Corso di Laurea in Informatica

Software Quality Models and Metrics: a Survey

CANDIDATO Federica V. Maira

RELATORE Prof. Massimo Torquati

Anno accademico 2022/2023

Dedicata a chi mi guarda da lassù

Dadino, Antonio, Massimo

Dedicated to those watching over me from above

Dadino, Antonio, Massimo

Contents

Index of Figures

List of Acronyms

CBSD:	Component-Based Software Development
COTS:	Commercial Off-The-Shelf Components
EOSC:	European Open Science Cloud
IEEE:	Institute of Electrical and Electronics Engineers
ISO:	International Organization for Standardization
LOC:	Lines of Code
OSS:	Open-Source Software
SQA:	Software Quality Assurance

1 Introduction

A software product is a software solution, i.e., source code, including all its documentation produced from the first phases of the software development life cycle (SDLC), such as problem description and its requirements (user, technical, installation), up to test cases. In order to have a good product, its quality depends on both software solutions supplemented by its documentation, manual and online help.

Often an item is deemed to be of high quality if it is expensive, but this is not always true since a good quality software could reduce costs in terms of time, maintenance, and modification.

Actually, the definition of quality states that a good quality product needs to be compliant to customer's requirement. In 2003 the author Stephen Kan sustains that quality is represented by two aspects, the small q, the intrinsic product quality, and the big Q, which includes product and process quality and customer satisfaction, which could be measured through anonymous surveys [1]. All this leads to the IEEE[1] Glossary of Software Engineering Terminology which defines software quality as "the degree to which a software meets established requirements."

A bad software may be unreliable and cause several problems: sometimes it may just need high maintenance, some other issues could be loss of data, malfunction, security or privacy risks, loss of productivity, failure and, in medical environment, it may lead to death. As mentioned above, a high-quality software has to fit development standards to be compliant with customer's requirements and software quality models provide a guide to programmers in order to realise it. The evaluation is done through attributes named factors, defined as "attribute of software that contributes to its quality," which is the final characteristic that a customer expects to meet in the final product.

Factors may be divided into criteria and sub-criteria, which are the characteristics that define itself, offering a more complete definition of factors, showing the interdependence between them, helping in quantitative evaluation, defined as metric.

Software Quality Assurance is the process that ensures that the software meets the quality standard requirements during the whole life cycle of development.

[1] IEEE stands for Institute of Electrical and Electronics Engineers. It is one of the world's largest technical professional organizations dedicated to advancing technology for the benefit of humanity.

The aim of the study presented in this thesis is to provide general information about software quality, why it is so important and how it can be assessed.

We should consider that software quality assessment history dates back to the birth of software development, but the first model was developed by John McCall in 1977. Already at the time, it became necessary to evaluate software to get a high-quality product.

There are many reasons why it is fundamental to assess the quality of software.

Assessment helps to identify issues during the Software Development Life Cycle. The sooner the issue is detected, the sooner it is fixed, avoiding its propagation in the following steps, which could require more effort and higher costs to solve it.

Moreover, software compliant with the customer's requirements and which satisfies his needs creates a positive experience for the end-user and improves his satisfaction. The increase in customer trust and loyalty leads the business to a stronger reputation.

Software also needs to meet quality standards, which will be illustrated later in the text, to guarantee certain characteristics, such as reliability or efficiency. Indeed, assessment could identify if a software or part of it needs any kind of optimisation or improvements, in order to obtain more reliable and efficient software.

However, the assessment is difficult to realize: some parameters could be uncertain or subjective, software development is constantly evolving and therefore evaluation is considered a challenging task, reason why the research in software quality assessment is still in progress.

As the matter of fact, European countries agreed in providing a framework named EOSC (European Open Science Cloud), whose aim is to host and process research data to support European science, providing researchers and citizens with an environment where they can publish or look for data, tools and services, for research, innovation and educational purposes [2].

The study has been conducted by reading and analysing papers, books, and a few websites. Scientific publications have been mostly retrieved from the Google Scholar portal that links to the official journal versions or personal copies shared by authors. The Mathematics, Computer Science and Physics library at Pisa University allowed us to recover part of some books not available online or in nearby bookstores.

The search keywords used to find out papers and articles have been "software quality, models, metrics, software quality assessment, software quality evolution, software quality evaluation, software quality basic models, software quality tailored models."

The study has been carried out by analysing about thirty publications, whose results have been exported into a publicly available Excel spreadsheet that helped in collecting various information to write chapters and paragraphs.
The methodology used to select the scientific works and the method of study followed are described in detail in Chapter 4. The Excel spreadsheet contains information about publications, authors, year and place of publication and number of citations and we hope can be useful to researchers and practitioners for further research investigations and analysis.

1.1 Structure

The survey begins with chapter 1 Introduction which is composed of two parts: the aim of the thesis and its structure.
It is followed by chapter 2 Software Quality Background which describes what is quality software, why it is so important to assess it and how the assessment could be done.
Chapter 3 State of the Art is the main part of the thesis. It shows the evolution of the models starting from the first one from McCall, realized in 1977, and going through the years showing the progress from basic models to tailored models. The last section of the chapter proposes some of the most popular metrics used to evaluate in a quantitative way software quality and lists a few tools.
Chapter 4 Method of Study and Results will explain how the thesis research has been conducted and the results obtained.
The survey ends with chapter 5 Conclusion, in which we summarise the main outcomes of this study.

2 Software Quality Background

The chapter provides the reader with the needed definitions to understand the thesis content, focusing mainly on the one related to Software Quality. It explains its importance and necessity to be carefully analysed and explains which are the methods used to evaluate it qualitatively and quantitatively.

2.1 Definitions

A *software product* comprises a software solution attached to all its documentation.
A *software solution* consists of programming code, in particular source code. The product documentation refers to all the descriptions produced during the software process, starting from the problem explanation, going through software requirements specification (SRS), user-, technical- and installation manuals, and lastly support documentation, which includes test cases.
The quality of a software product depends on both quality of the software solution and the quality of all the other elements which make up the product (for example, the lack of quality of a user manual or online help involves a negative effect on the total quality of a software).

2.2 What is Software Quality?

There is no universal definition of Software Quality.
According to Stephan Kan in [1], the term *quality* is ambiguously defined and therefore commonly misunderstood. The common vision sustains that quality cannot be quantified and it is represented by the concept "I know it when I see it". Another popular theory argues that the more expensive a product is, the more quality it has.
On the other hand, from a professional standpoint, software quality must be defined and measured for improvement and its best definition is "conformance to customer's requirements".
Nevertheless, Kan sustains that the definition of quality consists of two levels: small and big Q. The first one represents the intrinsic product quality, often restricted to defect rate and reliability; the big Q, at the same time, enlarges the concept adding process quality and customer satisfaction.

Indeed, together with quantitative characteristics, there are also subjective features: the user's vision of quality differs from the developer's one.
Customer satisfaction is usually measured through surveys blindly proposed to the end-user, to realise whether the product complies with their needs.
As it will be shown later in the text, several models are based on a similar idea.
From [3] "Quality is the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs. Software Quality is an essential and distinguishing attribute of the final software product. Evaluation of software product, in order to satisfy software quality needs, is one of the processes in the software development life cycle".
As cited by Berlas in [4], even Gaffney in [5] argued that "the perception of software quality is different under different circumstances, and it is more informative to determine the characteristics of high-quality software rather than define software quality".
In general, software quality could be defined as compliance with functional and performance requirements explicitly stated. Indeed, the IEEE Glossary of Software Engineering Terminology defines quality as "the degree to which a software meets established requirements" [6].

2.3 Why is it so important to analyse the Quality of Software?

Customers' number of requests for software systems is growing louder; this leads to an increase in quality expectation in terms of product reliability.
"Bad" software may lead to different kinds of problems. If errors are not fixed on time, they could propagate to the next consecutive steps, causing unreliable services, which may not work correctly. These might cause loss of data, privacy or security issues, financial loss, permanent disability, mission failure, and others.
As reported in [7], healthcare institution uses complex software to manage big amount of medical data, to make accurate diagnoses, and thus come to the right clinical decision. Therefore, healthcare software developers must validate and verify their software to guarantee characteristics such as maintainability, reliability, and safety characteristics. Software affected by malfunctions or wrong calculations is unacceptable in any environment, especially in the medical field, considering that it may cause damage to human lives: a wrong diagnosis, caused by a system error, can lead the patient to death.
People often associate high quality with high costs. This is not always true in software quality assessment. Good quality software is not software that does not have problems or

bugs but it is the one in which bug fixes and repairs are quick and modification costs are minimized. It is also possible to reuse good pieces of code: this leads to the birth of COTS-based systems (Commercial Off the Shelf). These systems reduce costs and time-to-solution for companies [8].

It should be considered that creating software free of defects or bugs is impossible due to increasing product complexity. Software products allow several operational possibilities, so ensuring all of them is a big challenge; moreover, defects in software are not visible, so they are challenging to be detected. Several research efforts are ongoing on software verification in software engineering and DevOps practices to define tools and techniques [9].

2.4 How to check and measure Quality Software?

As mentioned above, software needs to be compliant with customers' requirements. Moreover, the software product development has to fit development standards (i.e., the acknowledged rules and processes for the creation of software) promoted by the company or coming from the core software team.

Software quality cannot be directly evaluated but need to be assessed through defined attributes. That is the reason software quality models come into play.

Quality models are tools that check and measure a software product quality; they have been defined to give a standard definition of quality and they provide a reference scheme that must be calibrated to represent quality requirements committed by the user. Their aim is to support high-quality software development, making it editable (for example to add new features) and reusable [10].

The attributes used to evaluate software are called software *factors*. According to [11] a quality factor is "an attribute of software that contributes to its quality". It follows that quality factors are characteristics that the final user expects to find in the software.

They describe different types of system characteristics. Factors are then divided into *criteria* and sub-criteria, which give a better and complete definition of a factor.

Criteria are the characteristics that define the quality factors. The criteria for the factors are the attributes of the software product or software production process by which the factor can be judged or defined.

According to [12] there are four reasons for expanding the list of criteria for each factor:

1. Criteria offer a more complete, concrete definition of factors.

2. It helps to illustrate the interdependence between factors.
3. Criteria allow audit and review metrics to be developed more easily.
4. Criteria allow us to identify those areas of quality factors which may not meet a predefined acceptable standard.

A quantitative approach to evaluate software is given by software quality *metrics*.
Metrics are the quantitative measures of the degree to which software processes a given attribute that affects its quality. In the next chapter, in section 3.4, a list of the most common metrics used is provided.

2.5 How is the Software Quality reached?

The process that helps achieve software quality is called *Software Quality Assuran*ce, SQA.
As stated in [13] "Measures of quality assurance are determined by standards of software quality lifecycle and a quality model with defined parameters for quality evaluation." SQA is the formal process that evaluates and documents the product's quality produced during each stage of the software development lifecycle [12].
Indeed, SQA ensures that the software product meets and complies with the defined or standardized quality specifications. It is part of the Software Development Life Cycle.
In particular, SQA must perform three important functions [5]:

- Define the standards for the software products developed in its organizational unit;
- Specify and implement tools or aids for assessing software product quality;
- Apply those tools to evaluate the degree to which the products developed by its unit comply with the standard proper to that product. The assessment may be qualitative or quantitative.

2.6 Summary

In this chapter, we introduced the concept of Software Quality and the importance of its assessment. Verifying the software quality is a fundamental process in the Software Development Life Cycle. Simple bugs may cause serious issues and damage as in the healthcare environment or in critical and widely-used services. Unfortunately, there is no single and widely accepted definition of Software Quality. Among the different

definitions reported in this chapter, a common point is that the assessment of Software Quality must take into account the customer's requirements and needs.

Software Quality could be measured in both qualitative and quantitative ways, through the use of models and metrics. Software Quality Assurance is the process that assesses and documents the quality reached in each step of the Development Life Cycle.

3 State of the Art

The chapter will give an insight into the literature on software quality models and metrics, dating from the seventies to nowadays. It lists and describes the most important quality models, categorising them into basic and tailored, which is the classification proposed first by Thapar [14] and a few years later extended by Miguel [15], who distinguish quality models as basic, tailored and open-source. A graph has been attached for each model for a better understanding. In most cases, these graphs represent graphically the relationship between factors and their relative criteria. The last part of the chapter shows the most common metrics used to provide a quantitative measure of software quality.

3.1 Evolution of Quality Models

As mentioned before, quality models are the tools that consent to value software product quality, using certain factors and their criteria.
The first model goes back to the 1970s.
According to [15], quality models can be categorized into two types:

- *Basic model*s, evolved during the years 1977-2001.
  Those models are suitable for any kind of software product; they follow a hierarchical structure; their aim is getting a complete and comprehensive product assessment [13].
  The most cited basic models are McCall, Boehm, FURPS, Dromey, ISO 9126 and its updates. It is possible to find a reference to them in almost all scientific papers about software quality.
  These models are considered the basic quality models: most of the other quality models are based on these ones by improving them somehow.
- *Tailored Quality Model*s. The first one was developed in 2001. They are also known as component quality models.
  These models have been realized to satisfy the need of organizations and software industries to get specialized evaluations on individual components.
  They are appropriate for a particular area of application, which means that features may vary in relation to a general pattern.
  They are developed by adding and modifying sub-factors to the basic models, especially the ISO 9126, in order to reach the goals of different domains.

Some of the tailored quality models are Bertoa, Georgiadou, Alvaro, and Rawashdesh and open-source models.

3.2 Basic Quality Models

In the following, I report some of the most important and used models, which are quickly described, and attached to the representation of their pattern.
There are many other quality models, but most of them are just mentioned in the texts analysed, as it will be shown in the next chapter through a table which summarizes the research.

3.2.1 McCall's Factor Model

McCall's model, together with Boehm's one, is one of the earliest and most cited for software in general. According to [16] those two models describe the main requirements for software users, including quality and characteristics, without any bound on particular software environments. The model has been developed in 1977 by J. McCall and it has been conducted by General Electric Company for the Air Force Systems Command Rome Air Development Centre [8] [17]. Indeed, it is also called FCM (Factor, Criteria and Metric) or GE (General Electronics model) [18].
It distinguishes into two levels: at the higher level there are the quality factors or external parameters: those attributes can be measured directly. At the lower level, there are quality criteria, or internal parameters, which are objectively or subjectively measurable.
The model consists of eleven factors, and each of them is so related to a set of criteria, and each criterion may refer to more than one factor and vice versa [13] [16].
Quality is defined through the following pattern, as shown in Figure 1 below:

- Product operation factors: Correctness, Reliability, Efficiency, Integrity, Usability. These factors determine the user's satisfaction.
- Product review factors: Maintainability, Flexibility, Testability. They are used to assess the ease of system adaption and error correction.
- Product transition factors: Portability, Reusability, Interoperability. This determines the ability to adapt to new environments [8] [12] [15].

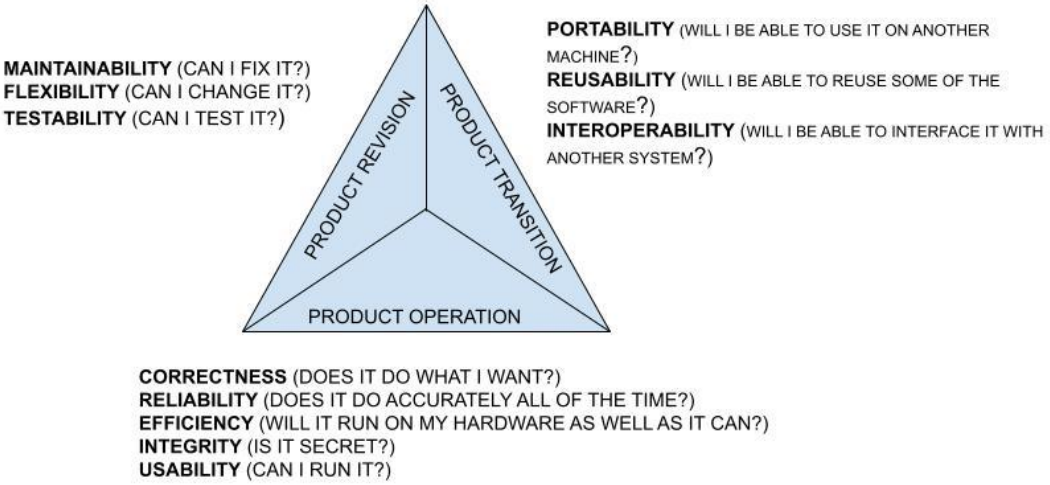

Figure 1 McCall Software Quality Factors

Nistala et al. state that "factors are user-oriented, and criteria are software oriented." [19]. Indeed, McCall's model seems that both consider users' view and developer priorities.
These could be one of the reasons why his model is used as the base for many other quality models (as it will be shown further in the text), together with the fact that it considers relationships between quality characteristics and metrics [15] [20].
Despite all, McCall's model has some drawbacks as well. Some of them are that the only responses allowed are "yes" or "no", which means loss of accuracy; moreover, it doesn't consider functionality, a big point of interest for the user [8] [15] [21].

3.2.2 Boehm Model

The model was developed by Barry W. Boehm in 1978. Together with McCall, as already written above, it is one of the most cited in software quality history. Boehm's model takes inspiration from Mc Call's one, "adding factors at different levels." It focuses on reliability and easiness of software.
The model is based on a wide range of characteristics, centring on maintainability, and it includes a new parameter, which evaluates hardware performance [13]. This information seems to disagree with the paper [15] by Miguel et al. and the one by Ronchieri and Canaparo [7]. Performance, Usability and Maintainability seem to belong to the model in [7] but they do not according to [15].

Boehm's model differentiates levels for quality attributes into three sections, which are categorized according to their characteristics, as represented in Figure 2.

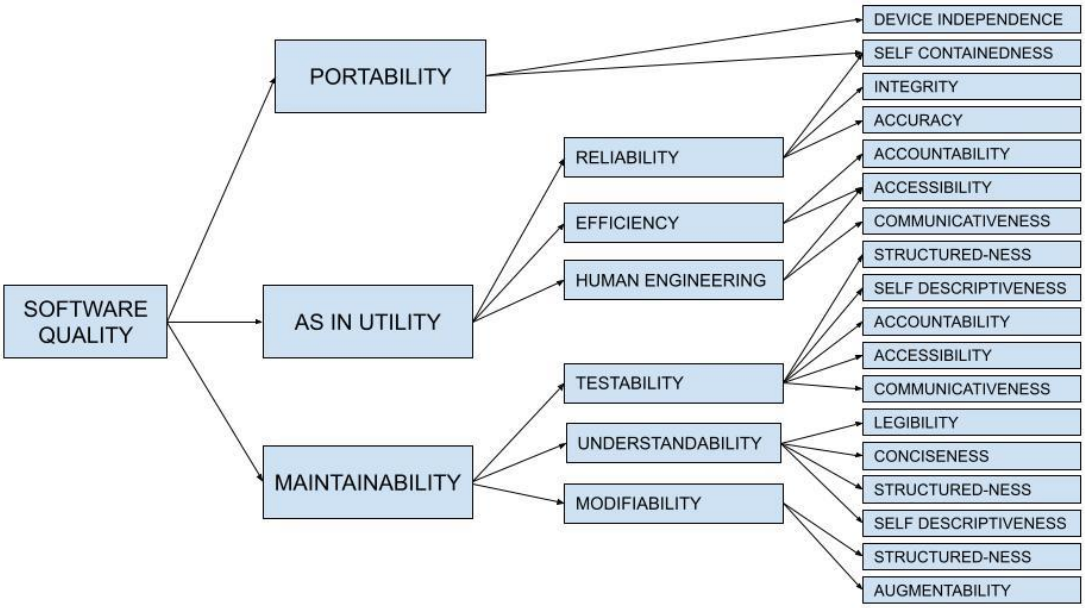

Figure 2 Boehm Model

These levels are:

- Primary uses (high-level characteristics):
  - *Utility*: it indicates how much the software is easy to use, reliable and efficient;
  - *Maintainability*: it shows if it is easy to modify, to test and to understand;
  - *Portability*: it shows if it is possible to use the software in a different environment.
- Intermediate (mid-level characteristics): it identifies seven factors which are expected by the software:
  - portability
  - reliability, efficiency, human engineering, usability [referring to Utility]
  - testability, understandability, flexibility [referring to Maintainability].
- Primitive constructs (primitive characteristics): it provides the ground in order to define quality metrics [7] [16] [18].

Its disadvantages are the lack of criteria and that it is very difficult to apply in practice [21]. Additionally, Boehm's model doesn't provide any suggestions about measuring the quality characteristics [8] [20].

3.2.3 FURPS and FURPS+ Model

This model has been presented by Robert Grady and Hewlett-Packard Co in 1987. An updated version will be introduced by IBM Rational Software some years later, in 2000, defined as FURPS+. The acronym FURPS stands for the five factors it is represented by: *Functionality, Usability, Reliability, Performance, and Supportability* [13] [21], as shown in Figure 3.

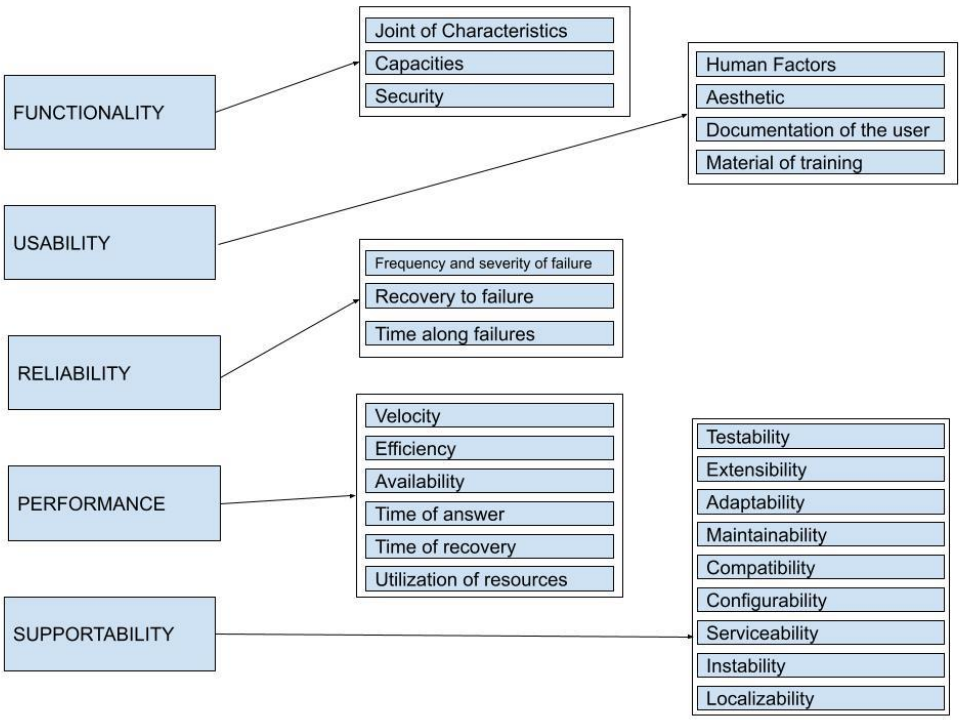

Figure 3 FURPS Characteristics

The model divides factors into two categories: functional and non-functional requirements. Functional requirements are defined by the input and output expected or by the system functionalities. Non-Functional are organised as usability, reliability, performance, and product support. Portability is not considered in the model [15].

The updated version FURPS+ adds to the previous model factors such as design constraints, implementation requirements, interface requirements and physical requirements [12].

3.2.4 Capability Maturity Model

The Capability Maturity Model, also known as CMM, is one of the most important models for quality software maintenance. It was released in 1991 by SEI[2] (Software Engineering Institute), belonging to the sponsored research and development centre of the U.S. Department of Defence. It has then been upgraded to CMMI, Capability Maturity Model Integration. CMM focused on software engineering and software handling; indeed, CMMI's priorities are development and product maintenance.

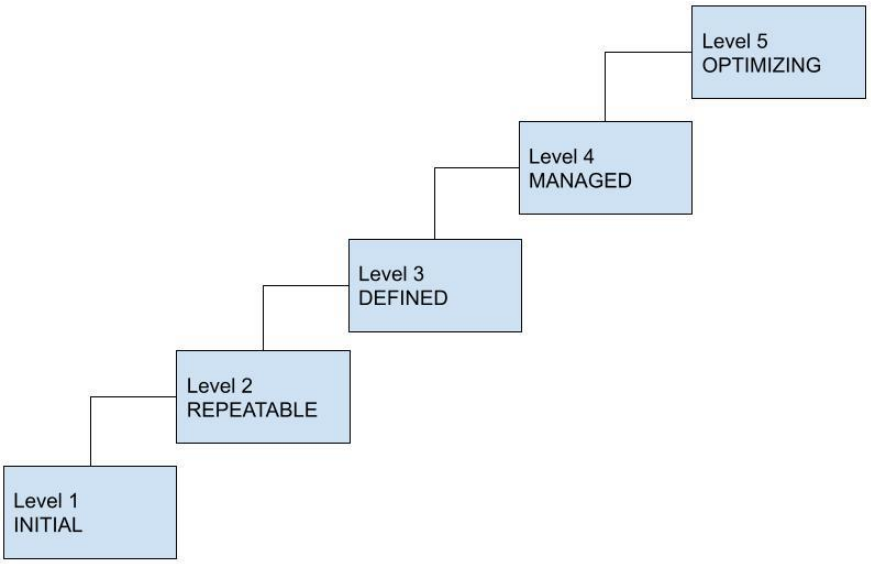

Figure 4 CMM Model

The CMM model provides a framework for continuous improvement. It is composed of five levels, which are also illustrated in Figure 4:

- initial level: small business, the processes are not organized and defined;
- repeatable level: business has reached success; processes are properly defined and documented;
- defined level – business has defined the standards for the software development process according to its own needs;
- managed level - business monitors and measures its processes for controlling performance through data collection and analysis;
- optimizing level – business aim is process improvement, by continuously monitoring feedback and using new tools and techniques [13].

[2] SEI is the Software Engineering Institute located at the Pittsburgh University in Pennsylvania. It is a non-profit, public– private partnership that conducts research for the United States government about software engineering, systems engineering, cybersecurity, and many other areas of computing.

3.2.5 ISO 9126 Model

Getting inspiration from McCall and Boehm models, its first version was released in 1991 and it proposes a standard that specifies a set of quality factors for software evaluation, in agreement with all country members of ISO[3] [7] [19].

The model involves six independent quality characteristics: Functionality, Reliability, Usability, Efficiency, Maintainability, and Portability. Those will be then further broken down into sub-characteristics. Furthermore, usage characteristics focus on productivity, effectiveness, safety, and satisfaction [11] [13] [16] [18] [22].

The model is divided into two parts: the first one describes the internal and external quality, and the second one describes the quality in use. The model schema is sketched in Figure 5.

- Internal quality: system properties evaluable without execution; indeed, it is assessed during the code review phase, before the code is executable;
- External quality: properties assessed by the software usage and its maintenance. Need execution to be evaluated (during the testing phase);
- Quality in use: it is the measure to which a product can be used by customers to meet their requirements and achieve specific goals. Those features refer to the effectiveness, productivity, security offered to the applications and satisfaction of users. It is assessed after the delivery to the end user.

The ISO model advantage is that it uniforms the guideline for quality characteristics in order to have a universal model, which makes it easier to compare different products and make it applicable to every kind of software. The model has been widely adopted in the software engineering community since it is easy to use and understand by users. Indeed, it has been used as the basis for developing tailored quality models.

Its disadvantage is that it does not show and discuss how attributes can be measured in software systems. Moreover, the traceability of the software and the consistency of the data are not represented in the model [21].

In 2011 it was updated to ISO 25010 model, also known as SQaRE (Software Product Quality Requirements and Evaluation). It adds Security and Compatibility, increasing from six to eight factors, and redefines the fundamental characteristics.

[3] ISO stands for International Organization for Standardization, and it develops and publishes International Standards

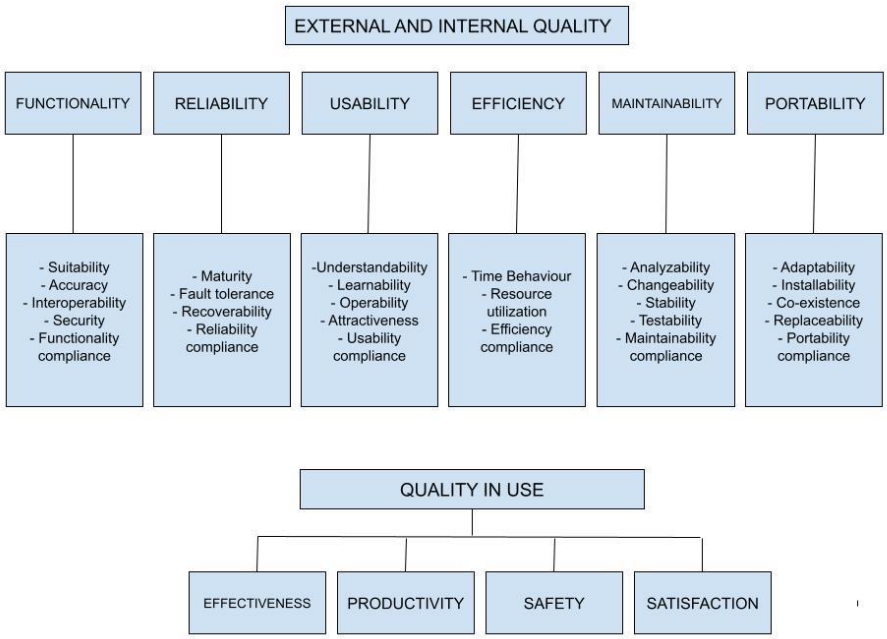

Figure 5 ISO 9126

3.2.6 Dromey Model

The model was developed by R. Geoff Dromey in 1995. Miguel J. et al. in [15] stands that "The model states that for a good quality product, all the elements that constitute it, should be so".

Dromey proposed a framework for the evaluation of requirements, design, and implementation phases [12]. He focused his attention on the relationship between quality attributes and sub-parameters, in order to connect software quality attributed to software product properties [7] [8] [20]. He stressed the importance of identifying defects and the factors that caused them. Properties can be classified into four parts: Descriptive, Contextual, Correctness, and Internal [12] [13]. The model is shown in Figure 6.

According to Nistala et al., in [19] "Dromey points out that software does not directly manifest quality attributes but exhibits properties that contribute to quality attributes."

The model is applicable to different systems [20] [21]; however, it seems to be difficult to understand and it does not define how product properties must be realized [15] [21] [19]. Moreover, it is not feasible to judge if a system is reliable or maintainable before it is actually operational in the production area [8].

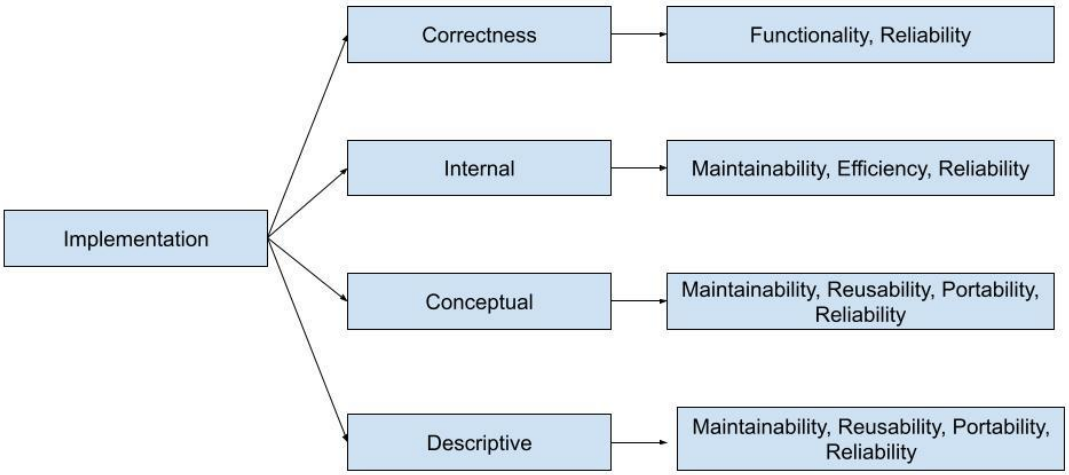

Figure 6 Dromey Model

The model gives inspiration for designing other specific models, e.g., Quamoco [23] [24], QMOOD [25] and Rawashdeh [8].

3.2.7 ISO 25010 Model

The model emerges in 2007 as the updated version of the previous ISO 9126. Despite all, according to research conducted in 2020 by Google Relative Search Index (as cited by Galli et al. in [26]) the elder version keeps being the most frequently used. ISO 9126 has been in use for several years before the release of its new version. As a result, numerous studies have been conducted, leading organizations to be accustomed to its utilization. However, as software continues to evolve, it is expected that ISO 25010 will gain even greater popularity in the future. Analysing 2021 and 2022 the results show a similar trend, but in the few months of 2023 (carried out on 16th May 2023) the situation seems to be turned around: ISO 25010 has been searched 56% more than ISO 9126.

The ISO 25010 goal is to "guide in the development of software products with the specification and evaluation of quality requirements" [7]. The model maintains the same view of its predecessor (internal, external, and quality in use attributes) but its features are redefined and partitioned into eight sub-keys (two more than the elder model), each of them with its characteristics. Security and Compatibility are two of the new factors.

Portability becomes a characteristic of the new feature *transferability*, together with *Security, Operability and Performance efficiency*; those last two are going to replace Usability and Efficiency, belonging to ISO 9126. Updates are visible in the following Figure 7.

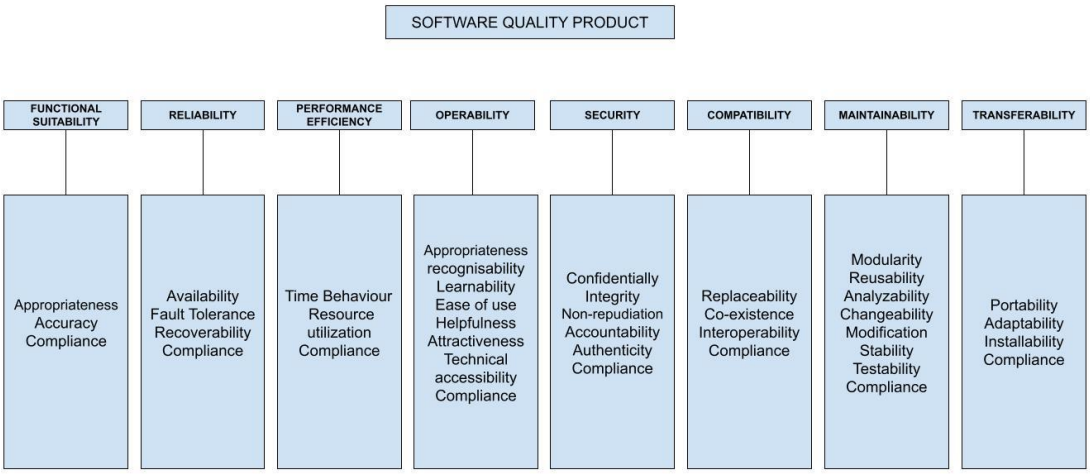

Figure 7 ISO 25010

ISO 25010 belongs to the series of standard 25000, which also includes 25020 regarding quality measurement, 25030 on quality requirements and 25040 regarding quality evaluation.

3.2.8 Other Basic Models

Some other basic models mentioned in the literature are:

- GHEZZI: The model was developed by Carlo Ghezzi et al. in 1991. Developers may use internal qualities to control software structure and reach the required external qualities. Internal factors to maintain are Accuracy, Reliability, Portability, Reusability, Maintainability, Flexibility, Usability, and Integrity [13].
- IEEE 1993: The model delineates the standard for software maintenance, which is qualitative. Factors included in the model are Reliability, Usability, Maintainability, and Portability. It includes other standards such as Software Quality Assurance Verification and Validation [13] [27].

3.3 Tailored Quality Models: CBSD and COTS

Most of the Tailored Quality Models get inspiration from ISO 9126. Those models consider a specific domain and then examine features and sub-features. They focus on the evaluation of components and the importance of features may vary depending on models. Hence, as mentioned above, they are also known as component quality models.

It is quite hard to make a comparison among tailored-oriented models since the model is used in a specific context; for instance, GEQUAMO is product-oriented, Bertoa is for particular domains, Rawashdeh is adapted from the user's perspective and so on [13].
Since 2001 software developers uses a new approach to develop new software according to Component-Based Software Development (CBSD), a technique whose aim is to promote code reuse.
COTS (Commercial Off-The-Shelf Components) and OSS (Open-Source Software) are two kinds of software that could be integrated to implement a CBSD application. As defined by the National Security Agency of the USA [28], COTS products (hardware, software and services) are "commercially ready-made and available for sale, lease, or license to the general public", and they encourage their use: they emphasize purchasing commercial components rather than building new software, in order to provide quick delivery to end users, share development costs, and having more warranties, considering that code has already been tested and assessed. In 2006 COTS were going to exceed 40% of the total development software. Hence, the importance of developing tailored software quality models, in order to evaluate component quality and avoid issues propagating throughout the system.

Below some examples of the most used tailored quality models will be described.

3.3.1 Bertoa Model

Based on ISO 9126 Model, Bertoa proposed the first tailored quality model in 2001 [29]. He sustains, together with his co-author Vallecillo, that ISO 9126 characteristics were too generic for dealing with open-source software and so they developed a new model which comprehends a set of quality attributes suitable for COTS so that software companies could use it to build complex software [16].
The model, shown in Figure 8, specifies characteristics belonging to the ISO 9126 which are compliant with individual components, in order to evaluate "commercial off-the-self" products for CBSD [7] [14].

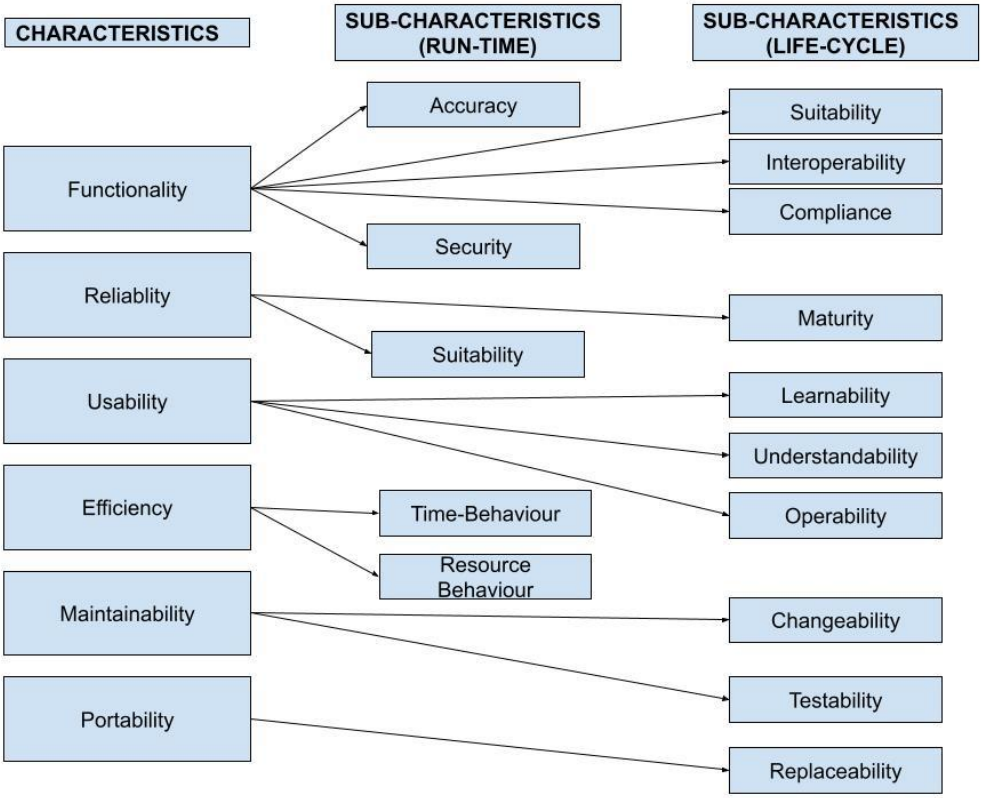

Figure 8 Bertoa Model

3.3.2 GEQUAMO Model (by Georgiadou):

It stands for Generic, multi-layered and customizable Quality Model [11]. It was created in 2003 by Ellie Georgiadou and it "encapsulates the requirements of different stakeholders in a dynamic and flexible manner so as to enable each stakeholder (developer, user or sponsor) to construct their own model reflecting the emphasis/weighting for each attribute/requirement".

The model is built using a combination of Kiviat[4] diagrams and CFD[5] (Composite Features Diagramming Technique) developed by the author. The CFD provides both qualitative and quantitative information. The CFD illustrated in Figure 9 represents an item under evaluation (such as method, model…), having three primary characteristics R1, R2, and R3, and each of them having its own sub-characteristics. The process continues until simple and more easily measurable characteristics are defined. At each note, a polygon, named the Kiviat diagram, can be constructed based on the number of

[4] Polygons (triangle, rectangle, pentagon) which can represent quantitative information belonging to the same layer. It represents actual and threshold values (the higher the value the more it is better)

[5] CFD provides a qualitative platform for depicting a profile (absence or presence of features/characteristics/sub-characteristics)

sub-characteristics. The end-user can build its own model assigning to each attribute its own weight, to get a quantitative evaluation. Further details could be found in [11].

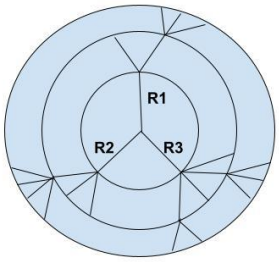

Figure 9 CFD with three primary Characteristics R1, R2, R3

3.3.3 Alvaro Model

It was realized in 2005 by Alvaro et al. [30] to make up for the lack of a good evaluation system for CBSD.

The model is represented in Figure 10: it is based on ISO 9126, and uses a framework composed of four modules:

- Component Quality Model (CQM): the aim is to find out which quality characteristics and sub-characteristics are necessary;
- a key CBD (Component Based Development) Quality Characteristics, in order to define which characteristics are essential for certification. It is organized into five levels, the higher the value, the more reliable it is (5 is the highest value);
- an Evaluation Process, to establish techniques, models, metrics and tools to assess and attest software components;
- a Metrics Framework, which defines a set of metrics to track the properties of the components and the evaluation process.

Most of the factors are the same as the ISO 9126, even if changes have been made to features and a new factor has been added: Business, which comprehends characteristics regarding marketing (such as time to develop a component, time to make it available on the market, cost and how affordable it is).

Sheoran and Sangwan [31] developed a method for software quality prediction, comparing models such as ISO 9126, ISO 25010, Bertoa, and Alvaro and selecting certain factors and criteria. Results show that the Alvaro model was able to provide better results

than the ISO 9126, ISO 25010 and Bertoa, especially regarding factors such as Accuracy, Testability and Understandability.

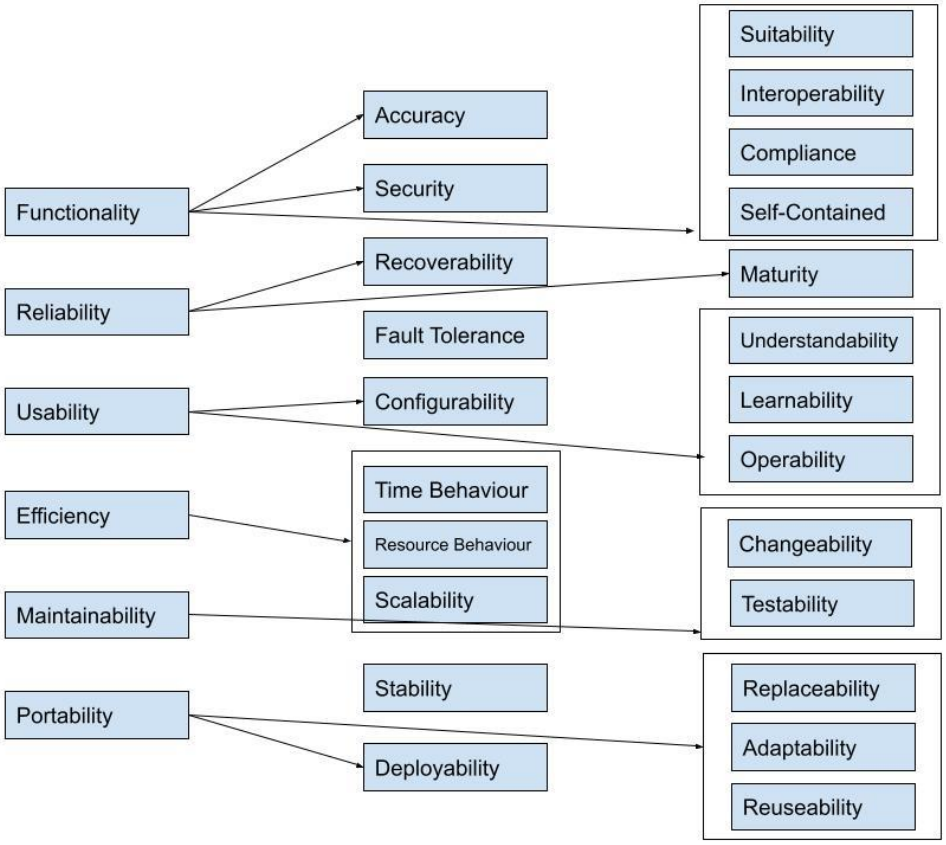

Figure 10 Alvaro Model

3.3.4 Ortega Model

Ortega's model [32] is based on a systemic approach, getting inspiration from basic models such as Dromey, ISO 9126 and McCall, and a model proposed in 1996 by Callaos and Callaos [33], which uses a systemic method as well.

From [32] "The evaluation method consisted of designing a survey, formulating, validating and applying the measurement instruments; defining an algorithm to obtain the quality estimate and analysing the results."

The model considers quality and sub-quality factors in order to get to quality metrics.

Features belonging to Ortega are product effectiveness (by ISO 9126), product efficiency (by Dromey), and process effectiveness and efficiency as well [13] [19] [14] [32]. The resulting model is shown in Figure 11.

Unfortunately, the model focuses on evaluating both product and process dimensions, but it doesn't take into account building or designing software quality [19].

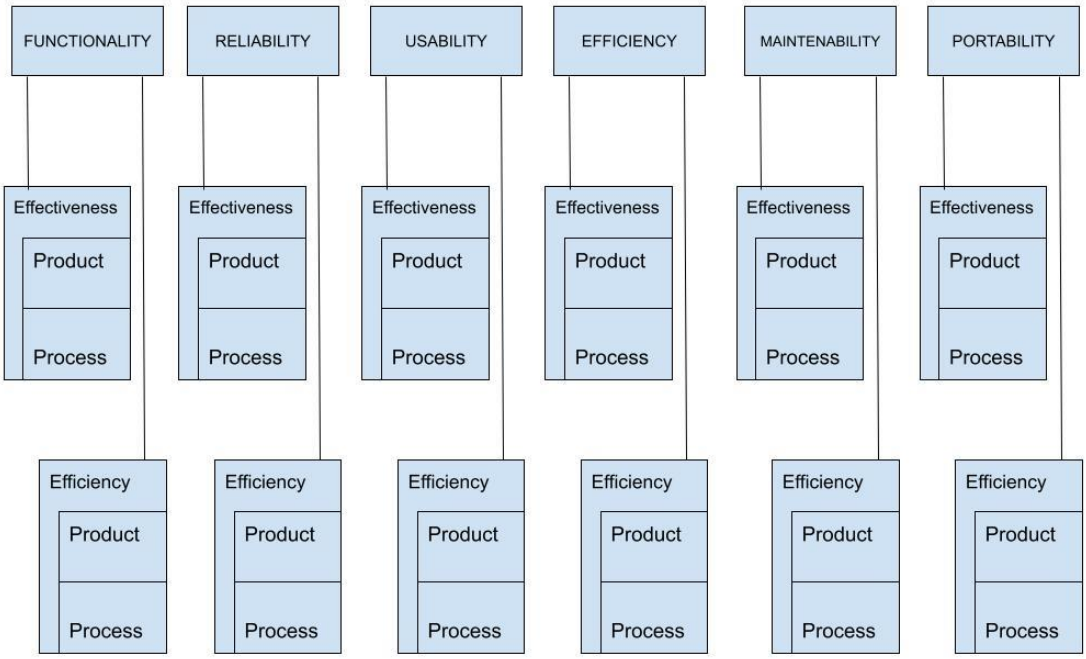

Figure 11 Ortega Model

3.3.5 Rawashdeh Model

It is based on Dromey and ISO 9126 models, and it has been released around 2006 by Adnan Rawashdeh and Bassem Matalkah [8].

It supports a set of quality features and sub-features suitable to evaluate COTS products and avoids considering those not applicable to COTS. Rawashdeh and Matalkah found out that there were many limitations in elder models regarding this kind of new software, so they built a new one tailored for them. Some of the limitations derived from other models are that in each of them one or more important factor is missing (McCall does not consider Functionality, Portability is missing in FURPS, and in Dromey's model it is not possible to evaluate Reliability and Maintainability before its execution; Boehm, ISO 9126 and some non-hierarchical models do not suggest how to measure characteristics).

The goal of this model is to assess the needs of different kinds of users: "it matches the appropriate type of stakeholders with corresponding quality characteristics," a feature not

available before [8] [14]. The model catches characteristics from both hierarchical and non-hierarchical models, to combine both advantages.

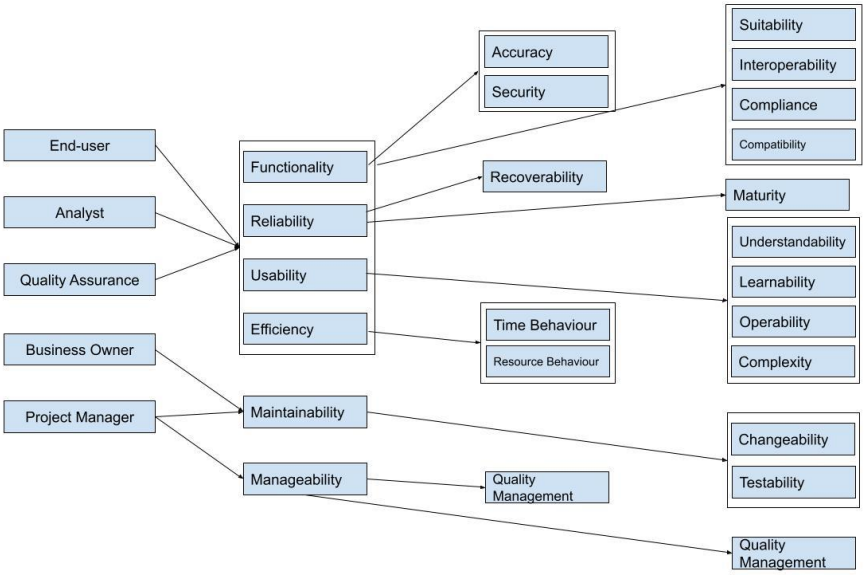

Figure 12 Rawashdeh Model

The model's representation is visible in Figure 12.

It describes how to create a product quality model using four steps:

- identify a few high-level quality attributes;
- for each high-level attribute to obtain their sub-attributes;
- split internal from external metrics: the first one measure source code or specification, the second one checks the system behaviour during testing operations and components;
- for each quality attribute, identify its users [13].

3.3.6 Open-Source Models

The "Open source" label first appeared in the late 1990s, at a meeting in Palo Alto (California), soon after the announcement of the release of the Netscape source code [34]. The first software quality models came out in 2003, finding their basis in basic models such as ISO 9126, integrating some specialized parameters for open-source software. In 2020, according to [13], an optimal model was not found yet.

Examples of quality models for open-source code are:

- Cap Gemini: the model uses product and application maturity indicators, and each of them gets a score on a 1-5 scale [13] [15] [35].

- OpenBRR: (Open Business Readiness Rating) it is a framework based on Cap Gemini and ISO 9126 models dated to 2005. It aims to accelerate the evaluation of open-source software using seven categories, which could be refined to simplify and assess at a granular level [13] [15] [35].
- SQO-OSS: (Software Quality Observatory for Open-Source Software) it is a hierarchical model which evaluates community processes and source code based on the automatic calculation of metrics. It doesn't consider product functionality [13] [15] [36]. Based on Ndukwe et al. study conducted in 2023 [16], SQO-OSS is at the top of the list of open-source models.

  The model has been implemented to reduce human interaction and realize tools completely (or almost completely) automatically. This has been subject to considerable controversy since Campbell in [37] sustains that human involvement is essential to better evaluate source code since writing and maintaining code are human processes and it is the programmer's responsibility to assess how their output has to adhere to mathematical models.
- QualOSS: the model focuses on work product, community members and software processes. It states that quality depends on the context in which it is used.

In [38] Haaland et al. made a comparison between OpenBRR, belonging to the first generation of FLOSS[6] models, and QualOSS, belonging to the second one. Results show that both models have strengths and weaknesses, so it is not possible to deduce if second-generation models outperform the first. The limited number of metrics of OpenBRR may cause a loss in important quality issues, and, moreover, the approach makes it susceptible to subjective biases. On the upside, QualOSS seems to be complex and, even if it tries to eliminate human imprint, sometimes it must rely on it to decode the results of assessment or when tools are not accessible.

3.3.7 Other Tailored Models

The surveyed publications offer some other tailored models, such as:

- SQUID (Software Quality User Interface Design), by Kitchenham et al. [39]. Its three major keys are product operation, product revision and product transition. The model is distinguished into two components: a structure model, which defines

[6] FLOSS st [39]ands for "Free/Libre and Open-Source Software".

the elements and their relationship, and a content model, which identifies entities linked to the structure [7] [13] [19].

- QMOOD (Quality Model for Object Oriented Design) by Baniya and Davis in 2002 [25]. The model aims to create a quality model which evaluates the outcomes of the first phases of the development lifecycle, such as analysis and design. It is based on ISO 9126 and Dromey models [13] [26].
- SAAM by Kazman, 2003, [40] focuses on characteristics such as Functionality, Availability, Efficiency, Reusability, Testability, Security, Portability, Inheritability, and Modifiability [13].
- QUAMOCO, by Wagner [24]. The model tries to connect quality characteristics to quality measurement. It is based on the Dromey model since its keys are product factors, quality aspects and entities [13] [19].

Many more models could be found in the literature available in the bibliography, but most of them are just mentioned.

3.4 Metrics for Assessment of Various Quality Characteristics

Software metrics measure development processes and software characteristics. They are valuable for many reasons, including measuring software performance, planning work items, measuring productivity, and many other uses.

Cavano and McCall in 1978 [17] state that "metrics are quantitative measures of the characteristics of the software which provide certain qualities".

Gaffney in 1981 [5] points out that a software metric is an "objective, mathematical measure of software that is sensitive to differences in software characteristics" and "it provides a quantitative measure of an attribute which the body of software exhibits".

Basically, metrics help developers in analysing and then improve their code if necessary. There is the idea that in order to measure something, this should be first converted into numbers.

IEEE 1061:1998 definition of software quality metric is "a function whose inputs are software data and 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" [41]. The definition has been reaffirmed by the same IEEE in 2009.

In the literature can be found at least three types of software metrics:

- Product metrics (or code metrics): they measure the final product and its target achievement. They describe characteristics of the product, for example, size, complexity, or design features. These metrics can be further grouped into:
    - Size metrics
    - Complexity metrics such as Halstead or Cyclomatic.
- Process metrics measure the development life cycle of the software system, including methodology, staff status, and time required to finalize the system. These metrics can be computed at any phase of software development [42]. They may improve the development and maintenance phases.
- Project metrics involve project characteristics and execution. They comprehend costs, number of developers, schedule, and productivity.

In [4] and [12] Lee adds three more categories: Requirements Gathering, Customer Satisfaction Metrics, and Maintenance Metrics. Customer Satisfaction Metrics are also cited in [43] which reports that it is, together with perception trends, one of the examples of the satisfaction quality attribute of Quality in Use (in ISO 25010). As written by Stephen Kan in [1] it could be measured through surveys, usually blind ones (customer do not know what company the interview represents, as, at the same time, the interviewer do not know the customer).

In the papers analysed, the Maintenance metrics are only mentioned by Lee [12] and by Doneva [22], referring to metamodels.

The literature offers a wide range of software metrics. In the text, just a few of them will be described.

Some of the most common software metrics which evaluate source code complexity are:

- LOC: lines of code. Below more details about this metric will be shown.
- McCabe's Cyclomatic Complexity (v(G)): it represents the logical complexity of the software and the effort to realize it [44].
- Object Oriented metrics, such as Cognitive Complexity [37].
- Halstead's metric [45] [46].
- Number of classes and interfaces: used on object-oriented code.
- FP: function point [47]. Used to evaluate functionalities offered to the customer before the software has been realized.

- Number of errors for each line of code: it could be used to estimate the reliability of the software. It measures the ratio between the total number of errors and code size.
- Code Churn: it measures the change made to a component over a period of time [48].

In the following, we will further explore certain metrics in more detail: LOC, Cyclomatic and Cognitive Complexity, and Halstead's Metrics. A few more metrics will be briefly mentioned.

3.4.1 LOC: Lines of Codes

The property assesses by this metric is the software size. It is used to measure the size of a computer program by counting the number of lines in the text of the program code. LOC is supposed to be the easiest software metric to understand, compute and interpret, and it is the most frequently applied measure in software estimation and quality assurance. It is usually used to predict the amount of effort required to develop a program or to estimate programming productivity (once the software is ready).

To compute the counting code, it is necessary to determine which rules to use for the comparisons (for example inclusion or exclusion of comments or considering only the released code and not the produced one). This explains the difference between physical and logical LOC: the first one excludes comments and blank lines from the count (if they are less than 25% of total lines): it is the physical length of the code as it appears when printed for people to read. In contrast, logical LOC attempt to measure the number of executable "statements", in terms of software instructions (for example in C-like programming language it counts only instructions ended by semicolons).

LOC could be represented as kLOC (thousand) or mLOC (million), depending on the measurement unit used. Moreover, there are various kinds of LOC, both for physical and logical LOC, such as SLOC, kSLOC and DSLOC (used for physical ones), where S stands for source, and D where only delivered source lines are included in the measure. Some others may regard the presence or absence of comments or executable instructions: NCLOC represents non-comment LOC, which is the opposite of ELOC (effective LOC); EXLOC expresses "only executable instruction."

LOC also presents some drawbacks: it correlates poorly with the quality and efficiency of code, and it doesn't consider complexity; it should be considered that there are

programmers whose productivity is being measured in lines of code, so they will have the incentive to write unnecessarily verbose code: this causes unneeded complexity to the code. Moreover, the number of lines of code differs from person to person. An experienced developer may implement certain functionality in fewer lines of code than another developer with relatively less experience, though they use the same language. Further, the number of lines of code needed to develop an application will differ using different programming languages.

Lack of counting standards: there is no standard definition of what a line of code is. Considering what was written before, comments may be included or not, and so too data declarations. A statement could be extended over several lines, how many lines should be counted? Both Robert Park (while at the Software Engineering Institute) and more official organizations, like SEI or IEEE, have published some guidelines in an attempt to standardize counting, but it is difficult to put these into practice, especially in the face of newer and newer languages being introduced every year [49].

3.4.2 Cyclomatic Complexity

Cyclomatic Complexity has been developed by Thomas J. McCabe in 1976 [44], and it is used to indicate the complexity of a program. It measures the number of linearly independent paths through the control-flow graph[7] of a program. Paths are generated by control instruction of a single method or program, such as IF-THEN-ELSE, LOOP, or SWITCH.

In 2005 Pete Lindstrom, CISSP Research Director at Spire Security, asserts: "Cyclomatic complexity is the most widely used member of a class of static software metrics. Cyclomatic complexity may be considered a broad measure of soundness and confidence for a program" [50].

Cyclomatic complexity allows to identify software modules that will be difficult to test or maintain. It determines the stability and level of confidence in a program. If a program has low cyclomatic complexity, it is easier to understand and less risky to modify.

The first definition given by McCabe in 1976 is:

"The cyclomatic number V(G) of a graph G with n vertices, e edges and p connected component is:

[7] A control-flow graph is a representation of all paths that might be traversed through a program during its execution.

$$v(G) = e - n + 2p$$

In a strongly connected graph[8] G, the cyclomatic number is equal to the maximum number of linearly independent circuits" [44].

A control flow graph describes the logical structure of software modules, which correspond to a single function or subroutine with a single entry and exit point. Each graph, as defined by T. McCabe, consists of edges and nodes, where nodes represent computational statements or expressions, and edges stand for the transfer of control between nodes.

In the definition of v(G), v corresponds to the cyclomatic complexity in graph theory and G expresses that the complexity is a function of the graph.

Properties of cyclomatic complexity:

- v(G)>=1
- v(G) is the maximum number of linearly independent paths in G;
- Inserting or deleting statements in G does not affect v(G);
- G has only one path if and only v(G)=1;
- Inserting a new edge in G, complexity increases by one unit;
- V(G) depends only on the decision structure of G.

It can be calculated following these steps: construction of graph with nodes and edges from code, identification of independent paths, calculation using v(G) = e – n + 2p, and design of Test Cases.

Since the calculation could be difficult and error-prone for programmers, two alternatives have been provided:

- if all decisions are binary, could be measured using π +1, where π represents the number of binary decision predicates in the program (decisional points, like if or conditional loops).
- counting the number of regions (zones) of the graph, including the infinite region outside the graph; this could be proved by applying Euler's formula.

Cyclomatic complexity should be limited to help avoid hard code, which could be hard to understand, test or modify. Its limit has been to ten, which represents a simple procedure with little risk. A higher value would represent a more complex code with higher risk.

[8] A connected graph is one in which each node can be reached from any other node by following directed edges in the graph.

Anyway, the precise limit number remains controversial: even if the original limit of ten has significant supporting evidence, in certain occasions, the limit could be incremented to fifteen [51].

3.4.3 Cognitive Complexity

Cognitive Complexity is a metric introduced as an extension of Cyclomatic Complexity by SonarQube[9] [10] to assess code understandability.

Programs with the same Cyclomatic Complexity may differ in understandability. Even if they obtain the same value, one of them could be much more difficult to be understood.

Unlike the other metrics, Cognitive Complexity abandons mathematical models on behalf of simple rules which turn programmer intuition into numbers. The score is reached by applying three basic rules [37]:

1. Ignore structures that allow multiple statements to be readably shorthanded into one;
2. Increment (add one) for each break in the linear flow of the code;
3. Increment when flow-breaking structures are nested.

There will be no increment for a method declaration or null-coalescing operators (to not penalize those developers writing shorter code); contrarily, complexity will increase in case the flow of statements is broken or to take the level of nesting of control flow structures into account (such as switch, if, else, for, while, do while, catch and similar).

In 2023 Lenarduzzia et al. [52] made a comparison between Cyclomatic Complexity and Cognitive Complexity making a study among developers with one to four years of professional experience. Two studies [53] [54] about the effect of Cognitive Complexity on code understandability had already been done (one of them conducted by the same author of the metric). Lenarduzzia considered that they were not enough to demonstrate the effectiveness of a new metric.

The research conducted asked developers to manually inspect twelve Java classes, already analysed by the tool SonarQube [10], to rate their understandability. Results show that Cognitive Complexity outperforms Cyclomatic Complexity, even if the difference is not drastic. Moreover, they suppose that it is impossible to claim that higher cognitive or cyclomatic complexity implies problems with higher severity [37].

[9] SonarQube is an open-source tool available in twenty-seven programming languages, which verify quality identifying vulnerabilities and bugs in the code.

3.4.4 Halstead's Metrics

Maurice Howard Halstead introduced a suite of metrics in 1977, known as Halstead metrics or Halstead Software Science [45] [46].

Unlike other metrics, which are usually applied to one specific aspect of a software product, Halstead metrics are suitable to several aspects of a program, along with the overall production effort. They are based on indexes, such as:

- n1 = the number of distinct operators which appear in a program (for example assignment, control construct…)
- n2 = the number of distinct operands which appear in a program (variables and constants)
- N1 = total number of operators occurrences
- N2 = total number of operands occurrences

Through those indexes, he derived several formulas, defined as "Halstead's Software Science Metrics." Some of them are:

- Program Vocabulary, defined as $n = n1 + n2$; n represents the total number of distinct tokens (or vocabulary) in the program. A computer program is seen as a sequence of tokens, which could be operands or operators.
- Program Length, defined as $\mathrm{N} = \mathrm{N1} + \mathrm{N2}$, represents the count of the total number of operators and the total number of operands.
- Program Volume, defined as $V = N \log_2 n$, where N is the program length, and n is the program vocabulary. V indicates the storage volume required to represent the program.
- Potential Volume (or Intelligence) is defined as $V' = N' \log_2 n'$, n' and N' indicate the potential vocabulary and the potential length of the program. The formula represents the minimum amount of "information" of an algorithm.
- Quality Level Program: defined as $L = V/V'$, shows the ratio between volume and potential volume. The more L is close to 1 (the ideal value), the better the program quality is.
- Effort, defined as $E = V/L$ indicates the ratio between volume and level. It relates to the difficulty a person finds in understanding or modifying a program; also relates to the error proneness of a program.

Halstead formulas provide several measurements on code, but they also present some issues and criticism. Sometimes they seem to be hard to compute since it could be difficult to identify operators or operands in a program. Moreover, Halstead does not specify a value to define the program as complex [4] [5] [42].

3.4.5 Other Metrics

The literature provides a large number of metrics for measuring the quality of software products. Among the texts analysed appear metrics not described above, but some of them are going to be cited below.

- RADC's methodology: it expanded Boehm's model. The requirements show a ratio between the actual occurrence to the possible number of occurrences for each situation [12] [55].
- Albrecht metric: it determines the number of elementary functions, hence the source code's value. The metric assesses the amount of effort necessary to design and develop customer applications software [47].
- Henry and Kafura Information Flow Metric: it describes the amount of information flow connection into and out of a procedure [42].
- Metrics such as size metrics, and test process metrics, for software maintenance, which assess reliability, readability, customer problem or customer satisfaction. A short description of them could be found in [12].
- Code Churn: measures the changes made to a component over a period of time, estimating how many lines of code have been added, changed or deleted by the programmer. It predicts the defect density in software systems. [48]

3.4.6 Tools

The market provides several software metrics tools, some of them are free-software/open-sourced, others by subscription only. Usually, they differ from each other in language or platform supports, licensing prices or license type [56].
Many development environments provide extensions to compute metrics: for example, in IDEs such as Eclipse or Visual Studio[10], it is possible to use one of their plugins for

[10] Eclipse and Visual Studio are two different integrated development environments for open-source software. Eclipse's primary use is for developing Java applications, but it provides several plugins in order

metrics tracking and analysis; “Checkstyle” is the software used for Java language. These add-ons check many aspects of source code: detecting class or method design problems, and analysing cyclomatic complexity; they also have the ability to check code layout and formatting issues.

One of the most used tools is SonarQube, whose description (in Italian) could be found in a thesis of 2020 by Eleonora Pirri [10] or on its website [57]. The tool has also been used by Lenarduzzia et al. [52] to assess cyclomatic and cognitive complexities to conduct their research.

3.5 Summary

In this Chapter, we presented quality models using Thapar and Miguel classification, which categorizes models into basic, tailored and open source. Getting on in years, it is noted that tailoring is addressed to context-specific areas, which could include code snippets, Internet of Things, mobile applications or AI technology. Indeed, a few years later, have been developed frameworks for measuring the quality of IoT systems [58] and for mobile applications [59].

This topic was not covered in the text. The interested reader may refer to [16]. The study conducted by Ndukwe et al. [16] is comprised of two parts: the first one examines the literature to get the factors most used for judging software quality and those most used by researchers to study the quality of code snippets. The second part, indeed, examines the factors used by practitioners for the same purpose. The goal is to provide a reference for software developers who use code snippets available online during development, since the big increase in consulting them during the last years.

to be used with other languages. Visual Studio is mainly used by .NET and C++ developers, but by installing extensions, it supports numerous programming languages.

4 Method of Study and Results

The chapter is going to explain how the research has been conducted, where materials have been retrieved and how the work has been organised to get a wider variety of information.

4.1 Method of Study

The study covers research of papers, books, thesis, and conference papers.
The search took place both through Google Scholar, and online scientific journals and through the help of the Mathematics, Computer Science, and Physics library at Pisa University.
Online is available a large assortment of technical papers which deal with the first models available in software quality history, such as McCall, Boehm, and the others mentioned in the previous chapter; some other papers concern a quantitative approach, dealing with metrics; about new frameworks developed, and others report results of studies carried out. Only a few of them published in the last few years handle with a specific focus on the comparisons between certain models or metrics.
The use of Microsoft Excel helped me a lot in collecting and summarizing data. The Excel file that we used for the study can be downloaded from the following link: https://github.com/fmaira/SWQualityModelsMetrics.
We analysed around thirty papers, but the complete research includes many more. All papers and documents analysed are reported in the Bibliography section. The papers we examined just for reference purposes were not mentioned in the spreadsheet. In the first sheet of the Excel file, named “List of Papers”, we listed all the papers we were going to analyse, along with their year of publication, number of citations, and place of publication. We also included notes and provided the online reference link.
The number of citations is updated to the 2nd of June 2023.
The next tab in the file, labelled “Table”, displays the table created to summarize the information gathered from papers, and, to be more explicit, reiterate the related details mentioned above. For each item, we collected the most important definitions and explanations concerning each model or metric referred in. This work was time-consuming, but it helped identifying the most cited authors as well as the most commonly used and mentioned models or metrics.

As soon as we need to get material about an entity, which could be a model or a metric, we could find it all on the same row, and so have a straight reference to the paper in which it was mentioned.

The tabs “Models_Papers” and “Metrics_Papers” take into analysis which models and metrics were covered in the texts, checking for each of them if it was mentioned or described. The analysis is covered by twenty-seven texts, including papers, websites and books. Some sources have been excluded from the count, such as the book “*Metrics and Models in Software Quality Engineering”* by Stephen Kan. It was not taken into consideration because the only part available to us was the first chapter, where FURPS and CMM are hardly mentioned. Moreover, the book “*Ingegneria del Software - Creatività e Metodo”* [60] was analysed only in the second chapter "*La Qualità del Software*", where the model ISO 9126 is mentioned in many parts of the text, and there is a short description of few metrics, such as LOC, Function Point, and just a few generic hints to object-oriented metrics and the number of errors metric.

Not all models or metrics have been mentioned or described in this study, but most of them are available in the sheet “Full List”, together with the reference to the original report (not all of them are available for free). The list has been made available to easily reach links for future work.

4.2 Results

The graph in Figure 13 shows part of the results obtained, and it illustrates that the most common model is ISO 9126, which is mentioned and described in nineteen texts out of twenty-eight. It is followed by McCall and soon after Boehm.

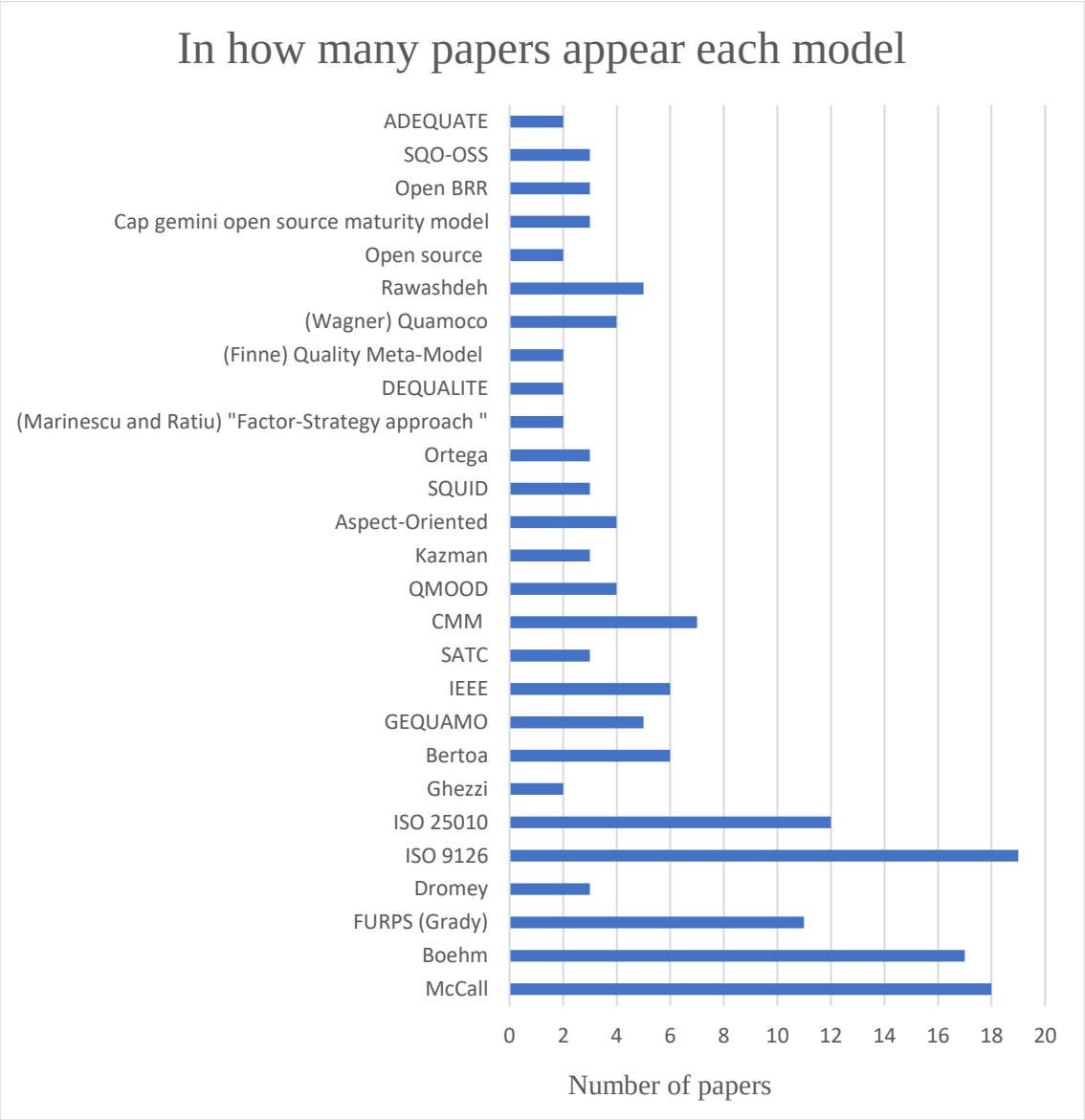

Figure 13 Graph that represents the Number of References of each Model in Papers

IEEE is one of the most cited standards, but just a few texts analyse it as a model. Moreover, it's possible to see that the paper which describes almost every model is *"Analysis and Assessment of Existing Software Quality Models to Predict the Reliability of Component-Based*" [13] by Yadav Shivani and Kishan Bal, published in June; however the paper, according to Google Scholar, has been cited only sixteen times (data updated on 2nd June 2023).

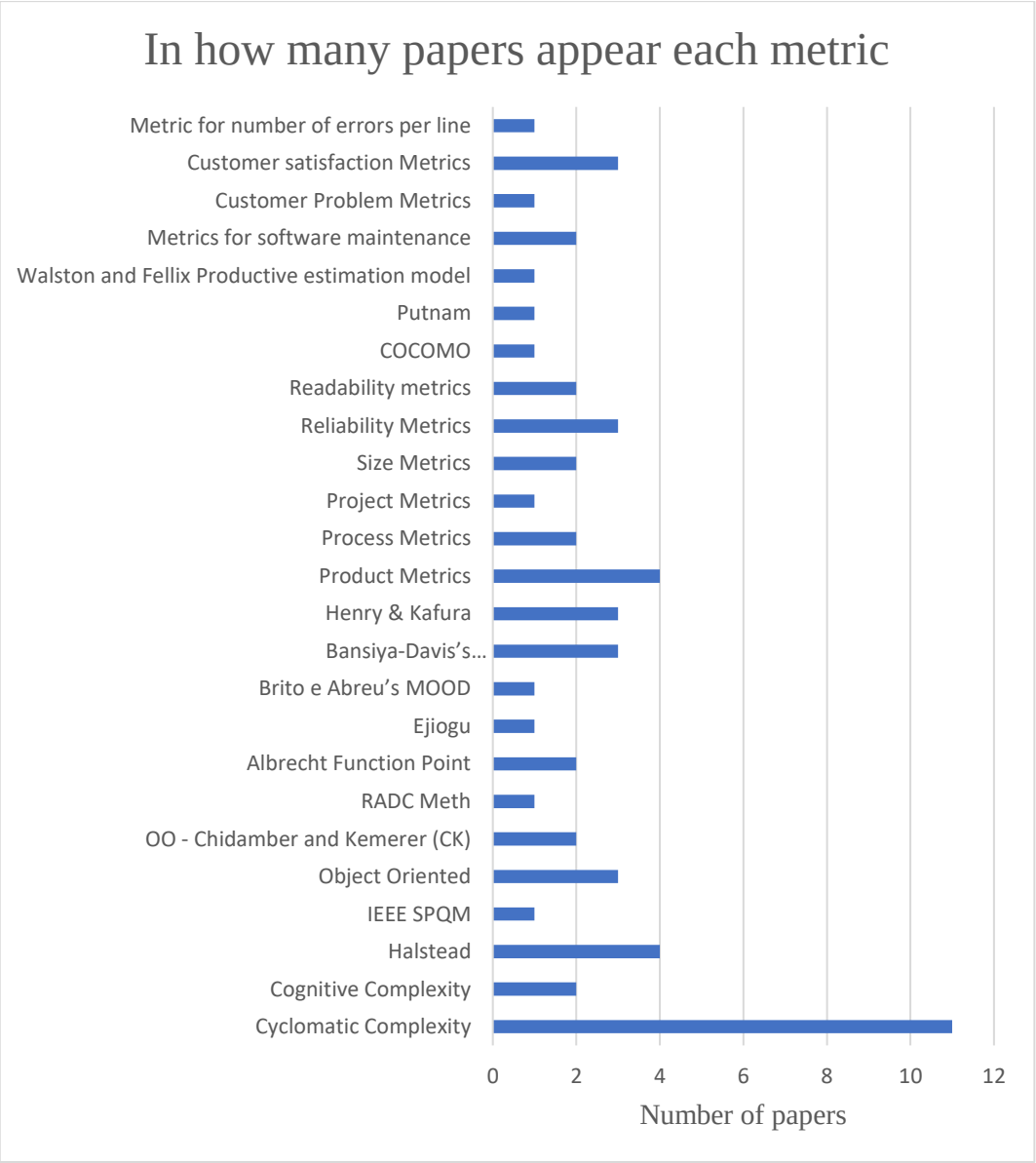

Figure 14 Graph that represents the Number of References of each Metric in Papers

The same work has been done for what concerns metrics, as represented in Figure 14, and the result shows that Cyclomatic Complexity seems to be the most popular among papers analysed, followed by LOC.

The paper which covers the majority of the metrics is "*Software Quality Factors and Software Quality Metrics to Enhance Software Quality Assurance*" by Ming-Chang Lee, published in 2014 in the British Journal of Applied Science & Technology and with sixty-three citations (on 2nd June 2023).

4.3 Summary

In this chapter we showed how the study has been conducted, beginning with data collection, analysis, and the final results obtained.

The data was gathered from several online scientific journals, a few websites and books. The information was then summarized in an Excel file and then got results through graphs. The graphs show that among the papers analysed, even if the ISO models (both 9126 and 25010) have been released several years ago, they are still the most mentioned and complete of characteristics, and this is the reason why they often are the basis for implementing newer models. With the advancement of years, it can be noted an increase in the number of characteristics required by each model: this reflects the evolution of technology and software and the change in customer requirements and needs over time.

5 Conclusion

In this thesis, we analysed different kinds of scientific papers and other sources regarding software quality models and metrics, ranging from 1978 to 2023. The research has been conducted through Google Scholar, a few websites, books, ACM, and other scientific online journals. For the ongoing research on the assessment of software quality around thirty articles have been selected for the study, in addition to the papers accessed solely for reference or citation purposes. The selection method was based on the relevancy of keywords, the highest possible number of citations and the most recent publication year. We might have missed some other studies where authors used terms different from the keys mentioned. At the current state of the art, it seems that no complete survey on the assessment of software quality is available online. Therefore, both the thesis and the Excel spreadsheet used for data collection are open-source and can be accessed on the GitHub platform at https://github.com/fmaira/SWQualityModelsMetrics. The thesis has been redacted in the English language to be used as a basis for future research.

The study concerns software quality models and metrics, focusing on those that have served as the foundation for newer ones. It is important to consider the rapid progress of software evolution, which introduces new technologies and consequently necessitates new evaluations for quality assessment.

Although numerous models have been developed, the most popular during the first decade of the XXI century continues to be the ISO 9126, which outperforms its successor ISO 25010; the latter model seems to be more complex and abstract with some criteria overlapping or ambiguous; moreover, some of its concept lack of clear definition; ISO 9126, on the other hand, is simpler to follow, it has been the subject of several studies for a long time resulting in a wide range of available material for consultation; additionally, many organizations could already be accustomed to its usage. However, considering the continuous evolution of software, it is expected that ISO 25010 will gain more popularity over time. Nevertheless, both models are often used as the basis for developing new models. In the subsequent decade, it seems that both these models started to decline, probably influenced by the evolution of new kinds of software, such as COTS, and OSS, among others [16]. This point of view could be the subject of further investigation for future research.

Relating to the metrics, even if the Cyclomatic Complexity is the most widespread among papers, it seems that its extension Cognitive Complexity is the more reliable in evaluating

the factor of Understandability. However, we could not affirm that there is a universal metric most widely used since the choice of a metric depends on the context and on which aspects we would like to assess.

Colakoglu et al. [43] provide a systematic mapping study based on seventy articles and papers from 2009 to 2019, showing results through explanations and graphics. The study reports that, even if the literature offers a large quantity of security-related metrics, the system is not completely safe yet, and it is required to implement more reliable security metrics.

Based on the papers, it can be deduced that the factor most researched by the end user is reliability. In 2023 Berlas [4] claims that currently the approach to measuring reliability is useful only for the last phases of development, at the end of the coding step or during testing, too late to improve reliability at these steps. Rizli et al. [61] introduced the "Fuzzy Logic based Software Reliability Quantification", a framework for quantifying reliability based on requirement and design stage measurement. A short brief of other metrics assessing reliability, such as Huang and Huang or Serban and Shaiks could be found in [4].

For further work and studies, EOSC [2], the European Open Science Cloud provides an environment which host and process research data to support EU science. It is a virtual, open, and safe cloud where the community can store, share and reuse scientific data and results. They set up a task force on Infrastructures for Quality Research Software, involved in promoting and improving high-quality software research. They aim to encourage the development and spread of tools and services concerning software research, enhancing it from both organisational and technical perspectives. They also identify and collect best practices to write quality research software, together with determining qualitative and quantitative methodologies to realize an impartial and objective assessment, recognising those researchers and engineers who developed quality research software projects.

6 Bibliography

[1] S. Kan, Metrics and Models in Software Quality Engineering, Addison Wesley, 2003.

[2] «EOSC - The European Open Science Cloud,» [Online]. Available: https://eosc.eu/. [Consultato il giorno 24 06 2023].

[3] P. A. Laplante e M. Kassab, What Every Engineer Should Know about Software Engineering, CRC Press, 2022.

[4] M. F. Berlas, «A Review Report on Software Quality Measurement and Estimation,» 2023. [Online]. Available: https://www.techrxiv.org/articles/preprint/A_Review_Report_on_Software_Quality_Measurement_and_Estimation/22058885/1.

[5] J. E. Gaffney, «Metrics In Software Quality Assurance,» in *ACM SIGMETRICS Workshop/Symposium on Measurement and Evaluation of Software Quality*, 1981.

[6] «IEEE Standard Glossary of Software Engineering Terminology,» *IEEE Std 610.12-1990*, pp. 1-84, 1990.

[7] E. Ronchieri e M. Canaparo, «Assessing the Impact of Software Quality Models in Healthcare Software Systems,» *Taylor and Francis Online* , p. 14, 2023.

[8] A. Rawashdeh e B. Matalkah, «A New Software Quality Model for Evaluating COTS Components,» *Journal of Computer Science*, 2006.

[9] S. Rakitin, Software Verification and Validation for Practitioners and Managers, 2nd ed. a cura di, Norwood, MA (USA): Artech House, 2001.

[10] E. Pirri, «Un Metodo di Analisi Statica di Qualità del Software,» Bologna, 2020.

[11] E. Georgiadou, «GEQUAMO—A Generic, Multilayered, Customisable, Software Quality Model,» *Software Quality Journal*, 2003.

[12] M. C. Lee, «Software Quality Factors and Software Quality Metrics to Enhance Software Quality Assurance,» *British Journal of Applied Science & Technology*, 2014.

[13] S. Yadav e B. Kishan, «Analysis and Assessment of Existing Software Quality Models to Predict the Reliability of Component-Based,» *International Journal of Emerging Trends in Engineering Research*, 2020.

[14] S. S. Thapar, P. Singh e S. Rani, «Challenges to the Development of Standard Software Quality Model,» *International Journal of Computer Applications*, 2012.

[15] J. P. Miguel, D. Mauricio e G. Rodríguez, «A Review of Software Quality Models for the Evaluation of Software Products,» *International Journal of Software Engineering & Applications*, p. 24, 2014.

[16] I. G. Ndukwe, S. A. Licorish, A. Tahir e S. G. MacDonell, «How have views on Software Quality differed over time? Research and Practice Viewpoints,» *The Journal of Systems & Software*, 2023.

[17] J. P. Cavano e J. A. McCall, «A Framework for the Measurement of Software Quality,» 1978. [Online].

[18] A. Seffah, N. Kececi e M. Donyaee, «QUIM: A Framework for Quantifying Usability Metrics in Software Quality Models,» 2001.

[19] P. Nistala, K. V. Nori e R. Reddy, «Software Quality Models: A Systematic Mapping Study,» *IEEE/ACM International Conference on Software and System Processes (ICSSP)*, p. 10, 2019.

[20] S. Agrawal, «Software Application Development Company,» iFour Consultancy, 2016. [Online]. Available: http://softwareindustries-ifour.blogspot.com/2016/04/types-of-software-quality-models.html. [Consultato il giorno 06 04 2023].

[21] W. Y. N. W. Z. Abidin e Z. Mansor, «The Criteria for Software Quality in Information System: Rasch Analysis,» *International Journal of Advanced Computer Science and Applications*, vol. 10, n. 9, 2019.

[22] R. Doneva, S. Gaftandzhieva, Z. Doneva e N. Staevsky4, «Software Quality Assessment Tool,» *International Journal of Computer Science and Mobile Computing*, vol. 4, 2015.

[23] S. Wagner, K. Lochmann, S. Winter, F. Deissenboeck, E. Juergens, M. Herrmannsdoerfer, C. Schubert, L. Heinemann, M. Klaes, A. Trendowicz e J. Heidrich, «The Quamoco Quality Meta-Model,» in *ICSE '12: Proceedings of the 34th International Conference on Software Engineering*, 2012.

[24] S. Wagner, A. Goeb, L. Heinemann, M. Klaes, C. Lampasona, L. Lochmann, A. Mayr, R. Ploesch, A. Seidl, J. Streit e A. Trendowicz, «Operationalised product

quality models and assessment: The Quamoco approach,» *Information and Software Technology,* vol. 62, pp. 101-123, 2015.

[25] J. Bansiya e C. Davis, «A Hierarchical Model for Object-Oriented Design Quality Assessment,» *IEEE Transactions on Software Engineering,* vol. 28, pp. 4-17, 2002.

[26] T. Galli, F. Chiclana e F. Siewe, «Software Product Quality Models, Developments, Trends, and Evaluation,» *SN Computer Science,* p. 24, 2020.

[27] Suman e M. Wadhwa, «A Comparative Study of Software Quality Models,» *International Journal of Computer Science and Information Technologies,* vol. 5, 2014.

[28] National Institute of Standards and Technology U.S. Department of Commerce, «Computer Security Resource Center,» [Online]. Available: https://csrc.nist.gov/glossary/term/commercial_off_the_shelf. [Consultato il giorno 16 04 2023].

[29] M. Bertoa e A. Vallecillo, «Quality Attributes for COTS Components,» in *In the Proceedings of the 6th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering*, 2002.

[30] A. Alvaro, E. S. De Almeida, A. M. De Vasconcelos e S. R. De Lemos Meira , «Towards a Software Component Quality Model,» in *5th International Conference on Quality Software*, 2005.

[31] K. Sheoran e P. Sangwan, «An Insight of Software Quality Models Applied in Predicting Software Quality parameters: A Comparative Analysis,» in *2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO)*, Noida, India, 2015.

[32] M. Ortega, M. Perez e T. Rojas, «Construction of a Systemic Quality Model for Evaluating a Software Product,» *Software Quality Journal,* pp. 219-242, 2003.

[33] N. Callaos e B. Callaos, «Designing with Systemic Total Quality,» in *Proceedings of the International*, Orlando, USA, 1996.

[34] «History of the OSI,» October 2018. [Online]. Available: https://opensource.org/history/.

[35] A. Adewumi, S. Misra e N. Omoregbe, «A Review of Models for Evaluating Quality in Open Source Software,» *ScienceDirect,* vol. 4, pp. 88-92, 2013.

[36] I. Samoladas, G. Gousios, D. Spinellis e I. Stamelos, «The SQO-OSS Quality Model: Measurement Based Open Source Software Evaluation,» in *IFIP – The International Federation for Information Processing* , vol. 275, Boston, Springer, 2008.

[37] G. A. Campbell, «Cognitive Complexity, a new way of measuring understandability,» 05 04 2021. [Online]. Available: https://www.sonarsource.com/docs/CognitiveComplexity.pdf. [Consultato il giorno 26 05 2023].

[38] K. Haaland, A. Groven, R. Glott e A. Tannenberg, «Free/libre open source quality models-a comparison between two approaches,» *citeseerx.ist.psu.edu,* pp. 1-17, 2010.

[39] B. Kitchenham, L. S. e A. Pasquini, «The SQUID Approach to Defining a Quality Model,» *Software Quality Journal,* vol. 6, pp. 211-233, 1997.

[40] R. Kazman, L. Bass, G. Abowd e M. Webb, «SAAM: a Method for Analyzing the Properties of Software Architectures,» in *Proceedings of 16th International Conference on Software Engineering*, Sorrento, Italy, 2002.

[41] «IEEE Standard for a Software Quality Metrics Methodology,» *IEEE Std 1061-1998,* 1998.

[42] H. R. Bhatti, «Automatic Measurement of Source Code Complexity,» 2011.

[43] F. N. Colakoglu, A. Yazici e A. A. Mishra, «Software Product Quality Metrics: A Systematic Mapping Study,» *IEEE Access,* p. 24, 2021.

[44] T. J. McCabe, «A Complexity Measure,» *IEEE Transactions on Software Engineering,* 12 1976.

[45] M. Halstead, «Software Science: Theoretical and Practical Considerations,» *Communications of the ACM,* 1977.

[46] M. H. Halstead, Elements of Software Science, Elsevier , 1977.

[47] A. Albrecht, «Measuring Application Development Productivity,» in *Proceedings of IBM Applications Development Symposium,* Monterey (California, USA), 1979.

[48] N. Nagappan e T. Ball, «Use of Relative Code Churn Measures to Predict System Defect Density,» in *ICSE '05: Proceedings of the 27th international conference on Software Engineering*, 2005.

[49] R. E. Park, «Software Size Measurement: A Framework for Counting Source Statements,» 1992.

[50] T. McCabe, *Software Quality Metrics to Identify Risk*, 2008.

[51] D. R. Wallace, A. H. Watson e T. J. McCabe, «Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric,» Dolores R. Wallace Editor, 1996.

[52] V. Lenarduzzia, A. Janesc e T. Kilamo, «Does Cyclomatic or Cognitive Complexity Better Represents Code Understandability? An Empirical Investigation on the Developers Perception,» *Journal of Systems and Software*, p. 10, 2023.

[53] M. M. Barón, M. Wyrich e S. Wagner, «An Empirical Validation of Cognitive Complexity as a Measure of Source Code Understandability,» in *International Symposium on Empirical Software Engineering and Measurement*, 2020.

[54] S. Scalabrino, G. Bavota, C. Vendome, M. Linares-Vasquez, D. Poshyvanyk e R. Oliveto, «Automatically assessing code understandability,» *IEEE Transactions on Software Engineering*, vol. 47, n. 3, pp. 595-613, 2019.

[55] T. Bowen, B. Gray e T. Jay, *RADC-TR-85-37, RADS, Griffiss Air Face Base N. Y*, New York, 1985.

[56] E. B. Belachew, F. A. Gobena e S. T. Nigatu, «Analysis of Software Quality Using Software Metrics,» *International Journal on Computational Science & Applications*, vol. 8, n. 4-5, 2018.

[57] «Home of Clean Code,» SonarSource , [Online]. Available: https://www.sonarsource.com/products/sonarqube/. [Consultato il giorno 27 05 2023].

[58] M. Abdallah, T. Jaber, N. Alabwaini e A. A. Alnabi, «A Proposed Quality Model for the Internet of Things Systems,» in *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, 2019.

[59] D. Franke, S. Kowalewski e C. Weise, «A Mobile Software Quality Model,» in *2012 12th International Conference on Quality Software*, 2012.

[60] A. Binato, A. Fuggetta e L. Sfardini, Ingegneria Del Software - Creatività e Metodo, Person Addison Wesley, 2006.

[61] S. W. Rizvi, V. K. Singh e R. A. Khan, «Fuzzy Logic Based Software Reliability Quantification Framework: Early Stage Perspective (FLSRQF),» *Procedia Computer Science*, vol. 89, pp. 359-368, 2016.

[62] B. A. Siket I., «Differences in the Definition and Calculation of The LOC Metric in Free Tools,» 2014.

[63] G. Taentzer, M. Goedicke, B. Paech, K. Schneider, A. Schürr e B. Vogel-Heuser, «The Nature of Software Evolution,» in *Managed Software Evolution*, Springer Open, 2019.

[64] H. Barkmann, R. Lincke e W. Löwe, «Quantitative Evaluation of Software Quality Metrics in Open-Source Projects,» in *23rd International Conference on Advanced Information Networking and Applications*, 2009.

[65] A. Alvaro, E. S. De Almeida e S. R. De Lemos Meira, «A Software Component Quality Framework,» *ACM* , 2010.

[66] «Software Quality Management,» tutorialspoint, [Online]. Available: https://www.tutorialspoint.com/software_quality_management/software_quality_management_introduction.htm. [Consultato il giorno 05 05 2023].

[67] G. R. Dromey, «A Model for Software Product Quality,» *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, vol. 21, n. 2, 1995.

[68] «ISO/IEC 25010:2011,» 03 2011. [Online]. Available: https://www.iso.org. [Consultato il giorno 16 05 2023].