – ISARCS. pp. 3–12. ACM, New York, NY, USA

Käfer, G., Green S. E. (2009): Ideas for including energy efficiency into your software projects. In:Technical Briefing (TB2). 31st International Conference on Software Engineering, Vancouver
Lin W. and B. Shao (2000), "The relationship between user participation and system success," Information & Management, 37(6), pp. 283-295

Maryoly O., P. María, and R. Teresita (2003), "Construction of a systemic quality model for evaluating a software product," Software Quality Journal, vol. 11, no. 3, pp. 219-242

Mathiassen, L., Ngwenyama, O. K. & Aaen, I. (2005). Managing Change in Software Process Improvement. IEEE Software, 84-91.

McFeeley, B. (1996). IDEAL: A User's Guide for Software Process Improvement, Pittsburgh, Pennsylvania: Software Engineering Institute, Carnegie Mellon University.

Muller, S. D., Mathiassen, L. & Balshoj, H. H. (2008). Organizational Change Perspectives on Software Process Improvement, s.l.: Informatics, Research group.

Muller, S. D., Mathiassen, L. & Balshoj, H.H. (2010). Software Process Improvement as Organizational Change: A metaphori cal analysis of the literature. The Journal of Systems and Software, 83, 2128-2146.

Naumann S., M. Dick, E. Kern, and T. Johann (2011). The GREENSOFT Model: A reference model for green and sustainable software and its engineering. Sustainable Computing: Informatics and Systems, 1(4):294–304,

Nermine M. Khalifa, Mona M. Abd Elghany (2013). Conceptualizing Quality in Software Industry. GSTF Journal on Computing (JoC), Vol. 3 No. 3,

Niazi, M., Wilson, D., & Zowdhi, D., (2006). Implementing software process improvement
initiatives: An Empirical study. s.l., Proc. PROFES 2006.

Nina, W.; Patricia, L.; Francesco, O. (2017). Sustainability in Software Engineering. 2017. Available online: http: //dl.ifip.org/db/conf/ifip6-3/sustainit2017/08_P08_S21_SustainIT2017.pdf (accessed on 8 December 2018).

Orlikowski, W. & Bauroudi, J., 1991. Studying Information Technology in Organizations: Research Approaches and Assumptions, s.l.: The Institute of management Sciences.

Patton J. (2007), "Understanding user centricity," IEEE Software, vol. 24, no. 6, pp. 9–11

Pedrycz W., and Succi G. (2005), "Genetic granular classifiers in modeling software quality," Journal of Systems and Software, 76(3):277-285

Penzenstadler, B.; Femmer, H. A (2013) Generic Model for Sustainability with Process- and Product-Specific Instances. In Proceedings of the 2013 workshop on Green in/by software engineering (GIBSE'13), Fukuoka, Japan, 26 March 2013; ACM: New York, NY, USA, pp. 3–7.

Porta, D. & Keating, M., 2008. Approaches and Methodologies in the Social Sciences. Florence: European University Institute.

Pressman R.S. (1997) Software engineering, a practitioner's approach, Fourth Edition, McGraw-Hill Press

Raffo D. M., W. Harrison and J. Vandeville (2002), "Software Process Decision support: making process tradeoffsusing a hybrid metrics, modeling and utility framework" Proceedings of the 14th ACM international conference on Software engineering and knowledge engineering, pp. 803-809.

Research Project GREENSOFT (2014): Website: Research Project Green Software Engineering— Downloads (2014) http://www.green-software-engineering.de/en/downloads.html

Ries E. (2011), The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. Crown Publishing Group.

Sahota, M. (2012). An agile adoption and transformation survival guide: Working with organizational culture.s.l.: InfoQ.

Santa B (2017), CA, USA, 22–24 June 2017; ACM/IEEE: New York, NY, USA; pp. 71–83.

Schwaber K., and M. Beedle (2002), Agile Software Development with SCRUM, Prentice Hall,

Seacord, R.; Elm, J.; Goethert, W.; Lewis, G.A.; Plakosh, D.; Robert, J.; Wrage, L.; Lindvall, M (2003). Measuring Software Sustainability. In Proceedings of the International Conference on Software Maintenance, Amsterdam, The Netherlands

Shedroff, N (2018). Design is the Problem: The Future of Design Must be Sustainable. p. 582. Available online: https://designethosandaction.files.wordpress.com/2015/01/design-is-the-problem.pdf (accessed on 28 November 2018).

Shumskas A. F. (1992), Software risk mitigation total quality management for software, New York: G.G.a. J.I.M. Schulmeyer, Van Nostrand Reinhold, (pp.190–220)

Sivagami S. S., G.Sri Arjuna Ragini, T.Chandrakumar and S.Parthasarathy (2013) Software Quality Metrics for CRM: A Quantitative Approach. International Journal of ComputerApplications (0975 – 8887)

Sommerville, I. (2011). Software Engineering, 8th ed. Harlow, England: Person Education Limited.

Suman and M. Wadhwa," (2014) A comparative study of software quality models," International Journal of Computer Science and Information Technologies, vol. 5, no. 4, pp. 5634–5638

Tashakkori, A., & Teddlie, C. (2003). Handbook of Mixed Methods in Social & Behavioral
Research. Thousand Oaks: Sage.

Wallen, N. E., & Fraenkel, J. K (2001). How to design and evaluate research in education. The McGraw-Hill Company, Inc. New York.

Walsham, G., 1995. The Emergence of the Interpretivism in IS Research, Institute of Operations Research and the Management Sciences. Institute of Operations Research and the Management Sciences, pp.p.376-394.

Yahaya J, A. Deraman, A. R. Hamdan, and Y. Jusoh (2014), "User-perceived quality factors for certification model of web-based system," International Journal of Computer, Information, Systems and Control Engineering, vol. 8, no. 5, pp. 640–646

Yahaya J, A. Deraman, S. R. Ibrahim, and Y. Y. Yusoh (2013), "Software certification modeling: From technical to user centric approach," Australian Journal of Basic and Applied Sciences, vol. 7, no. 8, pp. 9-18, 2013.

Zahran, S. (1998). Software Process Improvement: Practical Guidelines for Business Success. Harlow, England: Addison-Wesley.