Democratic and Popular Republic of Algeria
Ministry of Higher Education and Scientific Research
University of Oum El Bouaghi

Faculty of Exact Sciences and Sciences of Nature and Life
Department of Mathematics and Computer Science
Research Laboratory on Computer Science's Complex Systems

Quality Assurance For Embedded Systems

To obtain the degree of
DOCTOR in Computer Science
(Option: **Imagery for Embedded Systems**)

Thesis presented by:
TAMRABET Zouheyr

Before the jury composed of:

President	Pr. BOUTEKKOUK Fateh	Professor	University of Oum El Bouaghi
Supervisor	Pr. MOKHATI Farid	Professor	University of Oum El Bouaghi
Co-Supervisor	Dr. MARIR Toufik	MCA	University of Oum El Bouaghi
Examiner	Dr. BENABOUD RohAllah	MCA	University of Oum El Bouaghi
Examiner	Dr. HOUASSI Hichem	MCA	University of Khenchela
Examiner	Dr. SOUIDI Mohammed El Habib	MCA	University of Khenchela

2021-2022

Acknowledgment

"First and foremost, praises and thanks to ALLAH the Almighty"

I would like to express my deep and sincere gratitude to my thesis supervisor **Pr. MOKHATI Farid**. This work would never be completed without such a positive person, who sees possibilities even in the simplest attempts. There are not enough words of appreciation to give him the thanks he deserves. Again, thanks a lot professor.

A special appreciation and gratitude to my thesis co-supervisor **Dr. Marir Toufik**, not only for his contributions to this work but also for all the time we have worked together in scientific research since my early days in the Department of Computer Science. I learned a lot from you professor, and I consider myself very lucky to have this opportunity. Not to mention your modesty, generosity, and personality are one of a kind. For this, thank you very much.

I would like to thank all the members of the Jury,

Pr. BOUTEKKOUK Fateh

Dr. BENABOUD RohAllah

Dr. HOUASSI Hichem

Dr. SOUIDI Mohammed El Habib

For the great honor they have gave me by agreeing to examine this work and to provide constructive criticism and valiant comments.

Finally, I would like to thank **Dr. Kalache Ayyoub**, **Dr. Chebout Mohamed Seddik,** and **Mr. Zinai Housseyn** for their quality discussions during the accomplishment of this work.

Dedication

In memory of my mother

To my father

To my beloved sister

To my brothers, nephews and nieces

To my supportive wife who believed in me more than I believed in myself

To my son: Tamim.

الملخص

البحث المقدم في هذه الأطروحة يعالج إشكالية ضمان جودة الأنظمة المدمجة. الدافع الرئيسي لمساهمتانا هو غياب نماذج الجودة القياسية التي تشمل وتغطي المفاهيم الخاصة بالأنظمة المدمجة. بالإضافة لأهمية تمثيل جودة الأنظمة المدمجة لما لها من تأثير على تعقيد ومصداقية هذا النوع من الأنظمة. مساهمتانا في هذا البحث تركز على تمثيل جودة البرمجيات في سياق الأنظمة المدمجة. من خلال دراستنا هذه، قمنا باقتراح نموذج جودة خاص ببرمجيات الأنظمة المدمجة ESQuMo الذي يرتكز على نموذج الجودة القياسي ايزو 25010. تكمن أهمية النموذج المقترح في القدرة على تغطية المميزات الخاصة بالبرمجيات المدمجة، بالإضافة للمميزات العامة التي تشترك فيها البرمجيات المدمجة مع باقي البرمجيات العادية. كخطوة ثانية في مساهمتنا، قمنا بتطبيق نموذج الجودة المقترح ESQuMo على فئة البرمجيات المدمجة للتصوير الطبي، وذلك من خلال تقديم مجموعة من مقاييس الجودة التي تتوافق مع التقييس ايزو 25023 المكمل للتقييس 25010 المعتمد كأساس لنموذج الجودة المقترح. تهدف مقاييس الجودة هذه بشكل أساسي في التحكم والإشراف على الجودة العامة للأنظمة المدمجة.

كلمات مفتاحية: ضمان الجودة، الأنضمة المدمجة، نماذج الجودة، مقاييس الجودة، التعقيد، المصداقية، ايزو 25010، ايزو 25023.

Abstract

The research presented in this thesis addresses the problem of embedded systems quality assurance. The main motivation for our contributions is the absence of standard quality models that include and cover concepts specific to embedded systems. In addition to the importance of representing the quality of embedded systems because of its impact on the complexity and dependability of these types of systems. Our contributions in this thesis focus on the representation of software quality in the context of embedded systems. Through our study, we have proposed an Embedded Software Quality Model ESQuMo that is based on the standard ISO/IEC 25010 quality model. The importance of the proposed model lies in the ability to cover the relevant characteristics of the embedded software, in addition to the general characteristics that the embedded software share with the rest of the ordinary software. As a second step in our contribution, we have applied the proposed ESQuMo quality model to the medical imaging embedded software category, by providing a set of quality measures that comply with the ISO/IEC 25023 the complementary standard to the ISO/IEC 25010 adopted as the basis for the proposed quality model. These quality measures are primarily intended to control and supervise the overall quality of embedded systems.

Keywords: Quality Assurance, Embedded Systems, Quality Models, Quality Measures, Complexity, Dependability, ISO/IEC 25010, ISO/IEC 25023.

Résumé

La recherche présentée dans cette thèse aborde le problème de l'assurance qualité des systèmes embarqués. La motivation principale de nos contributions est l'absence des modèles de qualité standard qui incluent et couvrent des concepts spécifiques aux systèmes embarqués. En outre, l'importance de représenter la qualité des systèmes embarqués en raison de son impact sur la complexité et la fiabilité de ces types de systèmes. Nos contributions dans cette thèse se concentrent sur la représentation de la qualité logicielle dans le contexte des systèmes embarqués. Au cours de notre étude, nous avons proposé un modèle de qualité du logiciel embarqué ESQuMo qui est basé sur le modèle de qualité standard ISO/IEC 25010. L'importance du modèle proposé réside dans la capacité à couvrir les caractéristiques pertinentes du logiciel embarqué, en plus des caractéristiques générales que le logiciel embarqué partage avec le reste des logiciels ordinaires. Dans une deuxième étape de notre contribution, nous avons appliqué le modèle de qualité ESQuMo proposé à la catégorie des logiciels embarqués d'imagerie médicale, en fournissant un ensemble de mesures de qualité conformes au standard ISO/IEC 25023, le standard complémentaire à l'ISO/IEC 25010 adoptée comme la base du modèle de qualité proposé. Ces mesures de qualité visent principalement à contrôler et superviser la qualité globale des systèmes embarqués.

Mots clés : Assurance Qualité, Systèmes Embarqués, Modèles de Qualité, Mesures de Qualité, Complexité, Dépendabilité, ISO/IEC 25010, ISO/IEC 25023.

Table of contents

General Introduction ***13***
1. General Context and Problem Statement 2
2. Contributions 3
3. Thesis Plan 5
Chapter 01 - Embedded Systems ***7***
1. Introduction 8
2. Definitions 9
3. Components of Embedded Systems 11
4. Features of Embedded Systems 13
5. Application fields 16
6. Real Time Embedded Systems 18
7. Specification of embedded systems 19
7.1. Structural representations 20
7.1.1. Topological representations: 20
7.1.2. Relational representations 21
7.2. Behavioral representations 22
7.2.1. Equations 23
7.2.2. Finite State Machines 24
7.2.3. Petri Nets 25
7.3. Communication-based models 27
7.3.1. Discrete Event Models 27
7.3.2. Synchronous Event Models 28
7.3.3. Asynchronous Message Passing 28
7.3.4. Synchronous Message Passing 28
8. Design of Embedded Systems 28
8.1. Classical design flow of embedded systems 28
8.2. Co-design of embedded systems 30
9. Conclusion 31
Chapter 02 - Software Quality Assurance ***32***
1. Introduction 33
2. Software Quality Assurance 33
3. Software Quality Modeling 36
3.1. The DAP Classification 37
3.2. Quality of Specific Software Products 39
3.3. Quality Models 41

3.3.1. McCall's Quality Model 42
3.3.2. Boehm's Quality Model 43
3.3.3. Dromey's Quality Model 44
3.3.4. ISO/IEC 9126 Quality Model 46
4. SQauRE Series of Standards 48
4.1. The ISO/IEC 25010 Quality Model 51
5. Conclusion 54
Chapter 03 - Quality Assurance for Embedded Systems 55
1. Introduction 56
2. Embedded Software Quality 57
3. Embedded Software Quality Attributes and Quality Models 59
4. Comparative Study 65
5. Conclusion 69
Chapter 04 - ESQuMo an Embedded Software Quality Model 70
1. Introduction 71
2. Quality Model for Embedded Software 72
2.1. Investigating quality models and quality attributes of embedded software 73
2.2. Identifying the relevant characteristics of embedded software 75
2.2.1. Complexity 76
2.2.2. Dependability 78
2.2.3. Discussion 80
2.3. ESQuMo: an Embedded Software Quality Model 82
3. Comparative study 84
4. Applying the ESQuMo Quality Model to Medical Imaging Embedded Software 86
4.1. Modalities of Medical Imaging 87
4.1.1. X-ray 87
4.1.2. CT scan 88
4.1.3. Ultrasound 88
4.1.4. MRI 88
4.2. Quality Measurements of Medical Imaging Embedded Software 89
4.3. Specific Sub-characteristics of Medical Imaging Embedded Software 90
4.3.1. Cost of Use 90
4.3.2. Vulnerability of Data 91
4.4. Extended ISO/IEC 25010 Sub-Characteristics 91
4.4.1. Time Behaviour 91
4.4.2. Resource Utilization 92

4.4.3. Maturity 93
4.4.4. Fault Tolerance 94
4.5. Format of the Proposed Quality Measures 94
5. Conclusions 95
Conclusion and Future Works 97
1. Conclusion and Perspectives 98
2. Future Works 99
References 101

List of Figures

Figure 1.1: Relationship between embedded systems and CPS (Marwedel, 2021).....11
Figure 1.2: Scheme of a typical Embedded System (Serpanos et al., 2011). 12
Figure 1.3: Real Time Embedded Systems.....19
Figure 1.4: A topological representation of a system. 21
Figure 1.5: Data flow of an equation representation.....23
Figure 1.6: Example of a deterministic FSM.....24
Figure 1.7: Example of a non-deterministic FSM. 25
Figure 1.8: Example of petri nets representation. 26
Figure 1.9: Extension of petri nets representation with weighted transitions.....27
Figure 1.10: The classical design flow of embedded systems. 29
Figure 1.11: Co-design of embedded systems. 30
Figure 2.1: Definition, Assessment, and Prediction Classification for quality models. 38
Figure 2.2: Definition, Assessment and Prediction in COQUAMO.....39
Figure 2.3: McCall's Quality Model (McCall, 1977). 42
Figure 2.4: Boehm's Quality Model (Boehm et. al, 1976).....44
Figure 2.5: Dromey's Quality Model (Dromey, 1995). 45
Figure 2.6: ISO/IEC 9126 internal and external quality model. 47
Figure 2.7: ISO/IEC 9126 quality in use model. 48
Figure 2.8: Organization of the SQuaRE (ISO, 2005a). 49
Figure 2.9: Software product quality measurement reference model of ISO/IEC 25020 (Wagner, 2013). 50
Figure 2.10: ISO/IEC 25010 quality in use model. 52
Figure 2.11: The meta-model of the ISO/IEC 25010 quality models (Wagner, 2013). 52
Figure 2.12: ISO/IEC 25010 product quality model.....53
Figure 4.1: The Dependability classification (Avižienis et al, 2004). 79
Figure 4.2: The Dependability attributes (Laprie, 1992). 79
Figure 4.3: ESQuMo: an Embedded Software Quality Model (Tamrabet et. al, 2022a). 83
Figure 4.4: SQuaRE general reference model (ISO, 2005). 89

List of Tables

Table 2.1: Classification of quality models (Tamrabet et. al, 2018).....41
Table 3.1: Quality attributes of embedded systems (Tamrabet et. al, 2018). 64
Table 3.2: Comparative study (Tamrabet et. al, 2018). 68
Table 4.1: Results of the investigation on quality models and quality attributes (Tamrabet et al., 2022a). 74
Table 4.2: Results of the comparative study (Tamrabet et. al, 2022a). 84
Table 4.3: Quality measures of medical imaging embedded software (Tamrabet et. al, 2022b).. 95

General Introduction

1. General Context and Problem Statement

Nowadays we are living in a world expanded with countless computer systems in various application fields. These systems were previously massive and big-sized computers, while now they are shrunk down to the size of pocket devices. This phenomenon is the result of the technological advancements in many scientific aspects, as well as the continuous needs of the users that seek perfection and quality.

Although the physical presence of these systems is important in our daily life, the software as an intangible part of such systems is important as well since it is integrated into every daily usage system. Moreover, software products represent essential components, especially for complex and critical systems. Thus, the development of low-quality software is not only discouraged but sometimes becomes intolerant. Consequently, quality management represents an important activity in the software development process. Contrary to presumptions that consider quality management a post-implementation activity, this activity covers the entire development process (Pressman, 2010).

Quality management requires first and foremost the specification of the concept quality. This viewpoint is backed by the ambiguity, complexity, and variety of features that characterize software quality. To address these problems, it is essential to model this notion in order to provide better software quality management. Quality models, in particular, can contribute to the specification, evaluation, and/or prediction of quality (Wagner, 2013).

The concentration of early studies of software quality to the proposal of quality models demonstrates the importance of quality modeling. As a result, various models have been presented, including the McCall et al. (McCall, 1977) model and the Boehm model (Boehm et al., 1978). These models illustrate the various characteristics and attributes that determine software quality.

The variety of quality models is a logical result of the diversity of views on this concept. However, this variety is a major setback when it comes to exchanging results or comparing different products. Naturally, using different models to specify or evaluate the quality of a software product will yield different results. Therefore, it is important to propose a consistent quality model that unifies the various points of view. The

international standardization organization has proposed many standards across many areas, and software products are no exception. A standard quality model for software products ISO/IEC 9126 is proposed , as well as its revision the ISO/IEC 25010 (ISO, 2011a).

The introduction of the first quality models and their standardization is the initial steps toward the evolution of this field. The nature of software products has completely changed since the first models emerged at the end of the 1970s. This massive change is the result of the emergence of new paradigms, principles, concepts, and methodologies that contribute to the evolution of the software engineering field. Clearly, quality models must take into account these novelties introduced, as well as the specificities of the diverse products such as the embedded software.

Although the quality of software represents an interesting track in the field of software engineering, very few works have addressed the issue of the quality of embedded software. In addition, the rare attempts to model the quality of embedded software are limited because they omit the fundamental characteristics of software quality, on the one hand. On the other hand, they omit the specificities of this embedded software as particular software.

We believe that the field of embedded systems is quite important and requires the study of the quality of embedded software. This is justified by several reasons such as the criticality of these systems. However, we are convinced that the quality of embedded software has not yet been addressed profoundly. A quality model for embedded software must present in a unified and coherent view the common characteristics of software quality with the inherent specificities of embedded software. In this context, the problem of our thesis emerges from the absence of a quality model of embedded systems with this unified view. Such a model represents the basic building block for the quality management of embedded systems.

2. Contributions

Many works dealing with quality models and quality attributes have emerged in the context of embedded software (Choi et al., 2008; Carvalho & Meira, 2009; Guessi et al., 2012; Jeong & Kim, 2012; Ahrens et al., 2013; Oliveira et al., 2013; Nakagawa et al., 2013; Jeong et al., 2014; Bianchi et al., 2015; Kiran and Simons, 2016; Garcés et

al., 2017). As part of the study of the quality of embedded systems, we have classified the various works carried out in this field according to two axes. On the one hand, numerous works proposed a set of quality characteristics based on their definition of quality and the application area of the targeted products (Guessi et al., 2012; Oliveira et al., 2013; Nakagawa et al., 2013; Bianchi et al., 2015; Kiran and Simons, 2016; Rahman et al,. 2018; Liebel et al,. 2018; Vegendla et al,. 2018; Akdur et al,. 2018; Mohsin and Janjua, 2018; Kaur and Singh, 2019; Cadavid et al,. 2020; Sales and Becker, 2021). On the other hand, several works extended previous models to make them more suitable for embedded systems. (Choi et al., 2008; Carvalho & Meira, 2009; Jeong & Kim, 2012; Ahrens et al., 2013; Jeong et al., 2014; Garcés et al., 2017). Furthermore, the study of these different works allows us to identify the major shortcomings that affect this field. These shortcomings can be summarized in the absence of a global quality model that highlights the relevant attributes affecting the quality of embedded systems, on the one hand. On the other hand, it covers the generic characteristics of the software quality in addition to the previous specific characteristics. We believe that the existence of quality models of embedded software, which unify and cover at the same time the common characteristics of the software, as well as the specificities of embedded software, represents the steppingstone for the development of secure, safe, and reliable embedded systems. Our contributions proposed in the context of this thesis can be organized around three axes:

- **The identification of quality attributes of embedded systems**

  With the intention of identifying the shortcomings in the context of embedded systems, we present our first contribution, a survey of the most important works related to embedded software quality engineering (Tamrabet et. al, 2018). As mentioned previously, the choice of this context is justified by the criticality of embedded systems, in which the absence of quality may lead to serious consequences. In addition, a detailed description of each work is also provided, along with deep analyses and discussions. Finally, a comparative study is also available at the end of this contribution to demonstrate the strengths and weaknesses of each work. Eventually, we believe that the findings of this first contribution and the identification of embedded software quality attributes are of prime importance in developing an embedded software quality model.

- **Modeling the quality of the embedded software**

In order to develop a quality model specific to embedded systems, we took advantage of existing standard models. Our second contribution consists of the development of a quality model for embedded software called ESQuMo (Embedded Software Quality Model) based on the ISO/IEC 25010 standard model (Tamrabet et. al, 2022a). The proposed model offers the advantage of combining the common characteristics of software quality defined in the standard model, as well as the inherent characteristics of embedded software.

- **The application of the embedded software quality model**

As part of the application of our proposed model ESQuMo, we chose medical imaging systems as a typical example of embedded systems in the third contribution (Tamrabet et. al, 2022b). Then, we proposed specific measures that comply with the ISO/IEC 25023 standard. This step is very interesting, showing the capacity of our model not only to define quality but also to be directed in several paths to support other quality objectives such as the evaluation and prediction of quality.

3. Thesis Plan

This thesis is organized into four chapters. The first two chapters represent the state of the art in domains related to our problem. Then, we present our contributions in the next two chapters. Thus, this manuscript is composed of the following chapters:

Chapter 01 - Embedded Systems: In this chapter, terms and definitions related to embedded systems will be presented. In addition to presenting the components, the features, the application areas as well as the various representations used in the specification of such systems. Furthermore, Co-design is introduced as an alternative design flow for embedded systems rather than the classic design flow because it has many issues that make it unsuitable for the design of such systems, particularly in the case of high complexity, which is one of the most prominent features of embedded systems.

Chapter 02 - Software Quality Assurance: This chapter introduces concepts and definitions related to the context of quality assurance. Aside from presenting the most acceptable quality models in the specialized literature. In addition to emphasizing the

need for standardization in offering a uniform view of the sophisticated term quality. Finally, it discusses the ISO/IEC 25010 Quality Model in full, which served as the foundation for our following contributions.

Chapter 03 - Quality Assurance for Embedded Systems: In this chapter, we give our first contribution, a Survey on Quality Attributes and Quality Models for Embedded Software (Tamrabet et. al, 2018). The importance of embedded systems justifies the choice of this subject. In addition, a full description of each work is offered, as well as its merits and limitations. Finally, a comparative study to draw the major shortcomings of the works presented is also given at the end of this chapter.

Chapter 04 - ESQuMo an Embedded Software Quality Model: In this chapter, we present our second contribution, ESQuMo an embedded software quality model that is based on the well-established standard ISO/IEC 25010 product quality model (Tamrabet et. al, 2022a). Besides, it is important to note that our model covers the generic characteristics of the ordinary software, side by side with the specific characteristics of the embedded software. In addition to the establishment of ESQuMo, the third contribution proposes a set of quality measures for the Medical Imaging Embedded Software as an application of the proposed quality model (Tamrabet et. al, 2022b).

Chapter 01
Embedded Systems

1. Introduction

We use general-purpose computers to achieve daily life goals from the tiniest computations to the most complex in a very explicit way like writing a report, googling on the web, playing games…etc. However, the majority of computer systems that we use are often less explicit to the point we do not even know they exist. These computing systems are usually found in airbags, signal transmitters, surveillance cameras, microwaves, desktop equipment, medical devices …etc. and they are known as embedded systems (Lee et al., 2016).

With the emerging technology of embedded systems, ordinary computing systems are on the verge of demise caused by ubiquitous computing, pervasive computing, and ambient intelligent computing. These three terms represent the futuristic trend of information technology in the next years. Ubiquitous computing focuses on the long-term goal of providing "*information anytime, anywhere*", whereas pervasive computing focuses somewhat more on practical aspects and the exploitation of already available technology. For ambient intelligence, there is some emphasis on communication technology in future homes and smart buildings. It should be noted that embedded systems are the cornerstone used in each of the aforementioned technologies (Marwedel, 2003). This embedded system technology has the ability to change the way people live in diverse aspects through extending sensors in life's everyday objects that will allow the information to be collected, shared, and processed in an unprecedented way.

Appeared in the 70s, embedded systems are now an integral part of diverse fields and are tending to generalize in all areas where computing power and algorithms are necessary. Every day we use about 80 embedded systems transparently, this increasing growth in the use of such systems has been fueled by the advent of microprocessors and microcontrollers that 80% of which are directed to the manufacture of the embedded systems. For example, in 1991, the industrial and medical devices market alone accounted for $31 billion compared to the general-purpose computing systems market of $46.5 billion. In addition, the market for microcontrollers amounted to $4.6 billion and has been rising at an 18% annual growth rate compared to a 10% annual growth rate for general-purpose systems (Gupta, 2012).

Embedded systems are generally the combination of software and hardware side by side to form electronic computer systems dedicated to specific tasks. These tasks are expressed through functional and non-functional requirements. On the one hand, functional requirements represent the functional properties of the system like taking pictures using the phone's camera. On the other hand, non-functional requirements represent how these tasks were achieved by the system like the phone camera should take a picture in less than 100 milliseconds while pressing the shooting key. Moreover, embedded systems are not ordinary systems as they submit to temporal, resources, and environmental constraints.

Despite the advantages that embedded systems offer in the various areas in which they operate, the designers of these types of systems are constantly confronted with the challenges of excessive use of resources, complex hardware architecture, high power consumption, and unpredictable time responses. Moreover, the lack of maturity of such systems may produce financial, health, or even injuries and life losses in the case of failure as the majority of embedded systems are considered critical operation systems.

In this chapter, we will introduce terms and definitions related to embedded systems in section 02. Section 03 is devoted to presenting the components of these systems. The fourth and fifth sections are devoted, respectively, to the features of embedded systems as well as the areas of use of these systems. Section 06 is dedicated to presenting the real-time embedded systems, as a relevant property of these systems that require more attention. The specification of embedded systems is presented in section 07, while their design is presented in section 08. Finally, some conclusions are found in the final section.

2. Definitions

In the context of embedded systems, many terms and definitions related to them have emerged, such as cyber-physical systems and the internet of things. In the following, we present the most accepted definitions.

Definition 01: "*Embedded systems can be defined as information processing systems that are normally not directly visible to the user, embedded into enclosing products such as cars, telecommunication, or fabrication equipment. Such systems come with a*

large number of common characteristics, including real-time constraints, and dependability as well as efficiency requirements" (Marwedel, 2003).

Definition 02: "*An embedded system is a microprocessor-based system that is built to control a function or range of functions, where a user can make choices concerning functionality but cannot change the functionality of the system by adding/replacing software*" (Heath, 2002).

Definition 03: "*An embedded system is any application where a dedicated computer is built right into the system*" (Ganssle, 2008).

Definition 04: "*Embedded software is software integrated with physical processes. The technical problem is managing time and concurrency in the computational system*" (Lee et al., 2016).

Definition 05: "*A cyber-physical system (CPS) is an integration of computation with physical processes. Embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa. As an intellectual challenge, CPS is about the* intersection*, not the union, of the physical and the cyber. It is not sufficient to separately understand the physical components and the computational components. We must instead understand their interaction*" (Lee et al., 2016).

Definition 06: "*Cyber-Physical Systems (CPS) refer to next-generation embedded ICT systems that are interconnected and collaborating including through the Internet of Things, and providing citizens and businesses with a wide range of innovative applications and services*" (Marwedel, 2021).

Definition 07: "*The term Internet of Things describes the pervasive presence of a variety of devices (such as sensors, actuators, and mobile phones) which, through unique addressing schemes, can interact and cooperate to reach common goals*" (Giusto et al., 2010).

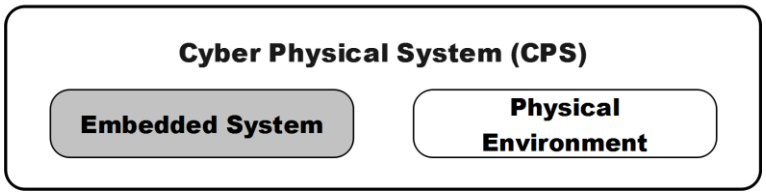

Figure 1.1: Relationship between embedded systems and CPS (Marwedel, 2021).

According to the above definitions, an embedded system is a computer system consisting of hardware and software, designed to perform specific tasks. Embedded systems are not always stand-alone systems, in many cases, they represent a part of a larger system. Actually, embedded systems are now the most demanded and used compared to the PCs and the ordinary computer systems, as they are dedicated, smaller, and even invisible in most cases. In addition, they are much cheaper than general-purpose systems, and because of that, they are widespread in countless fields such as industrial, automotive, home appliances, medical devices, telecommunication, commercial and military applications. Moreover, as a result of being multidisciplinary systems, they do submit to several constraints like real-time, dependability, resources, and environmental constraints.

Furthermore, embedded systems are occasionally referred to by other synonymous terms such as Cyber-physical systems and the Internet of Things. What distinguishes these definitions is that they are often related to the term physical environment. An embedded system is usually connected to the physical environment depending on the context of its use as it is reflected in Figure 1.1, unlike ordinary systems and PCs. The use of one of the terms Cyber-physical systems or Internet of things, for systems connected to the physical environment, remains preferential. What really matters is the detailed knowledge of the fundamentals of the embedded systems as they represent the basic technology in the foundation of future applications within Cyber-physical systems and the Internet of things (Marwedel, 2021).

3. Components of Embedded Systems

An embedded system is generally the combination of software and hardware. A typical embedded system consists of several components as Figure 1.2 shows. In the following section, we will present these components in detail (Lee et al., 2016; Marwedel, 2003; Marwedel, 2021).

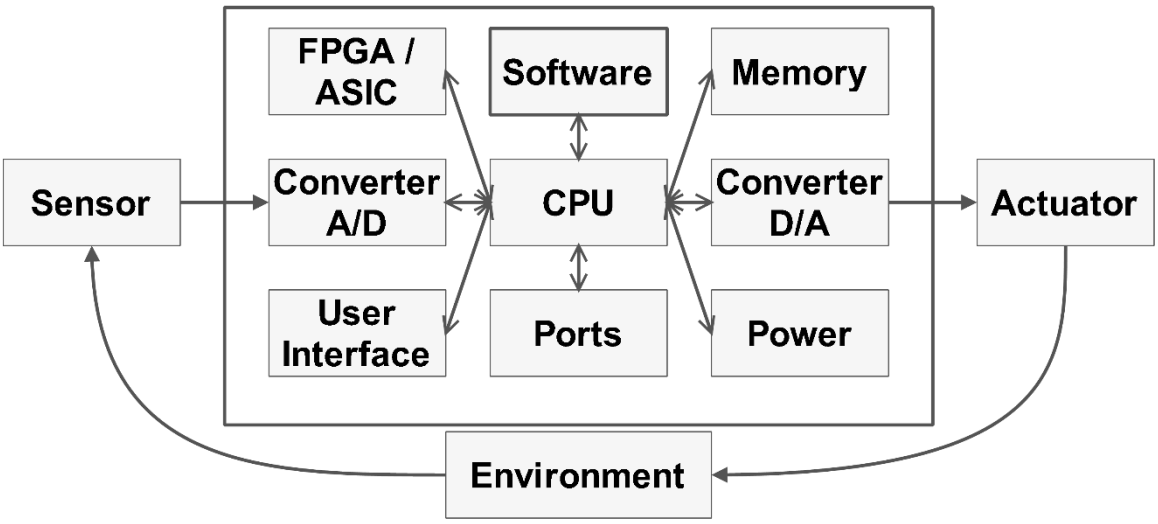

Figure 1.2: Scheme of a typical Embedded System (Serpanos et al., 2011).

Processor: The main part of an embedded system is the processor, which could also be a generic microprocessor or a microcontroller, programmed to perform the specific tasks for which the entire system has been designed (Emilio, 2015). Processors come in different architectures such as 8-bit, 16-bit, 32-bit, and 64-bit. Habitually, 8-bit processors are used for small-scale applications where no complex computations are involved. In the case of complex applications where performance matters the most, the embedded system requires supplementary resources to enhance the processor such as GPUs (Graphic Processing Units) in the context of Medical Imaging Devices.

Memory: Electronic memory is an important part of embedded systems and three essential types of memory can be described: RAM or random access memory, where data are temporarily stored during the execution of the system. ROM or read-only memory contains input and output routines that are needed for the system at boot time. The cache is temporary storage during the processing and transferring of data by the processor (Emilio, 2015).

Sensor: A sensor is an electronic instrument that can measure the physical quantity and generate a considerate output that is usually in the form of an electrical signal.

Actuator: An actuator is a device that alters the physical quantity as it can cause a mechanical component to move after getting some input from the sensor. In other words, it receives control input in the form of the electrical signal and generates a physical quantity through producing force, heat, motion, etc.

Power Supply: like any other system, the embedded systems need a power source. Embedded systems are usually characterized by being less energy-consuming systems, which makes them rely on small energy sources, and can also rely on autonomous energy sources such as small batteries or solar generators.

Real-Time Operating System (RTOS): Embedded systems use a RTOS in order to supervise the application software, schedule tasks, and manage hardware. Small-scale embedded systems may not need a RTOS.

FPGA/ASIC: In order to optimize the performance and reliability of embedded systems, dedicated integrated circuits for specific applications are used such as *Field Programmable Gate Array* (FPGA), and *Application Specific Integrated Circuits* (ASIC).

Ports: Embedded systems boards provide ports for the purpose of integration with other components such as sensors, actuators, memory supports, LCD display, etc.

Human Machine Interface (HMI): An embedded system does not have standard inputs/outputs such as a keyboard or a computer screen. Unlike a PC, the HMI of an embedded system can be as simple as a flashing Light Emitter Diode (LED) or as complex as a real-time night vision system.

Embedded Software: The embedded software is software that has a fixed functionality to run which represents the specific application of the designed system. The user does not have the possibility of modifying the software. Therefore, we use the term firmware instead. The firmware is a type of software that is usually found in ROM or Flash memory chips, and this is the main reason why the user could not alter it.

4. Features of Embedded Systems

An embedded system is a complex system that integrates software and hardware designed together to provide given functionality. It generally contains in the typical case sensors, actuators, a memory medium, and a microprocessor intended to complete the specific tasks of the system. The hardware and software systems are intimately linked and designed for each other to the point of not being easily discernible as in a conventional system. Other than that, embedded systems are characterized by the following (Lee et al., 2016; Marwedel, 2003; Marwedel, 2021):

Frequently: Embedded systems are frequently systems as they affect the environment through actuators. As well as affected by the same environment while collecting information through sensors.

Dependable: Embedded systems have to be dependable. The term dependability is always linked to the property of the embedded system being a safety-critical system. Autopilot systems, bank transitions systems, and nuclear monitoring systems are good examples of safety-critical systems. This term as it appears is always attached to the catastrophic consequences that could happen if something went wrong in the system. In other words, these safety-critical systems are directly connected to the environment and have an immediate impact on it. Dependability consists of the following aspects of a system:

- Reliability: the probability that the system will not fail.
- Maintainability: the probability that the system in case of failure can be repaired within a certain period.
- Availability: the probability that the system is available. Both the reliability and the maintainability influence the availability of the system.
- Safety: is the property of the system that reflects its ability in the event of a malfunction, not to cause any damage.
- Security: is the property of the system that reflects its confidentiality and data.

Efficient: Embedded systems should be efficient. The efficiency of such systems is usually related to the number of resources consumed by the system. The following aspects are important for the efficiency of embedded systems:

- **Energy:** In most cases, embedded systems are autonomous energy systems that rely on batteries as a power source. However, battery technologies are evolving at very low rates, as an example: Lithium batteries not only lose their charge during use, but they also lose their ability to store charge after a certain number of charges, as well as overtime even without frequent charges, not to mention the deterioration in performance during low or high temperatures. Therefore, every charge provided by these batteries must be exploited, otherwise we cannot consider the system efficient.
- **Code size:** All the code to be run on the embedded system has to be stored in a part of the system. For this, the code size should be as short as possible, and this

is for two main reasons. First, long-sized code means more instructions to be executed that makes them more energy-intensive, which is inconsistent with the previous point. Second, the resources related to embedded systems, especially memory, are usually expensive, so short-sized code is a major requirement, especially as embedded systems are intended for specific functions and not for general use.

- **Runtime efficiency:** In order to meet the time constraints while using the embedded systems in an efficient manner, the least amount of resources should be used. For example, in order to reduce energy consumption, a short-sized code must be used, as we mentioned earlier. In addition to the clock frequencies and the supply voltage as small as possible. Furthermore, all the additional and complementary components (such as many caches or memory management units) can be omitted for better efficiency.
- **Weight/Size:** One of the features that are unique to embedded systems from other systems is that they are designed to operate in limited spaces. Therefore, weight, size, and shape must be taken into consideration while building embedded systems.
- **Cost:** The price of embedded systems is a sensitive criterion in the electronics market. Customers and users usually tend to products that serve their purpose at the lowest possible cost, and herein lies the efficiency according to the user.

Dedicated: An embedded system is a system designed specifically to perform a dedicated function. For example, a building monitoring system is an embedded system that runs the software responsible for monitoring in a continuous way. It is not possible to run any additional program on the same system and this is for two main reasons: First, running any additional program exposes the monitoring system to a loss of dependability and this contradicts the dependability aspect. Second, in order for the additional program to run, there must be available memory for the additional program, and here the system is inefficient as an amount of resources is unused, which contradicts the efficiency aspect.

Dedicated User Interface: Embedded systems do not use standard inputs and outputs such as keyboards, mice, and computer monitors. Instead, embedded systems use dedicated Human Machine Interfaces such as push buttons, steering wheels, pedals, etc.

Therefore, the user hardly knows that the information is being processed by an embedded system.

Hybrid: Most of the embedded systems are hybrid systems. Hybrid in terms of containing both digital and analog components. Processors and memory are digital components, unlike sensors and actuators that are analog components. Converters are often used to guarantee coordination between them most of the time.

Reactive: Embedded systems are reactive systems. According to (Bergé et al., 2012), a reactive system is defined as "*a system that is in continual interaction with its environment and executes at a pace determined by that environment"*. The logical representation closest to reality for any reactive system is through automata where the system is in a certain state, waiting for input from the environment. For each input, some computations are made, and a new state is established.

Real-time: A real-time system is defined as a computer system whose operation is conditioned by the dynamic evolution of the state of the environment connected to it and of which it must follow or control this imposed environment while respecting the time constraints (Khadidja, 2013). Real-time embedded systems are systems in which operations are then performed in response to an external event. The validity and relevance of a result depend on when it is delivered. A missed deadline induces an operating error, which can cause either a breakdown of the system or degradation of its performance. Therefore, in the context of real-time embedded systems, it is not only the outcome of the output that matters but also the time of its occurrence.

5. Application fields

Every day we use embedded systems on an average of 80 different systems. Some of them we use directly and intentionally, while others are used transparently as these systems are disappearing systems within larger systems. In the following, we present the most common areas in which embedded systems are found. Despite the fact that these systems differ significantly in terms of the physical part and context of use, they have a lot in common such as the characteristics of embedded software (Lee et al., 2016; Marwedel, 2003; Marwedel, 2021).

Automotive electronics: The automotive industry is one of the areas most affected by embedded systems. Whereas, modern cars that support a greater number of embedded

systems such as the air conditioning system (AC), anti-braking system (ABS), and global positioning system (GPS) are the most demanded in the automotive market.

Aircraft electronics: Airplanes also in the field of aviation increase in value as long as they rely more on embedded systems. Higher quality of the equipment in the avionic embedded systems means a greater dependability ratio, which in turn is considered an important criterion in aircraft safety. The equipment of the embedded systems used in the aircraft includes flight control systems, anti-collision systems, and pilot information systems.

Trains electronics: For trains and railway control systems, the situation is similar to the one discussed for cars and airplanes. As a safety-critical system, embedded systems must be used to ensure the dependability of these systems.

Telecommunication: Telecommunication is one of the most growing fields in recent years, and this is due to the technology provided by the embedded systems, through digital signal processing, radio frequency design, switchers, hubs, etc.

Medical systems: Medical devices are just as important as the above. With the technology provided by embedded systems, medical devices now offer greater accuracy in the medical field, whether in diagnostic or surgical operations. Many medical devices can be found such as Defibrillators, Oximeters, Fetal monitors, CT scans, MRI, Ultrasound…etc. (Dere, 2021).

Military applications: Embedded systems have been used in the military application for many years. They are found in mine detectors, radars, missile control systems, etc.

Authentication systems: Embedded systems can be used for security purposes. This is currently possible via fingerprint and facial recognition systems.

Consumer electronics: This is probably the most commonly used type of embedded system in daily life. Embedded systems of this type include everything related to audio and video such as Personal Digital Assistants (PDAs), TVs, receivers, modems, drones, and game consoles like Nintendo and play stations.

Fabrication equipment: The industrial field is one of the traditional fields in which there are many embedded systems used. Fabrication equipment, for example, production lines depend mainly on embedded systems, where reliability and accuracy

are highly required, with the exception of intensive-energy consumption, which is a special case for embedded systems that are known to be less consuming.

Smart buildings: Embedded systems are also found as a core technology in smart homes. In addition to its primary goal of ensuring reliability and improving security and safety, embedded systems also seek to improve the experience of living in smart homes, especially with the emerging technology of the Internet of things with which a new era of homes arises.

Robotics: The field of robotics is also a traditional field that encounters and finds embedded systems technology present and very used. What distinguishes robots is that the mechanical part dominates the robots, as some robots are developed to simulate the movement of animals and humans. The Atlas robot developed by Boston dynamics is a good example (Nelson et al., 2019).

6. Real Time Embedded Systems

Embedded systems generally operate in real-time. Therefore, they are forced to meet real-time constraints. The non-respect for these constraints can result in a serious loss of dependability, performance, and quality provided by the system. Real-time systems are defined as "*systems in which the correctness of the system depends not only on the logical results of computation but also on the time at which the results are produced"* (Stankovic, 1988). From here, it is clear that the validity of the outputs provided by the real-time systems is not only related to their correctness, but also to the time of their occurrence and the degree of respect for time constraints. In the event of a system malfunction due to one of the time constraints, the consequences come according to the type of time constraints breached. According to Kopetz (2011), "*a real-time computer system must react to stimuli from its environment within time intervals dictated by its environment. The instant when a result must be produced is called a deadline. If a result has utility even after the deadline has passed, the deadline is classified as soft, otherwise it is firm. If severe consequences could result if a firm deadline is missed, the deadline is called hard"* (Kopetz, 2011). Hence, real-time systems can be divided into three categories, based on time constraints: Hard, Soft, and Firm real-time systems.

- Hard real-time: systems where failure to meet time constraints leads to a system breakdown are called hard real-time systems. In this kind of system, timely system

responses are vital and failure to respond is catastrophic. Examples: Flight control systems, nuclear power plant systems, etc.

- Soft real-time: systems where failure to meet time constraints leads to performance degradation of the system, but not to the point of breakdown. The late responses or non-responses of the system have no catastrophic consequences. Example: Multimedia systems.
- Firm real-time: systems where failure to meet time constraints can be tolerated. In this kind of system, timely system responses are essential, and the result is useless once the deadline has passed. Examples: Weather forecasting systems, telephony systems.

There is a fragile barrier between embedded systems and real-time systems. Consequently, it is difficult to differentiate between the two systems separately. Real-time embedded systems are usually referred to as the intersection of the two domains, as shown in Figure 1.3 (Li et al., 2003).

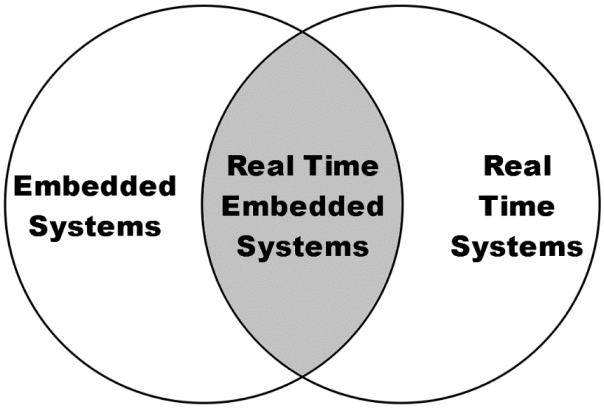

Figure 1.3: Real Time Embedded Systems.

7. Specification of embedded systems

In this section, we will focus on specifications and the different representations used in order to describe the embedded systems. The specification is an interesting phase in the software life cycle, it is an abstract description of the future system, and it describes what the latter must do without giving details of how it does it. First, we will introduce the structural representations. Then, we will focus on behavioral representations, before presenting some other types of representations. This will be in conjunction with the most popular representation examples. It should be noted that the same representations used to describe the embedded systems mentioned in this section are referred to as

computational models in other references (Lee, 1999; Marwedel, 2003; Lee et al., 2016; Marwedel, 2021). In this thesis, both terms are used alternately.

7.1.Structural representations

Structural representations are considered the simplest types of description for any system because they allow describing how a system is constituted, what are its components and what are their relationships. In the following, we will present two widely held structural representations: topological representations, and relational representations (Gauthier, 2001).

7.1.1. Topological representations:

Each element in these representations corresponds to a component in the real system. It should be noted that many hardware description languages such as VHDL and SDL propose this type of representation because of its ability to describe the architecture of a circuit (Airiau et al., 1998; Olsen, 2012).

Many topological representations exist for the representation of an embedded system. However, they generally share the same basic concepts, which are represented in modules, ports, and channels (Nicolescu et al., 2002).

- Modules represent the basic elements for each system, as they can be subsystems in their turn. Communication between modules is possible through ports and channels.
- Ports represent the interfaces of the modules, i.e. through these ports; the modules therefore interact with the outside world.
- Channels represent the hardware link that connects the ports, and therefore the modules are connected as well. Thus, two modules connected by a channel can interact, while two unconnected modules cannot.

Besides, several topological representations enhance these basic concepts by making them hierarchical. This hierarchy is of prime importance as human beings are incapable of handling complex components, especially when they have relations with other components. Hierarchy is then introduced to solve this problem through the decomposition of components into simpler components and fewer relations (Marwedel, 2003). For example: in languages such as VHDL, modules can contain sub-modules. The following Figure 1.4 gives an example of this type of representation: two modules

M1 and M2 respectively contain the sub-modules m1, m2, and m3. The M1 and M2 modules communicate with each other through the C1 and C2 channels using ports P1 to P4. While the m1, m2, and m3 sub-modules communicate through the channels c1 to c5 a via the ports p1 to p6 (Gauthier, 2001).

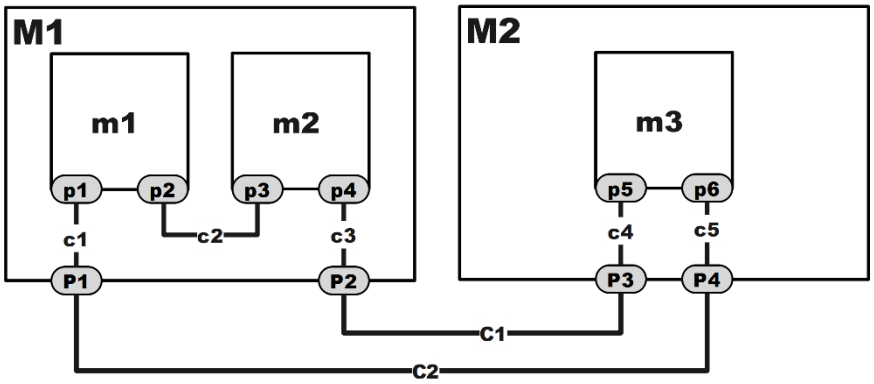

Figure 1.4: A topological representation of a system.

These are very general definitions, and they can be used with very different semantics and levels of abstraction depending on the description language used. For example, in VHDL (Airiau et al., 1998) the modules represent circuits, the ports are their input or output pads, and the channels are the wires connecting them. Thus, the ports and the channels respectively represent only communication points and the media passing on the communication. Nevertheless, in SDL (Olsen, 2012) channels encapsulate both media and behavior; this is why the data transmitted are usually complex messages.

Moreover, some topological representations exploit the instantiation mechanism with the purpose of explicitly materializing the components. The instantiation mechanism consists of using any element multiple times without having to duplicate the basic information, and this is possible through the instantiation with particular parameters for each instance. These instances are often used in hardware description languages such as VHDL, in order to define libraries of modules, with which the description of a circuit is made. In this case, it is only required to instantiate the module to obtain the description of the circuit (Airiau et al., 1998).

7.1.2. Relational representations

In relational representations, the description of a system depends mainly on the properties of the system and its relations rather than describing it through its

components as in the case of topological representations. In other words, relational representations describe the system by its semantics (Gauthier, 2001).

The object-oriented approach is a very good example of this representation. In the object-oriented approach, a system is represented in the form of objects. Each object is defined by its attributes and its methods, which both represent its state. The state of the system, therefore, corresponds to the state of its objects. Objects are identified by a unique identity that helps differentiate objects from each other in the same system (Booch, 1990; Derr, 1995).

To construct a new object, all you have to do is instantiate a class. The notion of a class is simply a grouping where the object belongs. During the instantiation of the object-oriented approach, the following concepts must be respected:

- Encapsulation: this concept consists in making the attributes and methods of an object hidden as an internal specification of the object. Only the external behavior is visible to other objects.
- Inheritance: is a concept that automatically passes attributes and methods from an upper class to another sub-class. This passage supplements the sub-class in addition to the other attributes and methods that can be provided.
- Message passing: this concept consists of the interaction between objects being made through the sending of messages.
- Polymorphism: it is a concept that makes it possible to generalize a given operation according to the type of parameterization of the operation. For example, a method such as addition can thus be for integers, floats, or strings.

7.2.Behavioral representations

Structural representations neglect the functional aspect of systems. For this, behavioral representations are used to overcome this deficiency by describing the functional aspect of systems, in addition to their evolutions and reactions. There are many behavioral representations used to describe embedded systems, in the following we will present the most commonly used representations in this context (Gauthier, 2001).

7.2.1. Equations

The physical approach is adopted to represent the behavior of a system using equations. A system is therefore represented by a series of equations whose parameters are related to the external environment such as time, voltage, etc. Equations are often presented in the form of a data flow graph as shown in Figure 1.5. In a data flow graph, nodes represent operators, and arcs represent data dependencies.

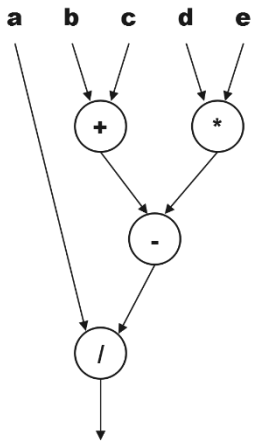

Figure 1.5: Data flow of an equation representation.

Differential equations are another type of equation that strongly represent physical models, where the use of such physical models requires solving its equations. This type of representation is commonly used in the description of analog circuits on the one hand. On the other hand, it is unsuitable for logical descriptions. In fact, differential equations allow precise modeling of the physical behavior of a system. However, they require huge calculation times, which limits their use.

Furthermore, sampled differential equations give us another sort of equation called finite difference equations. These equations are simpler to calculate than the differential equations and are usually used in signal processing. The use of finite difference equations requires a general clock for the sampling process, which makes it difficult to model temporal irregularities such as events. Hence, interactive event-based systems cannot be modeled by these equations.

7.2.2. Finite State Machines

Another famous behavioral representation is through finite state machines. Finite state machines consist of an oriented graph, whose nodes represent the states of the system, and the arcs represent the transitions (Gauthier, 2001).

The practicality of this representation lies in its ability to perform formal analyses, formal verifications, and simulations. Moreover, it is easily feasible in hardware as well as in software. However, the number of states in this representation can become very high even in the case of low complexity behaviors (Chiodo et al., 1993; Lee, 1999).

Finite state machines have often been extended with the addition of a few concepts. These extensions reinforce the expressiveness of these representations. For example, Communicating Finite State Machines (CFSMs) in an extension of FSMs that the finite state machines communicate with each other, where the method of communication must be defined (Marwedel, 2003).

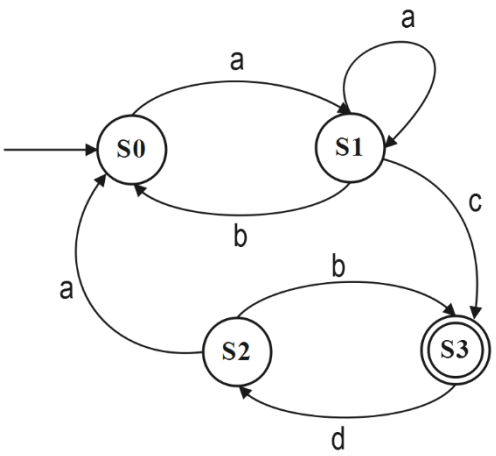

Figure 1.6: Example of a deterministic FSM.

Finite state machines have an initial state; an incoming arrow often indicates this state. A node with double contour on the other hand indicates the final state. The passage from one state to another is done through transitions. Each transition is represented by a curved arrow. Besides, the transition can be a self-transition if it starts and ends in the same state (Lee et al., 2016). Usually, a FSM is described by five tuples ($Q, \sum, \delta, q_0, F$) where:

- Q: a finite set of states.
- $\sum$: a finite nonempty inputs (alphabets).

- δ: transition functions from $Q \times \sum \rightarrow Q$.
- q_0: the initial state.
- F: set of final states.

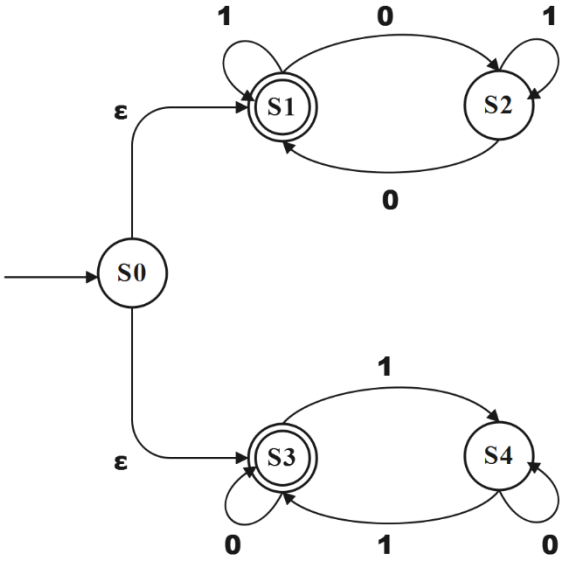

Figure 1.7: Example of a non-deterministic FSM.

There are two types of Finite State Machines FSMs: deterministic and non-deterministic. In deterministic FSM, there must be one and only one transition function for every input in $\sum$ from each state. This means at any given state, the next state is also known as it has only one unique next state as depicted in Figure 1.6. Contrariwise to deterministic FSM, the non-deterministic FSM can have multiple transition functions in the same state for the same input. This means at a given state, the next state is unknown due to the multiple possible next states. Furthermore, the non-deterministic FSM is distinguished by the presence of null transitions, which are indicated by ε the empty input as shown in Figure 1.7 (Aaronson, 2016).

7.2.3. Petri Nets

Another well-known representation is through Petri nets in which the systems are shown as places and transitions. The state of the system is therefore the state of the corresponding Petri net, the latter being relative to a marking of the places by tokens. The dynamic aspect of the system is obtained by all the firing transitions using tokens in the corresponding Petri net (Gauthier, 2001).

In Petri nets, the passage from one state to another is carried out by treating all the transitions. When a transition fires, a token is removed from each input place and added to each output place. For a firing transition to be completed, there must be at least one token in each input place, in this case, the transition is called enabled as it includes a sufficient number of tokens in each input place. Figure 1.8 gives an example of a Petri net in several consecutive states. In the figure, there is indeterminism at the level of state three; this happens when multiple transitions are enabled at the same time. Thus, they will fire in any order.

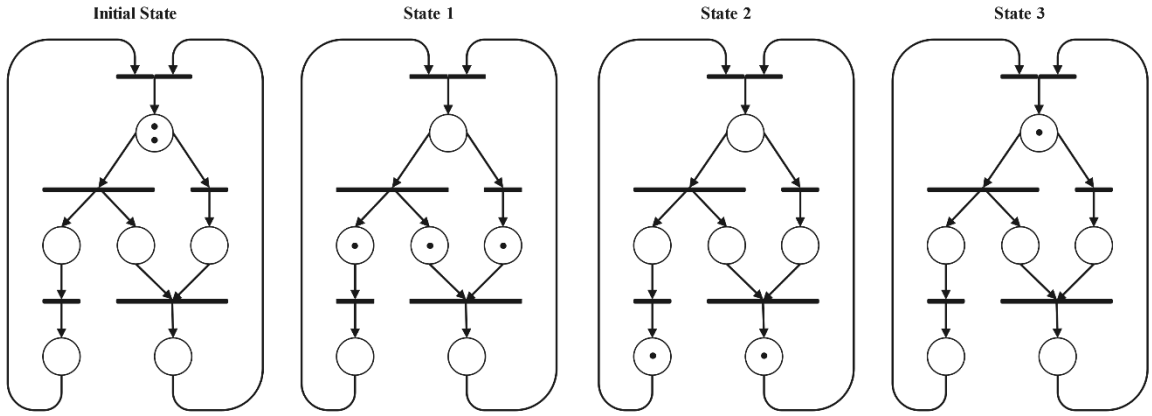

Figure 1.8: Example of petri nets representation.

Many extensions can be obtained from Petri Nets. For example, a very recognized extension is to add weights to transitions as depicted in Figure 1.9. A transition can only be fireable if each input place has at least as many tokens as the input weight. When the transition fires, these tokens are removed, and in each output place, tokens are added to as many as the output weight. Figure 1.9 illustrates the same Petri net as Figure 1.8 with the exception of the weights that eliminate the indeterminism situation as well as making state three the initial state.

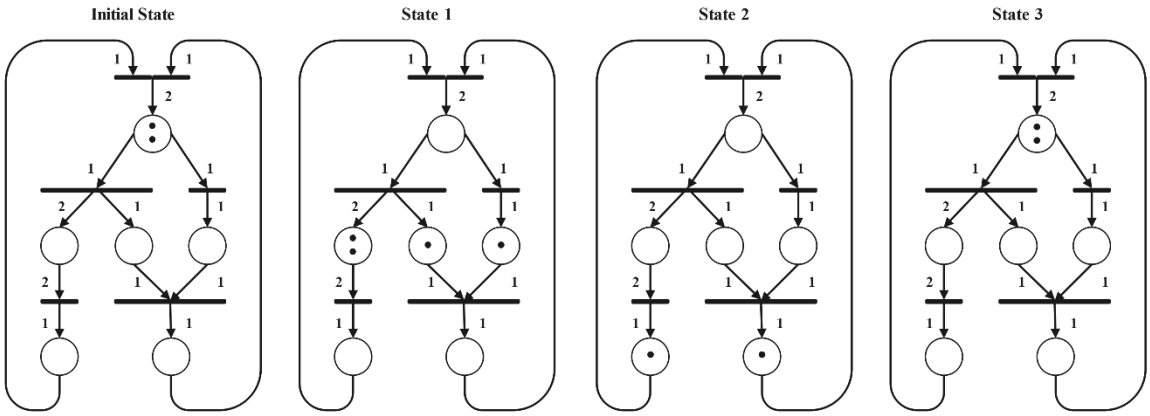

Figure 1.9: Extension of petri nets representation with weighted transitions.

Another extension consists in associating time with each transition, which makes the Petri net a temporal simulation model. Besides, it is possible to associate attributes with tokens in order to express the condition of the firing transitions.

Unlike finite state machines, Petri nets make it possible to represent systems with a very large number of states without being large. Using Petri nets, it is possible to perform formal verifications, simulations, circuit syntheses, and analyzes in various fields (Lee, 1999; Renaudin, 2000).

7.3.Communication-based models

Communication-based models are mixed representations in which the system is decomposed into several behavioral units communicating with each other. Such representations introduce the topological structure of the system through its units, as well as its relational structure in terms of the overall behavior of the system.

For the behavioral aspect, these models use the notion of an event to represent communication. Moreover, actions are executed only when the corresponding event happens. A set of actions can be associated with a set of events under the name of a process. These models are often used to describe and simulate digital circuits (Gauthier, 2001).

7.3.1. Discrete Event Models

The notion of time is often added to such models in the form of stamped events, indicating the time at which the events occur. In these models, events are generated by actions. In the case of simulation, a first action is executed that generates other events,

which in turn trigger actions. The simulation stops when there are no more events generated. These models can be realized in hardware as well as in software. However, they are usually slow in the case of software (Lee, 1999).

7.3.2. Synchronous Event Models

The events in this type of model arise from a global clock. The processes are executed at each clock tick and produce the result immediately. Similar to discrete event models, these models are applicable to hardware and software with the exception of high-speed execution in the case of software (Lee, 1999).

7.3.3. Asynchronous Message Passing

In asynchronous message passing, the communication between the processes is done by sending messages through channels, which allow the messages to be buffered. In this mode of communication, the sender does not need the receiver to be in a waiting state for the message. However, the possibility of an overflowed channel is very likely (Marwedel, 2003).

7.3.4. Synchronous Message Passing

Contrariwise to asynchronous message passing, there is no risk of an overflow as the communication is done with the presence of both parts of the communication. The first process reaching the *Rendez-Vous* point must wait for the second to join; the performance, in this case, might be deteriorated (Marwedel, 2003).

8. Design of Embedded Systems

With the advent of early embedded systems, there was no need to use a special methodology during their development. However, since the increasing rate in the complexity of these systems, it became necessary to adopt a specific design flow for these types of systems, as the classic design flow is no longer sufficient.

8.1.Classical design flow of embedded systems

The classical design flow usually begins with a general specification of the system, followed by an early partition of the system into hardware and software. These latter partitions are habitually developed by independent different teams. Each team

specializes in either hardware or software. An integration process follows this stage by another independent integration team, which in most cases encounters incompatibility problems. This integration directly gives a prototype to test. In the event of an error, the design flow must be completely restarted from the beginning (Gauthier, 2001).

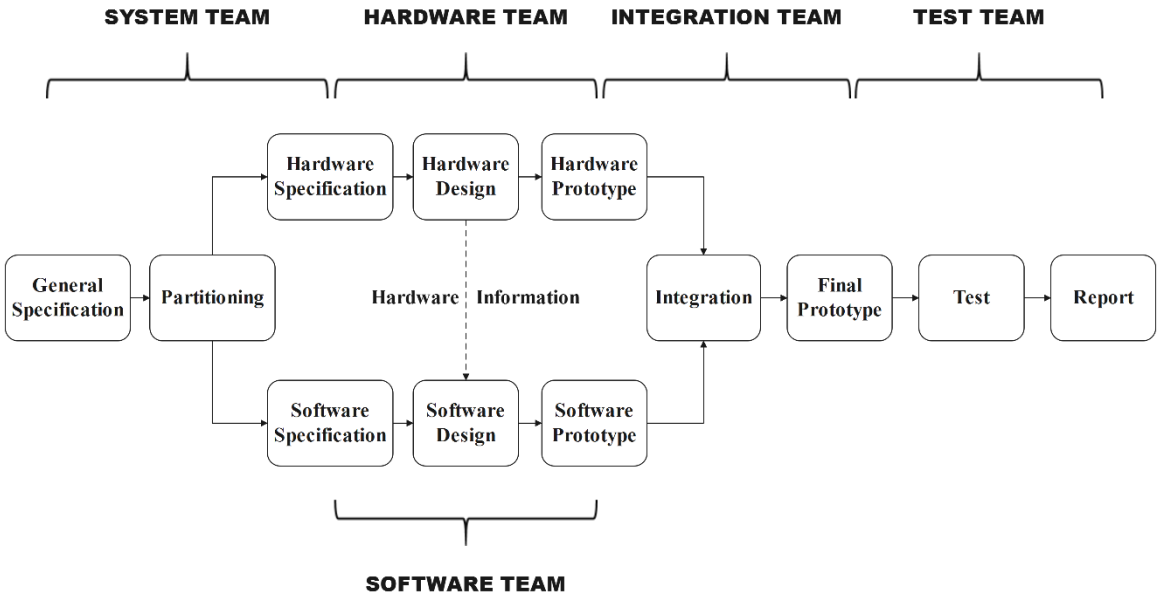

Figure 1.10: The classical design flow of embedded systems.

Although this design flow gives the illusion of parallelism in the partition of the system as shown in Figure 1.10. However, in reality, it is difficult to develop software without an accurate description of the hardware. Therefore, in order to complete software development, the hardware description must be finished first.

Furthermore, in this classic design flow, there is no general description of the embedded system that guarantees for designers to be aware of the totality of the system from the beginning until the end. Besides, the separation between software and hardware is done too early in the classic design flow. Hence, it is impossible to know the best configuration and vice versa in the case of the integration. While the integration process of hardware parties and software parties is done very late, it is impossible then to eliminate any incompatibilities.

The classical design flow has many weaknesses that make it unsuitable for supporting the design of embedded systems especially when the complexity of such systems goes beyond control. For that reason, Co-design is adopted as an alternative design flow for the embedded systems.

8.2.Co-design of embedded systems

The co-design is founded primarily to solve the problems of the classic design flow for embedded systems by offering a global description of the system, which makes it possible to develop at the same time the different software and hardware parts of a system (Gajski et al., 1994; Kumar et al., 1995; Kalavade, 1997).

When adopting co-design as a design flow for embedded systems, two approaches are possible to provide the global system description:
The first approach is to describe all parts of the system in a single language, which must then have semantics for each of these parts. The advantage of this approach is that it is simpler and easier for tools and users to manage a single language than multiple languages. However, it is difficult, and sometimes impossible, to define a language that is suitable for all parts of the system.

Contrary to the first approach, the second approach consists in using several languages, each one being adapted to a part of the system. Another coordination language is used to make the link between all the descriptions obtained by the different languages. The advantage is that this time it is possible to have the optimal language for each part of the system. Nevertheless, managing all these languages is quite difficult.

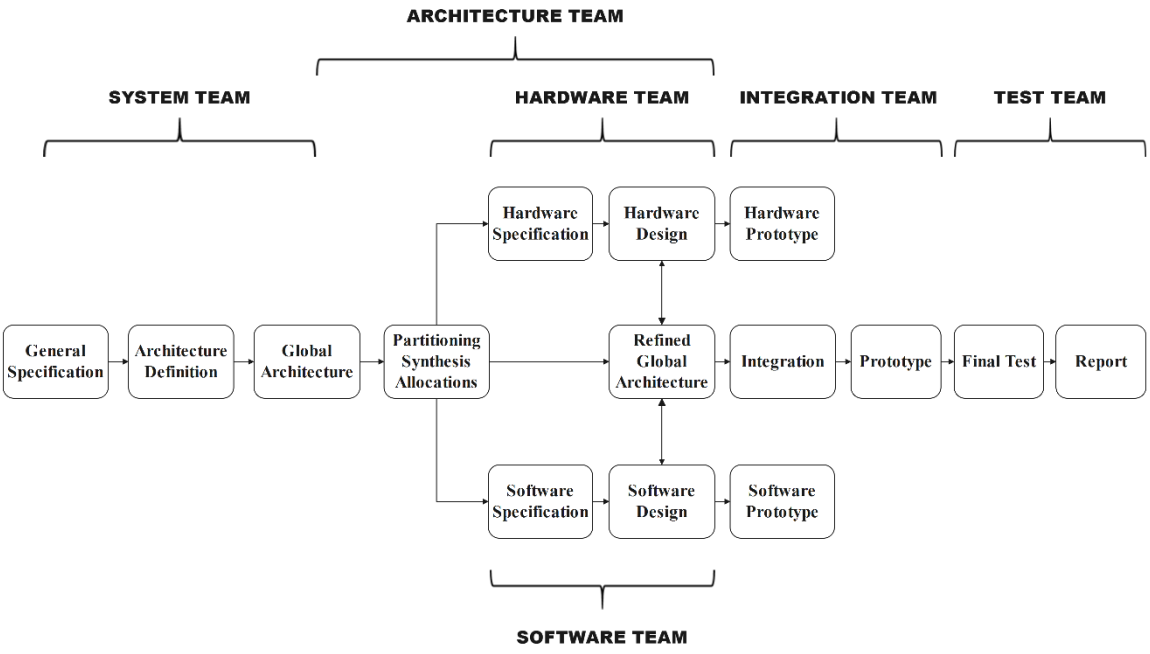

Figure 1.11: Co-design of embedded systems.

As depicted in Figure 1.11 an example of a design flow based on co-design is presented. In this design flow, the separation between software and hardware is done later for a better choice of software and hardware configuration. Moreover, unlike the classic design flow, the integration is done early in order to remove the incompatibilities as much as possible. In addition, a new team makes its appearance: the architecture team, which deals with the global system and the coordination between the teams. It should be noted that the hardest part of co-design is the partitioning between software and hardware. Although several heuristics have been developed. However, manual partitioning still prevails.

9. Conclusion

Embedded systems are electronic and computer systems, completely controlled by integrated software. Designed to solve specific tasks, but they are not general-purpose computers. Today, these systems are adopted by several application fields such as automotive, robotics, communication, etc. Usually, embedded systems submit to sensitive constraints such as low cost, low energy consumption, small size, high performance, and reliability, in addition to other temporal and environmental constraints. In this chapter, embedded systems and their features are presented, as well as the various representations used in the specification of such systems. Moreover, the Co-design was introduced as an alternative design flow for embedded systems instead of the classic design flow since it has many shortcomings, which make it unsuitable for the design of such systems, especially in the case of high complexity that is one of the most prominent features of embedded systems. In the next chapter, we will focus on quality assurance in the context of embedded systems.

Chapter 02
Software Quality Assurance

1. Introduction

Software is now a wide-ranging commodity. In fact, this product is integrated into our lives through its application in almost all daily use systems from the simplest to the most complex. In addition, software products represent essential components in complex and critical systems. The development of low quality software is not only discouraged but sometimes becomes intolerant.

As a result, software engineering took place. Software engineering emerged in the late of 1960s following the software crisis. This crisis was due to both the increase in the complexity of software on the one hand and to the significant decline in the quality of the latter. Therefore, the development of quality software is considered among the major objectives of this field.

In addition, the software product is characterized by its abstract nature, which complicates the development process and makes it difficult to obtain quality software. In fact, the nature of software products has completely changed over the past few years and this is mainly due to the continuous demands of users, which requires finding effective ways to create more stable software products that meet user expectations. Software engineering has attempted to do this through various systematic procedures in order to arrive at secure, reliable and efficient software.

In this chapter, we will present the different definitions and terminology related to the concept of software quality. Next, we will present software quality assurance as an important activity in software engineering, then the different quality models used during this activity. Finally, we end with highlighting how important standardization is in this context. With more emphasis on the standard ISO/IEC 25010 quality model.

2. Software Quality Assurance

Quality is a complex, vague and ambiguous concept. Several researchers have proposed various definitions due to the lack of consensus concerning this concept. In fact, quality is still a source of debate as there is no universal agreement on what it is. For demonstrative purposes, we cite some of the most accepted definitions:

- Quality is defined as conformance to requirements (Crosby, 1979).

- Quality is the degree to which a system, component, or process meets specified requirements, customer or user needs and expectations (IEEE, 1990).
- Quality is often used similarly with "customer value" or "defect levels" (Highsmith, 2002).
- Quality is the degree to which a set of inherent characteristics fulfills requirements (ISO, 2005b).

As we can notice from the above definitions, conformance to requirements and user needs represents the basis on which the definition of quality has been founded. However, conformity to user needs is not the only characteristic required to satisfy quality. It is highly possible to design or implement software that meets the needs of the user but with poor quality.

In software development, the user satisfaction is an important and referential criterion for product quality. However, the user's opinion of the product is usually superficial. This is quite natural, as the user sees the product as a black box, which he supplies with the inputs and yields to him the outputs that express only the functional requirements. From this perspective, it seems that the user is completely ignorant of the quality requirements as same as being ignorant of bits and bytes. This is mainly due to the multidimensional nature of quality as defined by Garvin (Garvin, 1984) through the five perspectives of quality, the very same perspectives adopted by Kitchenham and Pfleeger (Kitchenham et. al, 1996):

- The transcendental perspective: In this perspective, quality is represented as an ideal attribute that one can recognize, but can never achieve.
- The user perspective: this perspective is related to the user as its name indicates. Precisely, it is related to the user's satisfaction with a product in a specific context of use.
- The manufacturing perspective: this perspective means that a quality product is the product conforming to its specifications and requirements.
- The product perspective: according to this perspective, quality is represented by the inherent characteristics of the product.
- The value-based perspective: This means that different perspectives of quality may have different values to different stakeholders.

Nowadays, we are increasingly relying on software that provides all kinds of services. Thus, developing high quality software is a central goal for any software project. Based on this assumption, quality management activity has taken an important place in software development processes. This activity is mandatory in order to achieve a high quality software product. According to Sommerville (2007), this activity includes three quality layers: software quality assurance, software quality planning and software quality control. Firstly, software quality assurance is a systematic process that includes all the standards, regulations, techniques and tools in order to obtain quality software products. Secondly, software quality planning consists to choose among the previous standards and techniques the most appropriate ones for specific projects. Finally, software quality control consists to define the process which allows following all the standards and techniques used by developers. In other words, software quality control is the process in which we specify requirements, evaluate and compare the actual quality with the defined once to make the right correction activities (Wagner, 2013).

As we have already mentioned, the aim of software quality assurance is to build software products with a well-accepted level of quality. This process includes all the techniques either to build quality or to analyze it. This is feasible through *the constructive quality assurance* and/or *the analytical quality assurance* (ISO, 2010a). The constructive quality assurance includes all the means and techniques used to produce products conforming to their requirements (ISO, 2010a). In contrast, the analytical quality assurance includes all the means and techniques used to analyze the quality level of a product (ISO, 2010a). Considering this, quality models are considered among the most useful techniques in the context of quality assurance. A quality model is the representation of quality through a set of characteristics and sub-characteristics. While a quality characteristic denotes one of the many perspectives of quality. Because of the lack of a certain unique definition, we surround the term quality with a set of characteristics (ISO, 2010a). Eventually, we must point out that quality models have an objective to describe the decomposition of the concept of quality into a set of characteristics and sub-characteristics. This description is often used to define, assess and/or predict quality (Wagner, 2013).

3. Software Quality Modeling

As previously indicated, quality is a complex concept expressed through several characteristics that are usually in contradiction with each other. For example, a plane parked at the airport is not reliable since it does not provide the intended services, but it is safe enough for it cannot crash into the ground (Wagner, 2013). Consequently, quality modeling represents an instrument that facilitates the understanding of this concept. A quality model combines the various attributes contributing to quality in a single framework. Obviously, the relationship between these attributes is primarily dependent on the approach used for the construction of the model. Generally, a quality model is a well-accepted means to support software quality control. Basically, it allows to (Suryn, 2013):

- Help understand how the different facets of quality contribute to overall quality.
- Make it clear that quality is more than just faults and failures.
- Help in identifying and defining quality requirements.
- Help to navigate through the different definition levels of quality.
- Help to define an evaluation profile (what precisely should be evaluated).

In software quality engineering, to achieve an appropriate level of quality, a formal management is required all the way through software or system's lifecycle. In order to do so, the selected quality model must have the ability to support the definition of quality requirements as well as their subsequent evaluation. This is expressed as a top-down and bottom-up approaches of quality mentioned in the IEEE Standard 1061-1998 (IEEE, 1998):

- The Top-Down approach: In this approach, we start with the establishment of quality characteristics, which define the concept of the overall quality. These quality characteristics are then expressed in term of sub-characteristics. Finally, the identification of the measurements related to characteristics and sub-characteristics.
- The Bottom-Up approach: In contrast to the previous approach, we start with measurements to define the different sub-characteristics, which are in their turn combined to form the main characteristics.

Quality models are then much-supported tools to achieve quality, as they constitute the foundation of software quality assurance. In quality assurance, we distinguish the

establishment of quality models, frameworks, and organizational procedures. However, quality models proved that they are well-accepted means to support quality of software systems. Quality models have as objective to define, assess or predict quality.

The first software quality models were published in the late 1970s by Boehm and McCall (Boehm et. al, 1976; McCall, 1977) who described quality characteristics and their decomposition. Both quality models share the same hierarchical decomposition of the concept of quality into quality factors such as maintainability or reliability. Over time, several variants of these models have emerged. One of the most popular is the FURPS model, which decomposes quality into functionality, usability, reliability, performance, and supportability (Grady et. al, 1987). Aside from this hierarchical decomposition, the main idea of these models is to decompose quality to a level where it can be measured and thus evaluated.

After so many hierarchical quality models, a new way of decomposition quality characteristics was proposed by researchers in the 1990s. The idea is similar to UML; you can get models by the instantiation of a meta-model. This way not only allows for building richer quality models, but also gives a clear interpretation of the model due to rules founded by the meta-model. The Constructive Quality Model (COQUAMO) proposed by Kitchenham is a good example (Kitchenham, 1987).

Based on statistical terms, statistical quality models are used to capture properties of the product or process and turning them into quality factors by estimation or prediction. The famous example is the reliability growth models (RGM) (Lyu, 1996). They extend the concept of hardware reliability models to software. The purpose is to observe a software's failure behavior and predict how it will change over time.

3.1.The DAP Classification

Even though definition, assessment, and prediction are distinctive purposes of quality. Yet, they are not mutually exclusive; it is difficult to assess quality without knowing what it consists of, and it is even more difficult to predict quality without recognizing how to assess it. The DAP classification in figure 2.1 illustrates this relationship between the different quality purposes (Wagner, 2013).

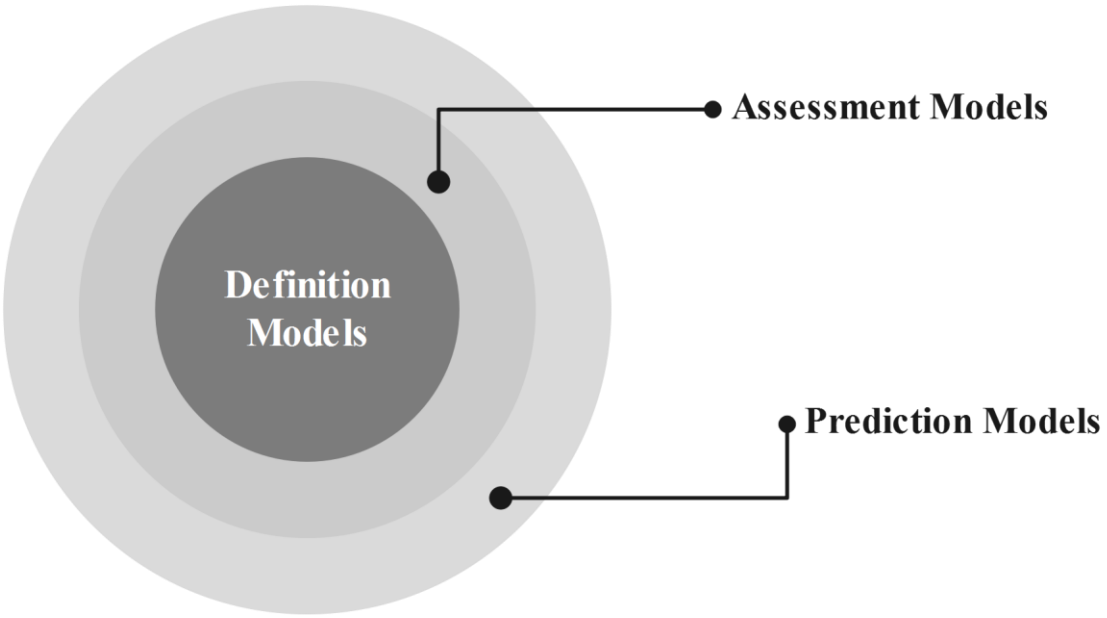

Figure 2.1: Definition, Assessment, and Prediction Classification for quality models.

Definition models are employed at many stages of the software development life cycle. During requirements engineering, definition models define quality characteristics and requirements for software systems, constituting a mechanism for agreement on what quality means to the customer. During implementation, they serve as the foundation for modeling, coding standards, and guidelines. They make immediate recommendations for system implementation and, as a result, constitute constructive approaches to achieving high software quality. Moreover, definition quality models are also used to classify quality defects identified during quality assurance. Eventually, definitional quality models can be used to communicate software quality information during developer training or student education, in addition to being used during software development (Wagner, 2013). The FURPS model is an example for this kind of models (Grady et. al, 1987).

Assessment models usually enhance the definition of quality obtained by the definition quality model in order to perform the subsequent evaluations. Assessment models can be used during requirements engineering to objectively describe and control specified quality requirements. Besides, they can serve as the foundation for all quality measurements during implementation. Using assessment models, we monitor and control internal measures that may have an impact on external characteristics. This concept is very prevalent in software quality engineering; many authors assume that high internal quality indicators will lead to good external quality indicators

(Kitchenham et. al, 1996; Pfleeger et. al, 1998). And since the external quality indicators are visible and reviewed by the customer, developers are required to work on the internal quality level by making the construction phase as good as it can be (Suryn, 2013). The EMISQ method represents a good example of an assessment model based on the ISO standard 14598 for product evaluation (Plösch et. al, 2008).

Prediction models have been used to anticipate the amount of defects in a system or specific modules, as well as the mean time between failures, repair times, and maintenance efforts. Predictions are typically based on source code metrics or previous fault detection data. The reliability growth models (RGM) is a sample for this category (Wagner, 2013).

Multi-purpose models combine all the quality model purposes. The idea behind this kind of models came originally from an ideal quality model which can covers all the purpose at once. Such models are particularly efficient as they can define, assess and predict quality using the very same model. The typical example for a multi-purpose model is COQUAMO (COnstructive QUAlity MOdel) based on COCOMO Boehm's estimation quality model (Boehm, 1981) in the prediction part. The next figure presents the different measures used in COQUAMO, note that every measure with its own purpose (Wagner, 2013).

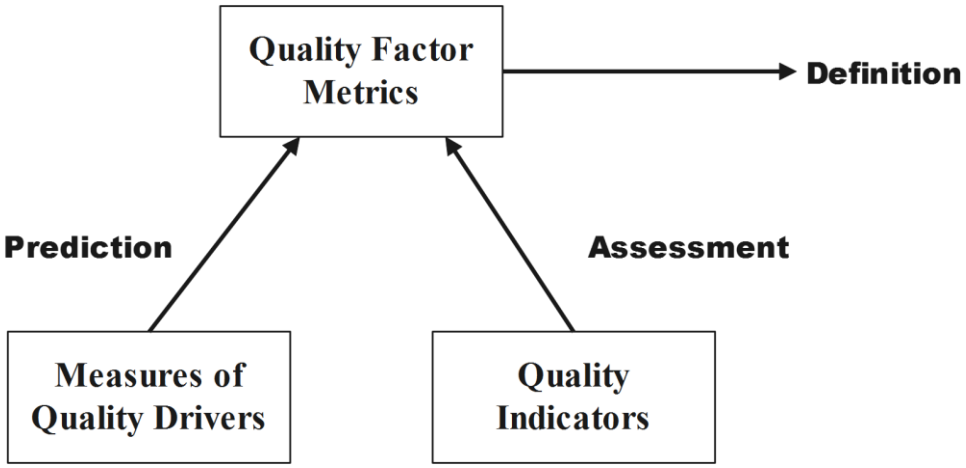

Figure 2.2: Definition, Assessment and Prediction in COQUAMO.

3.2.Quality of Specific Software Products

Since the appearance of the first quality models at the end of the seventies, the field of software engineering has experienced an almost radical change. This change does not

only affect the methods and techniques of development but also affects the paradigms of software development. Several new paradigms have emerged in recent decades. Inherently, quality models must take into account the specific characteristics of different paradigms or certain software products. In this context, two different paths can be distinguished: the development of new quality models to specify the novelties of the targeted product or the adaptation of existing models to express the particularities of the product studied.

In this first path, the works included are based on a simple idea: generic quality models cannot be used to capture the particularities of various specific software products. Thus, the proposal of specific models designed from the outset for specific products represents the alternative trend. Several works have been proposed in this context. By way of example, Bansiya and Davis (2002) note that the introduction of new concepts in the object-oriented paradigm compared to the procedural approach such as encapsulation, inheritance, and polymorphism; has changed the way of evaluating the quality. For this, they proposed a quality model for object-oriented designs called QMOOD (for Quality Model for Object Oriented Design). This model is a hierarchical model composed of three levels: attributes properties and metrics.

Unlike the previous path, another alternative is to take advantage of the existing generic models to specify the quality of particular software products. Many advantages can be attained through the exploitation of existing models, which limits the diversity of quality models that is a source of ambiguity. In addition, it allows the comparison of different products using similar frameworks. As an example for this category, the QM4MAS proposed by Marir et. al, (2016) is a specific quality model for multi agent systems. This model extends the standard quality model ISO/IEC 9126 with many concepts related to the multi agent systems such as autonomy, reactivity, and proactivity.

Table 2.1 illustrates the classification of some quality models based on various criteria.

Table 2.1: Classification of quality models (Tamrabet et. al, 2018).

Criteria	Types	Examples
Purpose of Quality Model	Define	ISO/IEC 9126 (ISO, 2001)
	Assess	EMISQ (Plösch et al., 2008)
	Predict	The maintainability index (MI) (Oman & Hagemeister, 1992)
	Multipurpose	*COQUAMO* (Kitchenham, 1987)
Structure of Quality Model	Hierarchical	ISO/IEC 9126 (ISO, 2001)
	Meta-model Based	*COQUAMO* (Kitchenham, 1987)
	Statistical	The maintainability index (Oman & Hagemeister, 1992)
Scope of Quality Model	Generic	ISO/IEC 25010 (ISO, 2011a)
	Specific	QM4MAS (Marir et al., 2016)

3.3.Quality Models

With a view to determine whether the quality model is useful or not, a quality model should (Suryn, 2013):

- Support all the five perspectives of quality defined by Kitchenham and Pfleeger (Kitchenham et. al, 1996).
- Be usable from the top to the bottom of the life cycle, by allowing the identification of quality requirements into a suitable set of quality characteristics, sub- characteristics, and measures.
- Be usable from the bottom to the top of the life cycle, by allowing the evaluation of the obtained results of the required measurements and their subsequent aggregation.

Therefore, quality models are founded for one particular goal, which is providing an operational definition of quality, and the reason for this is the lack of what constitutes quality in the general sense in software engineering. Several quality models have been established during the previous three decades, some of which have been primarily recognized by the scientific community, while others have gained acceptance within the industry. McCall, Boehm, Dromey, ISO/IEC 9126, and ISO/IEC 25010 are the most well-known quality models. In the next section, we will present a brief overview of each of the aforementioned quality models. Note that the ISO/IEC 25010 quality model

will be discussed later on separately as it constitutes the basis of our upcoming contributions.

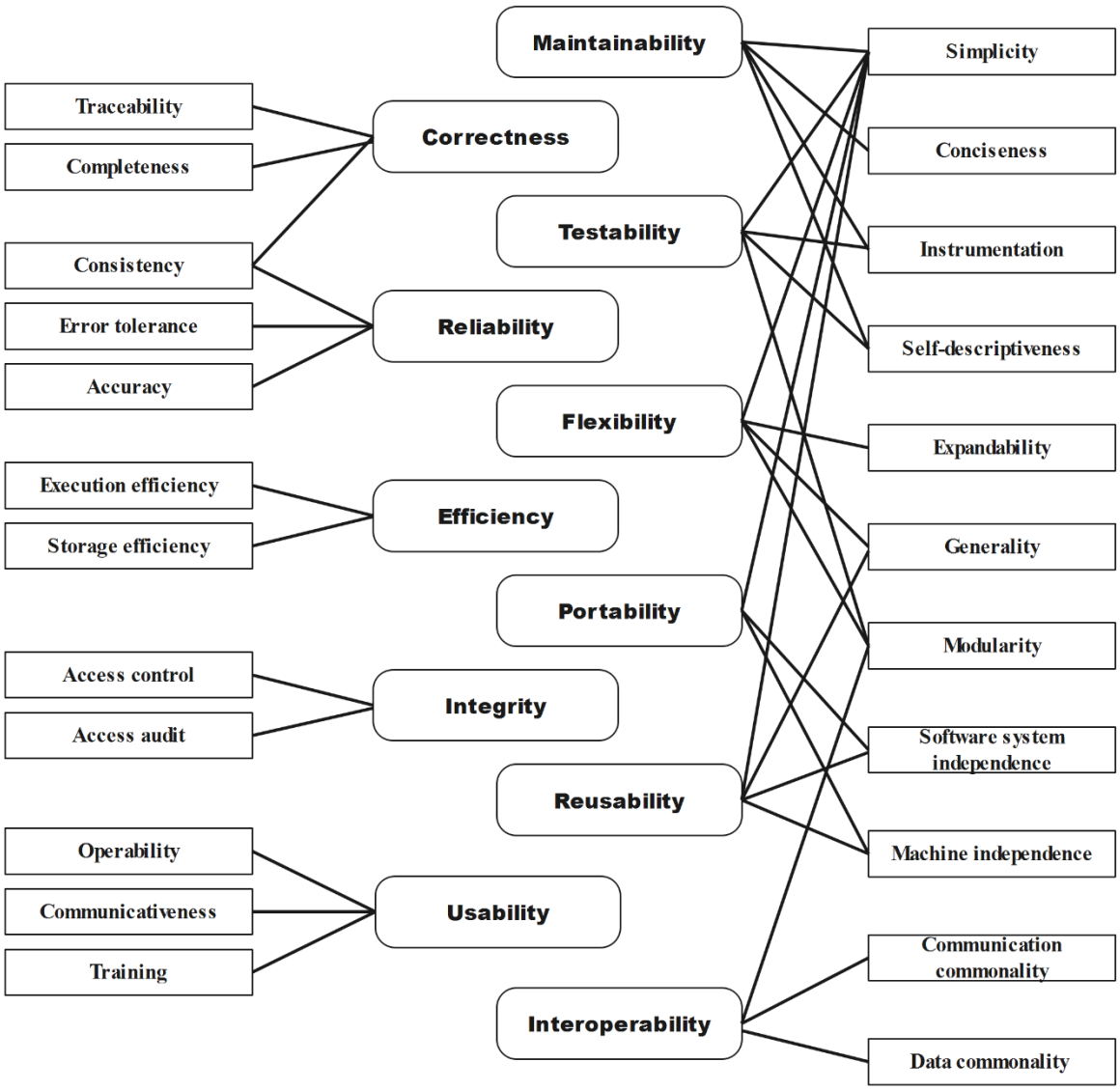

Figure 2.3: McCall's Quality Model (McCall, 1977).

3.3.1. McCall's Quality Model

McCall's quality model was developed in 1977 by McCall and his colleagues in response to an apparent need to describe quality practically and systematically. McCall's quality model is based on a set of eleven factors, which define the behavioral aspects of the system (McCall, 1977). These factors are composed of a different number of attributes. Each quality factor on the higher level of McCall's quality model as depicted in the figure 2.3 indicates a quality aspect that is not directly measurable. On a lower level, there are measurable properties that can be evaluated in order to measure

the quality in terms of the factors. McCall suggests a subjective grading scale ranging from 0 which represents the lowest grade to 10 the highest ones; however, many of the metrics McCall defines can only be measured subjectively. Furthermore, several factors and measurable properties, such as traceability, are not definable or even meaningful to non-technical stakeholders at early stages. From this viewpoint, this model does not meet the criteria stated in the IEEE Standard 1061-1998 for a top-down approach to quality engineering (IEEE, 1998). Furthermore, it is not very applicable in software quality engineering as it prioritizes a quality perspective at the expense of the other perspectives.

3.3.2. Boehm's Quality Model

Boehm's quality model (Suryn, 2013) improves on McCall's work, even though it vaguely preserves the arrangement of measurable properties for each factor, as shown in figure 2.4. The quality factors in Boehm's model are defined as portability, reliability, efficiency, human engineering, testability, understandability, and modifiability, using McCall's model as a reference. These factors can be broken down into measurable properties such as device independence, accuracy, and completeness. With the exception of portability as a special factor, which can be divided into smaller measurable properties, as well as being a top-level element that is a direct part of general utility at the same model.

In contrast to McCall's quality model, Boehm's model is organized in such a hierarchy that the end-user concerns are addressed at the top and the technical staff at the bottom. The measurable properties and characteristics focus on highly technical details of quality, whereas general utility is constituted of as-is utility, maintainability, and portability (Boehm et. al, 1976). These later elements attempt to answer the following questions:

- As-is utility: How well will I be able to operate the software system as it is?
- Maintainability: How easy is it to maintain the software system?
- Portability: Can I still use the software system in case of environment change?

Although Boehm's quality model is a step forward, and improves McCall's quality model in many aspects. Yet, the general utility and as-is utility characteristics are very generic and imprecise in defining verifiable requirements, limiting the model's

application in actual software quality engineering, similarly to the application of McCall's quality model.

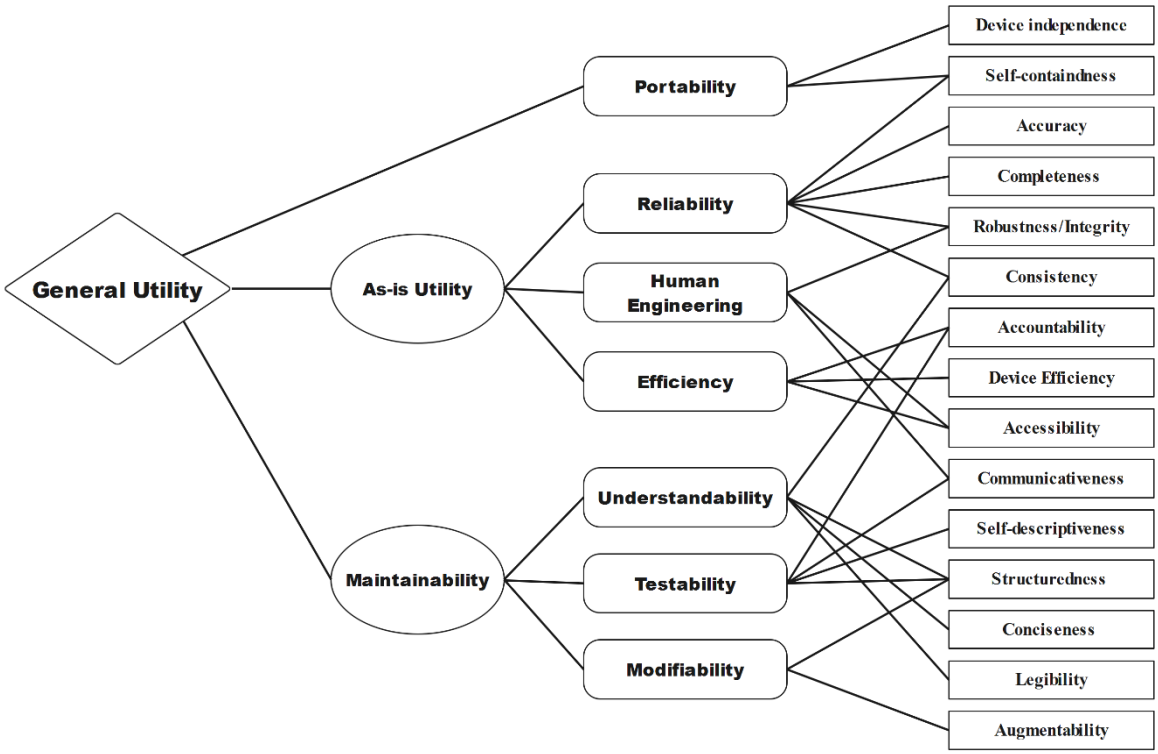

Figure 2.4: Boehm's Quality Model (Boehm et. al, 1976).

3.3.3. Dromey's Quality Model

Dromey's quality model differs from the previous two models in a few ways. Dromey believes that a quality model is more than just expressing a collection of quality characteristics, and he states that you will not make a delicious apple pie just by using a variety of quality ingredients. Aside from quality ingredients, you will also need a good recipe, the consumer's preferred taste, and a qualified person to carry out the recipe properly. From this point of view, Dromey has created a quality evaluation framework, which examines the quality of software components by measuring actual quality properties. A quality evaluation model can be linked to each artifact created during the software life cycle. In addition, Dromey presents the following examples of software components (Dromey, 1995):

- The implementation model can include as components: variables, functions, statements, and other items.

- A requirement is a type of component that can be found in the requirements model.
- A module can be considered as a component of the design model.

According to Dromey, these components all have important properties that can be categorized, as indicated in figure 2.5, into four divisions:

- Correctness: Determines whether or not some fundamental principles have been violated.
- Internal: Measures how successfully a component has been deployed in accordance with its intended usage.
- Contextual: Addresses the external influences that a component has, as well as to its effect while using it.
- Descriptive: Measures a component's descriptiveness (e.g., does it have a meaningful name?).

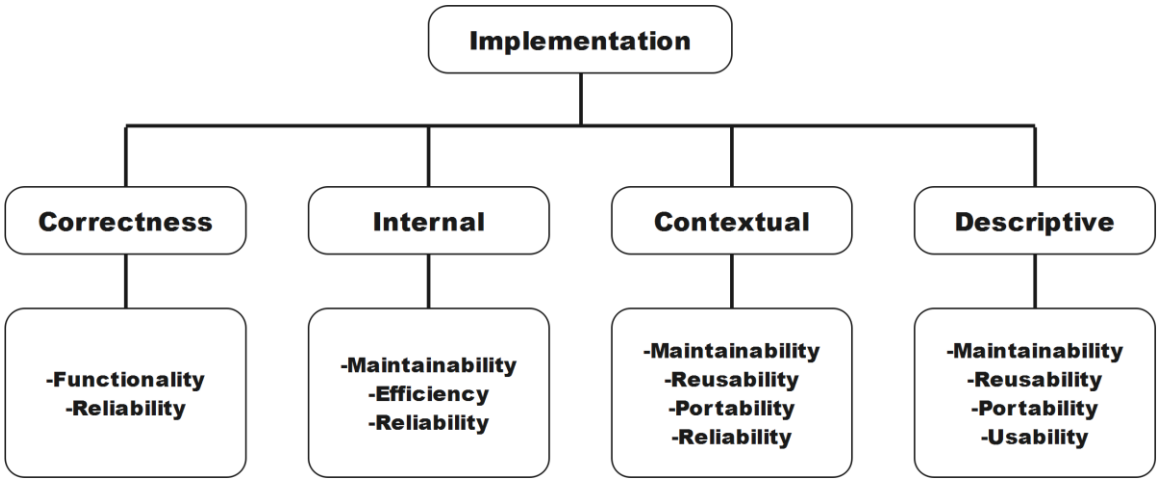

Figure 2.5: Dromey's Quality Model (Dromey, 1995).

Even though Dromey's quality model is remarkable from a technical perspective, it is difficult to see how it could be applied at the start of the lifecycle to establish user quality needs. As reported by Dromey, software quality must be considered from the tangible properties to the intangible, with more focus on the tangible ones. By doing so, Dromey ignored the aspects that allow one to develop a useful model for stakeholders that are usually involved at the beginning of the lifecycle. Consequently, this model is unable to specify user quality requirements. In addition, it fails to qualify as useful from software quality engineering perspective since this model is classified only as a bottom-up approach to software quality.

3.3.4. ISO/IEC 9126 Quality Model

The International Organization for Standardization (ISO) released ISO/IEC 9126 (2001): Software product evaluation – Quality characteristics and guidelines for their use. The goal of this standard was to provide a quality model for software as well as a set of rules for measuring the characteristics associated with it. ISO/IEC 9126 initially piqued the interest of IT professionals; nonetheless, numerous significant issues were linked with its initial release:

- No guidelines available for providing an overall assessment of quality.
- No indications presented for measuring the quality characteristics.
- The model primarily expressed the developer's perspective on software.

This led to the revision of the standard, creating another release that contains four parts:

- ISO/IEC 9126-1: defines an updated quality model (ISO, 2001).
- ISO/IEC 9126-2: defines a set of external measures (ISO, 2003a).
- ISO/IEC 9126-3: defines a set of internal measures (ISO, 2003b).
- ISO/IEC 9126-4: defines a set of quality in use measures (ISO, 2004).

The updated ISO/IEC 9126-1 quality model recognizes three aspects of software quality, which has been defined as follows:

- Internal quality: the set of product attributes that determine its ability to satisfy implicit and explicit needs when used under specific conditions. Although details of software product quality can be enhanced throughout code implementation, reviewing, and testing, the essential essence of the software product quality represented by internal quality remains intact unless modified (ISO, 2003b).
- External quality: it is the set of characteristics of the product according to an external point of view. It is the quality of the software when it is executed that is often measured and evaluated using external metrics while testing in a simulated environment with simulated data (ISO, 2003a).
- Quality in use: Quality in use refers to the user's perception of the quality of a software product when it is utilized in a specific context of use. Rather than measuring the properties of the software itself, quality in use measures the

amount to which users can achieve their goals in a given context of use (ISO, 2004).

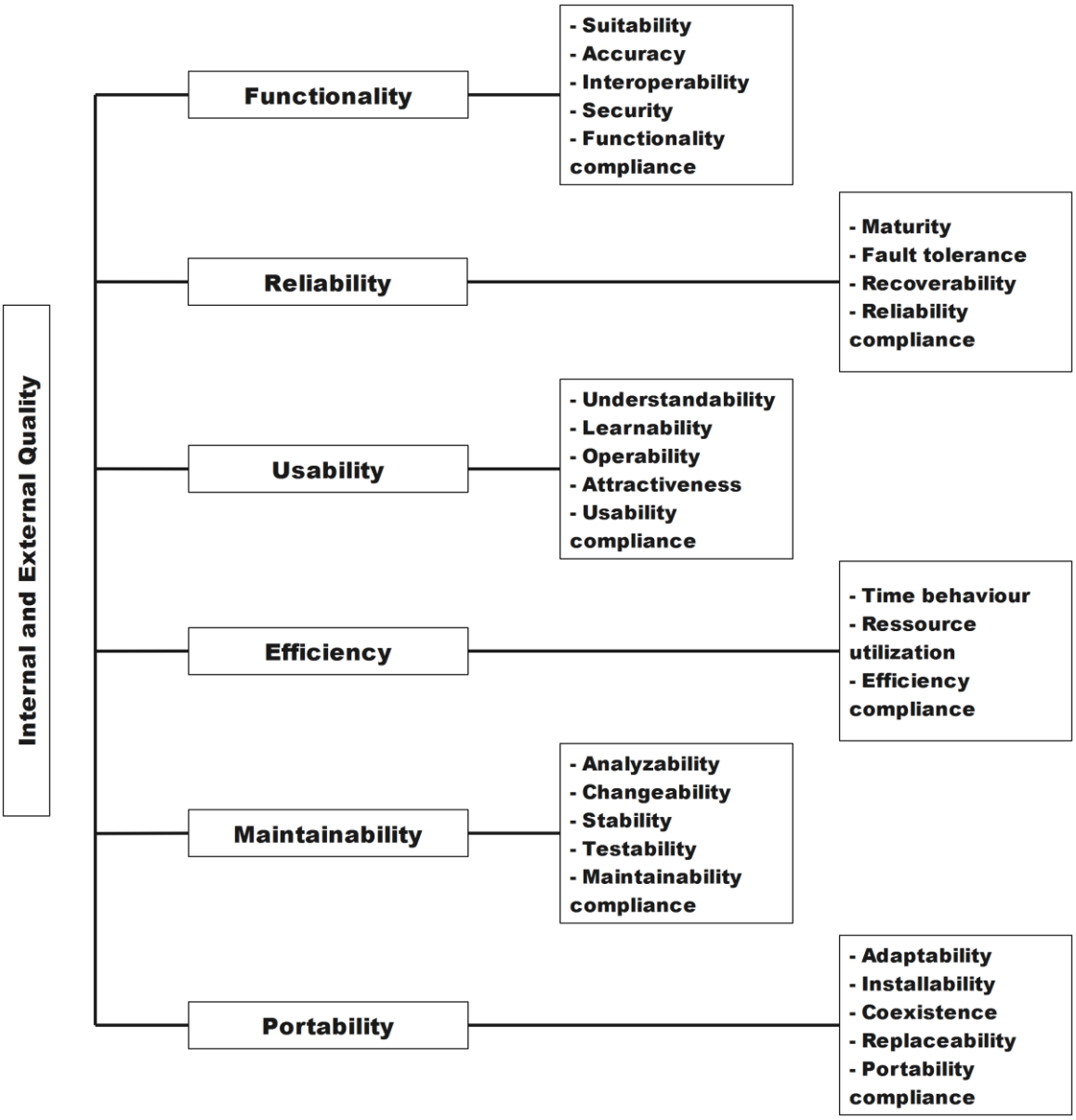

Figure 2.6: ISO/IEC 9126 internal and external quality model.

The internal and external aspects of quality were modeled by a hierarchical model organized in six quality characteristics: functionality, reliability, usability, efficiency, maintainability and portability. Each characteristic is divided into sub-characteristics. In addition, each sub-characteristic can be evaluated by a set of measures. This standard offers a list of more than two hundred metrics for measuring internal and external quality. Nevertheless, it is possible to extend this list by additional measures because of the non-exhaustive nature of the proposed measures. Figure 2.6 shows the

relationship between the characteristics and sub- characteristics of the ISO/IEC 9126 quality model.

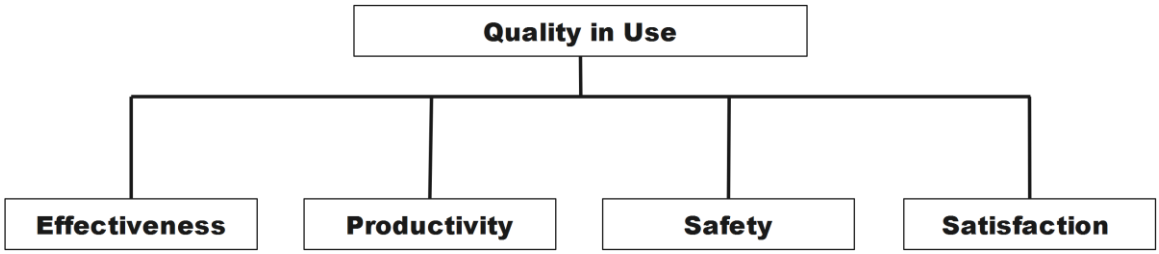

Figure 2.7: ISO/IEC 9126 quality in use model.

In contrast to the internal and external quality, quality in use is made up of four characteristics as shown in figure 2.7: effectiveness, productivity, safety and satisfaction. The different characteristics can be directly measured by a set of proposed measures.

Eventually, the ISO/IEC 9126 quality model is compatible with the different perspectives of quality proposed by Kitchenham and Pfleeger (Kitchenham et. al, 1996) that we mentioned earlier (Suryn, 2013). In fact, the model addresses four out of five perspectives; the internal and external quality represents the manufacturing perspective and the product perspective. In contrast, the quality in use model represents the value-based perspective and the user perspective. Of course, the transcendental perspective cannot be explicitly represented because of its nature. Therefore, the ISO/IEC 9126 quality model is qualified to be useful from software quality engineering perspective.

4. SQauRE Series of Standards

The ISO (the International Organization for Standardization) and the IEC (the International Electrotechnical Commission) developed countless international standards for particular products in many different fields. Software products make no exception among these products, the two standards ISO/IEC 9126 and ISO/IEC 14598 were proposed respectively for *Software Product Quality* and *Software Product Evaluation* (ISO, 2005). Nevertheless, these two standards are considered as complementary set of standards, and they share several normative, referential and functional roots as well. Moreover, there is a disagreement between them since they are used independently in the life cycle of the software product. As a result, the SQuaRE (Systems and software Quality Requirements and Evaluation) series of international

standards has been created to replace the predecessor standards, in addition to obtaining more consistency and coherent standards (ISO, 2005a).

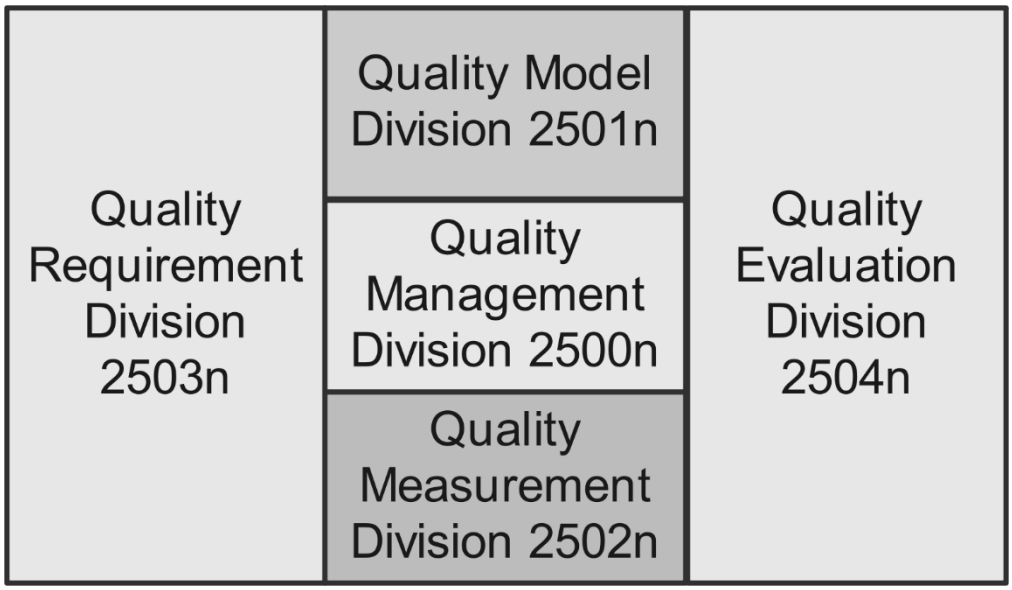

Figure 2.8: Organization of the SQuaRE (ISO, 2005a).

Compared to the previous standards, the SQuaRE is richer, more logically organized, and unifies a series of standards (ISO, 2005a). It is decomposed into five major divisions as depicted in figure 2.8, each division with a certain goal: Quality Management Division (2500n), Quality Model Division (2501n), Quality Measurement Division (2502n), Quality Requirements Division (2503n), and Quality Evaluation Division (2504n) (ISO, 2005a).

Starting with the division of standards called *quality management*. The first standard in this division is ISO/IEC 25000 which is the earliest among all standards, considered as a guideline that explains how to use these series as well as providing a complete overview of them (ISO, 2005a). The main contribution of this standard is the set of definitions and terms used across all the standards. The second standard is ISO/IEC 25001 that gives a clear description for the activities across the system life cycle (ISO, 2007a).

The second division is *quality model*. In this division, the standard ISO/IEC 25010 (ISO, 2011a) is of paramount importance as it presents quality models that are the cornerstone in quality control. This standard is based totally on the previous standard ISO/IEC 9126 (ISO, 2001) it defines quality as a set of characteristics and sub-characteristics that were previously known as quality factors. In addition, the standard

describes the quality according to a conforming meta-model. In other words, the standard provides definitions corresponding to quality as same as the previous standard with new structure for both models product quality model and quality in use model. Another quality model for data is defined in ISO/IEC 25012 (ISO, 2008), a compliment quality model for ISO/IEC 25010 which serves later on, to give more specific definition for the important characteristic.

Presented in the third division of this series the most complicated phase in quality evaluation, which is *quality measurement*. It is important to note that quality measurement is a part of quality evaluation, but due to its complications, it is carried out as a completely separate division. The standard ISO/IEC 25020 (ISO, 2007b) defines several basic terms and definitions such as quality measure element, which is the main concept in this section. Besides, it describes the measurements related to these quality measure elements. Quality measure elements are considered as atomic measures, with which we can obtain ones that are more complex. These quality measures are typically used as indicators for quality characteristics and sub-characteristics as shown is the next figure. Another standard is ISO/IEC 25021 (ISO, 2012a) dedicated for quality measure elements and the measurement methods associated with them. Moreover, the standard ISO/IEC 25023 (ISO, 2016) provides a set of quality measures for the characteristics of system/software products that can be used for specifying requirements, measuring and evaluating the system/software product quality.

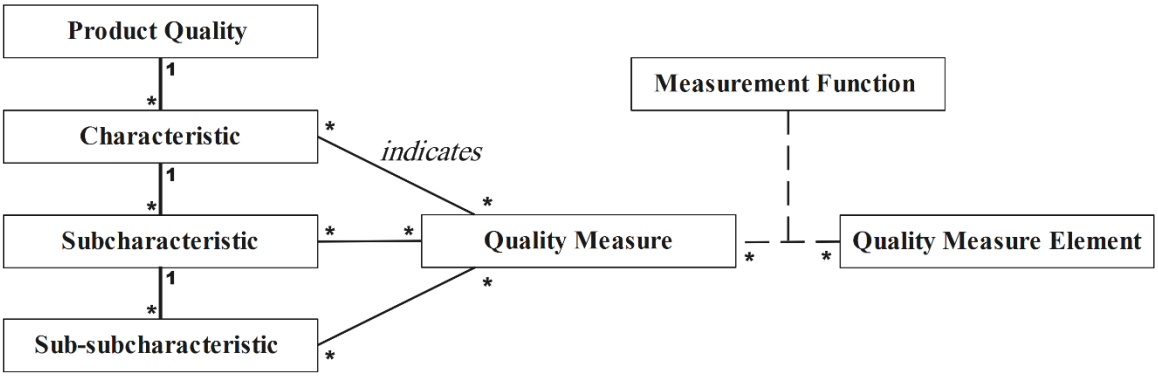

Figure 2.9: Software product quality measurement reference model of ISO/IEC 25020 (Wagner, 2013).

Among the implicit uses of the above-mentioned standards is the description of quality requirements that the user often fails to describe, and remains restricted to mention

functional requirements. The fourth division in this series presents *quality requirements*, which is a central key to master and better understand the term quality. The standard ISO/IEC 25030 (ISO, 2007c) describes the possibility of building quality requirement based on quality measures and their corresponding quality characteristics.

Finally, the fifth division in this series is *quality evaluation*. As we have mentioned earlier, this division is considered as the larger division of quality measurement that is only confined with collecting data and applying the measurements. Rather, quality evaluation transforms the measurements into ratings for easier understanding through providing certain interpretations for each collected data and measures. The standard ISO/IEC 25040 (ISO, 2011b) defines a software product quality evaluation reference model. Besides, just as its name indicates, this model is relied on as a reference for the evaluation process, as well as the evaluation records that are used for documenting the performed activities and their corresponding results. This division contains also 25041 and 25045 (ISO, 2012b; ISO, 2010b).

4.1.The ISO/IEC 25010 Quality Model

The ISO/IEC 9126 quality model has not experienced major changes since its last version appeared in 2001, In fact, an ISO working group has started an initiative to prepare the second generation of software product quality models. As a result, a series of models called SQuaRE (for Systems and software Quality Requirements and Evaluation) has been proposed for this purpose (Suryn, 2013). This series consists of fourteen models. Among these models the ISO/IEC 25010 as a new quality model.

The ISO/IEC 25010 quality model proposed within this series finds its continuity in the same approach proposed by its predecessor, the ISO/IEC 9126 quality model. Thus, it provides two unique perspectives of quality: the quality in use model and the software/system product quality model. Moreover, in opposition of the ISO/IEC 9126, the ISO/IEC 25010 quality model addresses quality not only for software but also for systems.

The quality in use model (Figure 2.10) has additional characteristics compared with the ISO/IEC 9126; it is composed of five characteristics. Besides, it introduces eleven new measures. The objective of this model is to define characteristics that relate to the outcome of the user-system interaction in a certain context of use (Suryn, 2013).

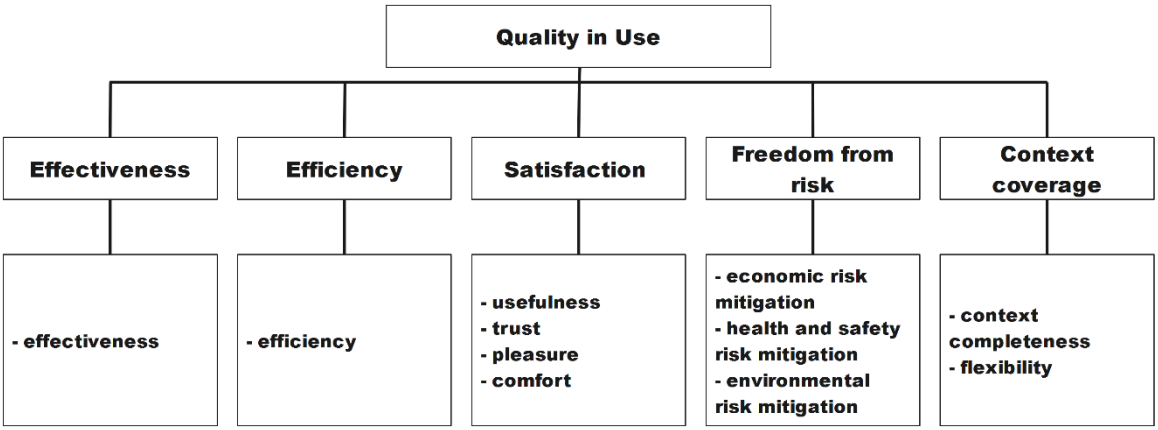

Figure 2.10: ISO/IEC 25010 quality in use model.

The system/software product quality model is composed of eight characteristics, which in their turn subdivided into sub-characteristics (Figure 2.12). The product quality model represents the internal and external quality model in the predecessor model. Obviously, the ISO/IEC 25010 product quality model not only includes more characteristics than its predecessor, but also the list of sub-characteristics has been updated by introducing some, and removing others (Suryn, 2013).

Furthermore, one of the model's distinct advantages is the existence of a defined meta-model; this is considered as crucial for quality models. The next UML class diagram represents the meta-model of the ISO/IEC 25010 quality models. A meta-model is required to define model elements and their interactions by showing the hierarchical structure that divide quality into a set of characteristics and sub-characteristics. According to Stefan (Wagner, 2013) the terms quality factor and measurable are not part of the standard, but they are introduced to make a well-structured meta-model.

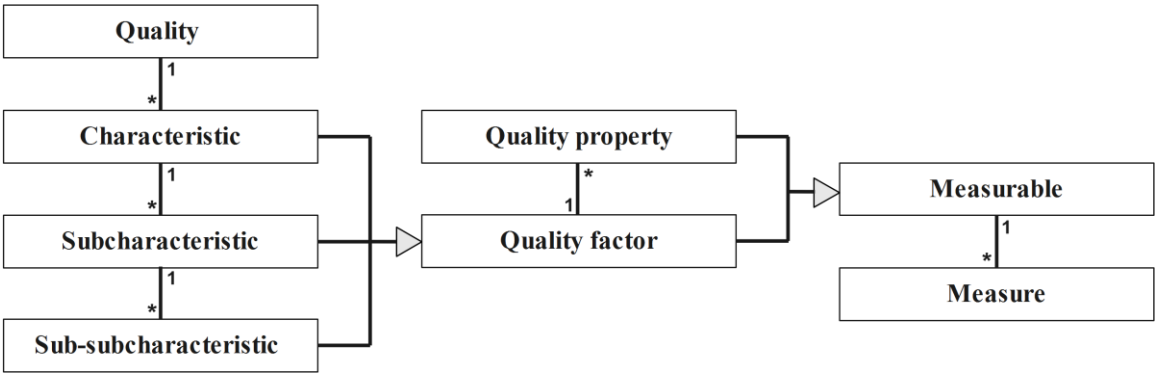

Figure 2.11: The meta-model of the ISO/IEC 25010 quality models (Wagner, 2013).

Compared to the ISO/IEC 9126 model, the revisions proposed in the ISO/IEC 25010 model can be summarized in the following aspects (ISO, 2011a):

- The model has been extended to support software systems.
- Security has been added as a characteristic instead of being a sub-characteristic in the ISO/IEC 9126 model.
- Compatibility was added as a new characteristic.
- The list of sub-characteristics has been modified by removing sub-characteristics or by adding new ones.
- The model considers only one perspective of product quality, which encompasses the two facades: internal quality and external quality.
- The existence of an implicit meta-model that helps in the comprehension and the interpretation of the ISO/IEC 25010 quality model.

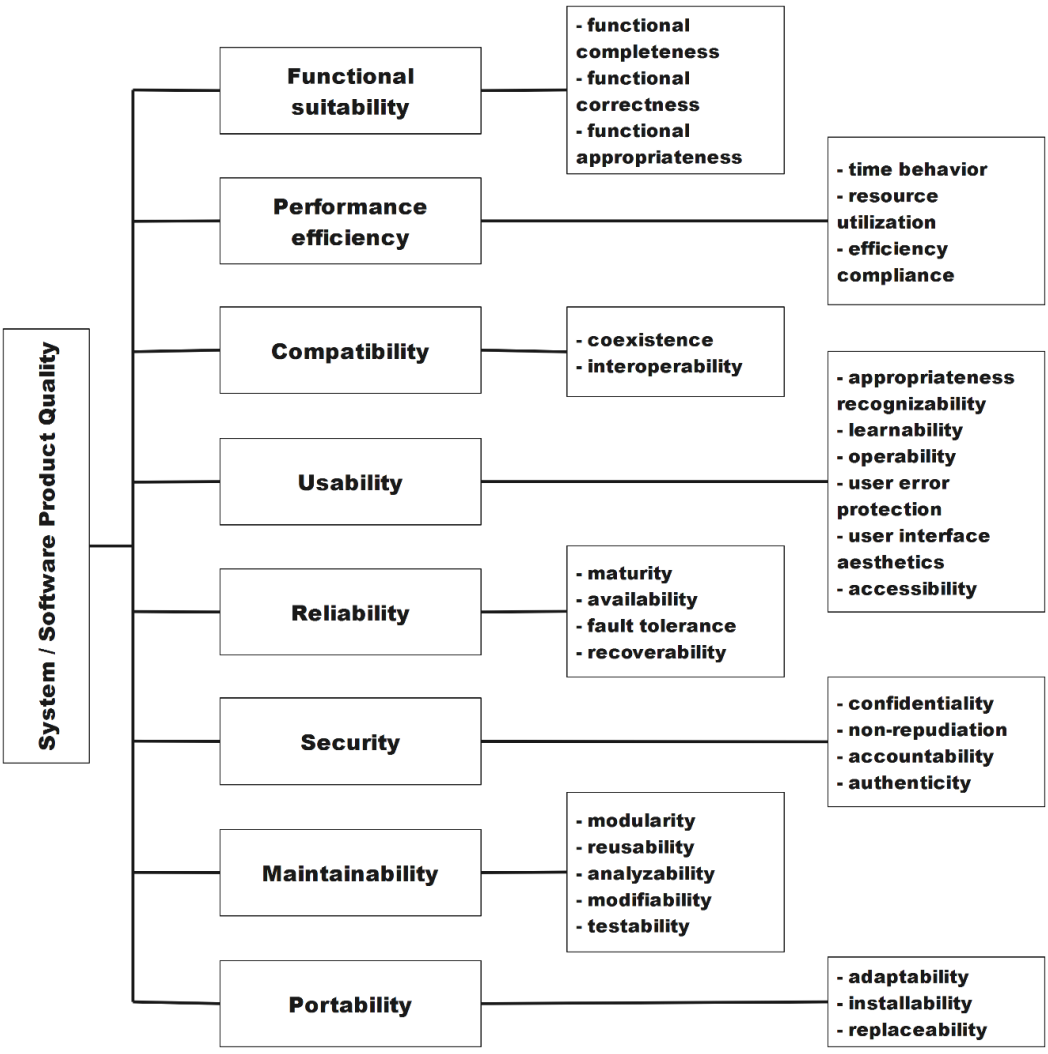

Figure 2.12: ISO/IEC 25010 product quality model.

As the ISO/IEC 9126 quality model was previously designated as a useful model from software quality engineering perspective, the ISO/IEC 25010 as a modernized version is qualified to be useful as well (Suryn, 2013).

5. Conclusion

Quality is a crucial requirement of software products. This requirement incorporates the diversity of actors' perspectives participating in software development, the abstraction levels of software products, and the essential properties for each type of software. As a result, software quality assurance is used to develop quality software products. In software quality assurance, the adoption of quality models is considered among the most promising ways to improve the quality of software products. In this chapter, terms and definitions related to the context of quality assurance are introduced. In addition to the presentation of the most acceptable quality models in the specialized literature. As well as highlighting the importance of standardization in providing a unified view of the complex term quality. Finally, we addressed in depth the ISO/IEC 25010 Quality Model, which constitutes the basis for our subsequent contributions.

Chapter 03

Quality Assurance for Embedded Systems

1. Introduction

In the context of embedded systems, quality represents a fundamental query in the development of such systems. In fact, quality is expressed through a set of quality attributes that refer to systems quality aspects like performance, usability, security, etc. (Muhammad et al., 2010). Nevertheless, there is no common consensus on how to identify quality attributes of a system in general and of an embedded system in particular (Sherman, 2008; Wijnstra, 2001; Choi et al., 2008; Jeong & Kim, 2012; Oliveira et al., 2013). Moreover, the presence of many taxonomies for the same quality concept according to different points of view further complicates the identification process of these quality attributes (Suryn, 2013; Wagner, 2013).

In addition, most embedded systems belong to the critical systems category. Consequently, the development of such systems with poor quality can produce financial, health, and even life losses. We can refer to the bug discovered in the embedded software of Volkswagen cars that affects the fuel consummation (Nikolaus, 2015). Thus, mastering the quality of an embedded system is a difficult task. The reason behind this difficulty stems mainly from the nature of the software (abstract, intangible, etc.), on the one hand. On the other hand, this is due to the nature of embedded systems, particularly the integration of the software component in the overall system (mechanic, electronic, etc.) (Wagner, 2013). As a result, software quality engineering for embedded systems, as for any other software system, is an indispensable activity in order to produce secure, safe, functional, and reliable systems.

In this chapter, we present our first contribution, a survey of the most important works related to embedded software quality engineering. The choice of this context is justified by the criticality of embedded systems. As indicated in several parts of this thesis, the absence of quality in such systems may lead to serious damage. In addition, a detailed description of each work is also provided, along with its strengths and weaknesses. Finally, a comparative study is given at the end of this contribution.

The remainder of this chapter is organized as follows. Section 2 is dedicated to presenting the state of the art of embedded software quality engineering. Section 3 presents the most important works dealing with software quality attributes and quality models in the context of embedded systems. Section 4 provides a comparative study

between the most important works of the previous section. Conclusions are provided in section 5.

2. Embedded Software Quality

The quality of embedded systems is an extreme important topic because of their wide range and critical use. The importance of this topic attracts more and more the scientific community. Recently, the studies in this topic targeted several aspects like techniques and methods of embedded software quality assurance and control (IEC, 2006; ISO, 2011c; RTCA, 1992), specific development methods to ensure the quality of such software (Bedoya et al., 2016; Shah et al., 2016; Xu & Zhuang, 2014) and the quality attributes for embedded software (Sherman, 2008; Wijnstra, 2001; Choi et al., 2008; Jeong & Kim, 2012; Oliveira et al., 2013). We present briefly some studies related to the embedded software quality before presenting a survey of quality models and quality attributes for such software.

Developing high-quality products by following high-quality processes is a well-known concept in the industrial sector. As a result, quality assurance can be achieved by ensuring process quality. This concept has been used in software engineering since the first studies of software quality (Pressman, 2010). The Capability Maturity Model (CMM) (Paulk, 1993) is the most famous example of a process quality model. In the case of embedded systems, several standards are proposed for the different categories of such systems. Thus, the IEC 62304 (IEC, 2006) is proposed to specify the requirements of the life cycle of software for medical equipment, the ISO 26262 (ISO, 2011c) is proposed for the automotive industry, and the DO-178 (RTCA, 1992) that specifies the development constraints of the avionics software systems.

The Verification and Validation (V&V) techniques play a central role in the quality assurance of the embedded software. Recently, Bedoya et al. (2016) presented a review on these techniques. In fact, the authors assert that the growing demands and needs of the users make the embedded systems reach a higher level of complexity, which in turn increases the different activities of verification and validation in the context of product development. In addition, they classified the V&V techniques into two categories: static techniques that do not require the execution of the software and dynamic techniques which require the execution of the software. Each technique has a precise goal and can be applied during adequate phases in the development cycle.

We think that testing is among the most applied V&V techniques on embedded software. This process represents a great challenge, specifically in the context of embedded software systems. However, if this process is properly conducted, it may provide an efficient way for error identification. The diversity of studies that targeted this technique is due to the variety of the considered characteristics during test performance. Shah et al. (2016) interviewed experts in seven different embedded systems domains (telecommunication, automotive, multimedia, critical infrastructure, aerospace, consumer products, and banking) to explore the current state-of-the-practice regarding the robustness testing of embedded systems. In this study, the authors claim that dependability of an embedded software system is critical to the success of almost any industrial application, and testing its robustness is one method of determining a system's dependability. This interview included several questions about the definition of the robustness, the key aspects of the robustness of embedded software, requirements for robustness testing, design for robustness testing, performing robustness testing, robustness failure, and robustness testing tools. The main results of this study, in our opinion, are the lack of a systematic test design in almost cases and there is neither dedicated staff to perform testing nor robustness-specific testing. This study shows a gap between the state-of-the-art and the state-of-the-practice in robustness testing because the proposed issues and methods in the literature are not effectively used in practice.

Given that high-quality software must be based on requirements specifications (Xu & Zhuang, 2014), formal specifications for embedded software might provide quality assurance for such systems. However, some languages cannot provide a precise semantic that allows analyzing and reasoning the properties of the system. A typical example of such language is MARTE language, which is an extension of UML to specify real-time and embedded systems (MARTE, 2015). In order to deal with this problem, an approach based on MDA (Model Driven Architecture) to transform MARTE specification to Object-Z model is proposed (Xu & Zhuang, 2014). This contribution allows enhancing the reliability of the embedded systems by providing an effective method as a support tool in the initial phases of system design. The main goals of the proposed transformation rules are providing precise specifications and verifying the correctness of the systems in the early stages of development. Hence, the reliability of the future software can be ensured.

Despite that all the previous works addressed the quality aspects of embedded software, they did not address the quality as a whole concept. In fact, robustness, reliability, and even performance are only some aspects of the quality of embedded software. We do believe that quality is more a lot than just some individual quality aspects. This concept as a whole is even more than the sum of these aspects. As a result, modeling the quality of such software is a mandatory requirement. In the next section, we present the different embedded software/systems quality attributes and models.

3. Embedded Software Quality Attributes and Quality Models

We have introduced quality as a complex term decomposed generally into smaller entities called attributes. Moreover, the quality models must be tailored to each specific software category. Particularly, we need to propose specific quality models to the embedded software as an important category of our current software. In fact, modeling the quality of embedded systems allows for a good specification of this concept. Consequently, stakeholders can define their requirements more precisely. Moreover, they can evaluate objectively the developed products according to a well-defined reference. These models show explicitly links between systems' attributes (like modularity, reusability, testability, etc.) and high-level quality characteristics (like maintainability, usability, security, etc.) on one hand. On the other hand, they show the relationships between these high-level characteristics and the whole quality. As a result, we can predict the quality of products under development by using the specified attributes of such products. Therefore, establishing quality models for embedded systems must begin with identifying the attributes that influence their quality. Indeed, the identification of quality attributes of embedded systems is a very active area. In this section, we summarize the main contributions in this area in order to elicit their advantages and shortcomings.

According to Sherman (2008), embedded systems should provide a set of functional requirements to achieve a certain purpose they were developed for. However, it is important to provide the non-functional requirements as well, which have been expressed for many years by quality attributes. Besides, the author affirms that there is no unique ultimate software quality attributes list, but they change according to the customer needs and the developer's focus on quality. The analyses made in this paper proved that embedded systems and software systems share several quality attributes.

Nevertheless, the author proposed to modify the definitions of some attributes in order to be more adequate to the embedded systems. Moreover, he proposed particular attributes related to embedded systems. Despite the importance of identifying the quality attributes of embedded systems, this work suffers from various lacks. Firstly, the author proposed all the attributes in the same level of abstraction. Different levels of quality are usually distinguished in the specialized literature, by dividing the concept of quality into characteristics and sub- characteristics. This is mainly due to the quality itself as a multi-perspective concept. Secondly, the relationships between the quality attributes are completely omitted. Although these relationships are extremely important in the concept of quality as a complex entity.

Wijnstra (2001) asserts that the usage of quality attributes and quality aspects introduces more than a single decomposition of the system. On one hand, quality attributes represent the observable properties of the system. On the other hand, quality aspects represent what should be realized inside the system to make it a quality one. As examples of quality aspects, we can mention initialization, error handling, logging, etc. In contrast to the previous work, this work presents a practical differentiation between the quality aspect as an internal view of quality, and the quality attributes as external ones. However, the method of differentiating the aspect from the attribute is still problematic in this work as well. Besides, the lack of clear criteria to distinguish the quality attributes from quality aspects, the proposed attributes and aspects are discussed only for the medical image product family context.

Based on the ISO/IEC 9126 quality model, Choi et al. (2008) proposed a hierarchical quality model for software components called SCQM (Samsung S/W Component Quality evaluation Model). This model is composed of eight characteristics including functionality, reusability, portability, maintainability, usability, reliability, modularity, and efficiency. These characteristics are related to twenty-two sub-characteristics. Although the model presented in this work is specific to component software, the new characteristics defined in this model (such as Reusability and Modularity) reflect the features of the software in general and not those that are supposed to be specific to component software. In addition, we think that extending new quality models using the ISO/IEC 9126 quality model is no longer desirable, due to the availability of its successor the ISO/IEC 25010 quality model. In fact, this remark is not limited to the SCQM quality model. Jeong and Kim (2012) tried to identify the most important quality

attributes from ISO/IEC 9126 quality model and extend the DeLone and McLean (2003) success model by the identified quality attributes to make it fit lightweight component embedded systems. The DeLone and McLean (2003) success model is used again by Jeong et al. (2014) to model the quality of secure embedded systems with sensor networks. Hence, quality attributes have been proposed to support the specifications of these systems. In this study, an Analytic Network Process (ANP) is used in order to take into consideration the correlation between the quality criteria. The proposed quality model for secure embedded systems consists of five criteria including system quality, information quality, function quality, efficiency quality, and maintenance quality.

The SCQM (Choi et al., 2008) quality model is proposed for component software because this paradigm is suitable to develop embedded systems. In addition, the complexity of embedded systems increases the demand for the assessment and evaluation of software components. Hence, Carvalho and Meira (2009) introduced an embedded software component quality verification framework, which includes an embedded software component quality model (EQM). The characteristics presented in the EQM are the most important in the embedded system domain. Furthermore, the authors proposed a new characteristic, called marketability, in order to enhance the credibility of the information given by the quality model.

Basically, identifying and defining the important relevant criteria to the domain is the stepping stone process in the development of any quality model. Thus, Ahrens et al. (2013) presented a quality model, which focuses on non-functional requirements of the automotive embedded software domain. The quality model is composed of seven quality characteristics. Each characteristic is decomposed into quality attributes. These quality characteristics and sub-characteristics are highly adapted and specialized to the specific needs of the automotive embedded software systems. Moreover, a set of metrics are developed and presented as well for each of the relevant quality attributes in the proposed quality model. It is important to note that there is no interrelation between the different quality characteristics. This gives the assumption that quality characteristics do not influence each other, but in reality, it is not. As we have seen in detail in Chapter Two of this thesis, managing a specific quality characteristic may have a negative effect on others.

Also, embedded systems could be integrated with other systems in order to provide more complex and sophisticated functions which could not be provided by any system separately. This integration opens the door to a completely new class of systems called Systems of Systems (SoS). For that, Nakagawa et al. (2013) have investigated the state of the art in this topic through a systematic review. They identified the most relevant quality attributes of Systems of Systems Software Architectures. However, quality is not only the result of a well-defined architecture. Indeed, the quality attributes must cover functional and behavioral aspects besides the structural aspect. Bianchi et al. (2015) targeted also the quality attributes of SoS. Additionally, they analyze the coverage of these quality attributes by the ISO/IEC 25010 standard quality model. Hence, five quality attributes are considered as the most relevant for SoS (security, interoperability, performance, reliability, and safety). In addition, the analysis shows that many quality attributes related to SoS are not covered by the quality model ISO/IEC 25010 such as complexity, dependability, robustness, survivability, etc. We think that the results of this study are essential toward a quality model for such systems.

For Garcès et al. (2017) the adoption of quality models and identifying critical quality attributes that address the non-functional requirement of certain products is a well-accepted way to improve quality. However, there is a lack and no consensus that determines which quality attributes are relevant for Ambient Assisted Living (AAL) systems. By the application of the systematic mapping technique, fourteen quality characteristics and their sub-characteristics in the context of AAL are identified. These characteristics and sub-characteristics were mapped into the standard ISO/IEC 25010. Some characteristics make no evidence in the AAL domain though they are defined by the standard. In opposite, adaptivity is a significant characteristic of the AAL domain, yet it is not defined in the standard. The findings of this work can be resumed in the necessity to develop a quality model that defines the common quality attributes for AAL systems, as well as considering the variability of such systems. We do believe that this step is important to build a more fitting quality model for these systems.

Despite that testing is a well-known V&V technique in software engineering, it is more challenging in embedded systems similar to cloud systems because of architectural, platform, and software issues. Consequently, Kiran and Simons (2016) studied testing of cloud ecosystems. The authors presented a state-of-the-art in this field and they concluded that most authors focused on web services, functional testing, and quality of

services. Thus, they proposed a framework Quality-as-a-Service (QaaS) which integrates quality issues like functional behavior, performance monitoring, and security.

Considering the Internet of Things (IoT) as one of the modern embedded systems, most works have not targeted the identification of their quality attributes or their quality models. However, the research community in this field attempt to propose architectures with some required quality attributes for specific applications. For example, Kantarci and Mouftah (2014) realized that the large amount of connected embedded devices that constitute IoT is an ideal platform for crowed management. Then, they proposed architecture for this purpose where safety is its main characteristic. Though safety is an essential requirement for such an application, we think that some other attributes cannot be omitted (like reliability, efficiency, security, etc.). Taking reliability as an example, we can make out that unreliable platforms can break down. Moreover, the heterogeneity of connected objects in IoT implies the emergence of security and privacy requirements. In fact, objects can access useless data as well as sensitive data. Hence, Sicari et al. (2014) proposed architecture that ensures the security and the quality of data. The security is expressed by authentication, confidentiality, integrity, and privacy attributes. On other hand, the data quality is specified by timeliness, completeness, accuracy, and source reputation attributes. According to our viewpoint, this work identified the attributes of the security and the quality in such systems in addition to proposing architecture that ensures these attributes. However, we think that the quality is not limited to data. Harmful processing can generate dangerous results that affect the security and privacy of users. Moreover, we think that security is an attribute among the attributes of quality.

In the context of quality attributes and quality models, systematic reviews are usually used to present a detailed panorama of the subject. Oliveira et al. (2013) for example, conducted a systematic review in order to identify software architectures and reference architectures in the context of embedded systems. In this study, many concerns of embedded systems architectures are found in the primary studies including dependability, adaptability, performance, safety, etc. On the other side, Guessi et al. (2012) made a systematic review to investigate quality attributes for embedded systems. The identified quality attributes are based on the number of cites in the primary studies.

However, we think that citing an attribute is not a sign of its importance. Sometimes a quality attribute can be cited negatively in research papers.

Finally, we summarize through the following table (Table 3.1) the most important quality features that were mentioned in the previous works in the context of embedded systems. It is important to note that these quality features are arranged according to three categories. The first category includes all the quality features that are considered high-level characteristics in the standard quality models. While the second category includes the quality features that are considered sub-characteristics of the second level in the standard quality models. As for the third and final category, it contains the relevant characteristics that are often mentioned to distinguish the embedded software.

Table 3.1: Quality attributes of embedded systems (Tamrabet et. al, 2018).

	Characteristics							Sub-characteristics							Specific attributes		
	Reliability	Usability	Maintainability	Portability	Performance	Security	Functionality	Availability	Fault tolerance	Interoperability	Adaptability	Recoverability	Reusability	Testability	Efficiency	Safety	Flexibility
(Wijnstra, 2001)	✓			✓												✓	
(Sherman, 2008)	✓	✓			✓	✓	✓	✓								✓	
(Choi et al., 2008)	✓	✓	✓	✓		✓	✓		✓		✓	✓	✓	✓	✓		
(Carvalho & Meira, 2009)	✓	✓	✓	✓		✓	✓		✓			✓			✓	✓	
(Guessi et al., 2012)	✓		✓		✓				✓	✓	✓					✓	
(Jeong & Kim, 2012)	✓	✓	✓			✓		✓	✓	✓	✓	✓			✓		
(Ahrens et al., 2013)	✓			✓			✓	✓		✓	✓		✓	✓	✓	✓	
(Oliveira et al., 2013)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
(Nakagawa et al., 2013)	✓	✓	✓		✓	✓		✓	✓	✓	✓			✓	✓	✓	✓

(Jeong et al., 2014)	✓	✓	✓			✓		✓	✓	✓	✓	✓			✓		
(Bianchi et al., 2015)	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
(Garcés et al., 2017)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓
(Kantarci et al., 2014)																✓	
(Sicari et al., 2016)						✓											
(Kiran and Simons, 2016)					✓	✓	✓										

4. Comparative Study

In order to identify the eventual shortcomings in the field of quality attributes and models of embedded systems, we present in this section a comparative study of the above studies. This comparative study can be served as the first stone to build a quality model for embedded systems. The concerned studies can be compared through the next criteria:

- **Attribute relationships:** as previously mentioned, quality attributes can be correlated to one another with strong or weak relationships. Besides, they are often listed with no specified relation or structure. We believe that the more sophisticated embedded systems get, the more these attributes' relations and structures take over.
- **Attribute foundation:** in order to define quality for embedded systems, it is necessary to make a set of important quality attributes that reflect the specifications of these systems. In this case, it is inevitable to make at least one of both choices: the proposition of new whole attributes for embedded systems or the adoption of other systems attributes with the possible modification of some attributes. The new proposed attributes can be viewed as the specific characteristics of the embedded systems. On the other hand, the adopted attributes are the common characteristics of software quality.
- **Attribute decomposition:** in most related works, the proposed attributes are always described with additional sub-attributes. This principle is very common in the context of software quality engineering, where attributes are decomposed into

sub-attributes until measurable entities are obtained. This decomposition helps to understand the vague concept of quality through smaller entities. Furthermore, these smaller entities play an important role in other divergent phases such as quality assessment or prediction.

- **Product/Process Quality:** when we talk about quality, we distinguish two different cases: product quality and process quality. Process quality is related to the way of product development (Kandt, 2005). The importance of this type of quality is due to the measurability of the process software which improves the developed product quality (Pressman, 2010). For product quality, we have to understand deeply the user needs, translate them into clear product attributes, and ensure that these attributes are implemented later in the product (Wagner, 2013).
- **Scope and application area:** due to the large acceptance of embedded systems in our daily life, quality attributes can be generic or related to the specific application domain of embedded systems. The general attributes in the context of embedded systems express how similar these systems are to other systems, since they share some components with them, such as the software side, for example. In contrast, the specific attributes express the uniqueness of the embedded systems compared to the rest of the systems. We do believe that generic attributes are so promising with their global view against the specific attributes, which can only cover a restricted view.
- **Standard-based attributes:** as we have seen so far, standard attributes are more adequate due to their applicability to every product no matter what type of these products or how complex they are. Furthermore, the efficiency of these standard attributes relies on their ability to present a unified and homogenous view of vague concepts such as quality. Therefore, these standard attributes help exchange results and the comparability between several products.
- **The purpose of the quality model:** quality models can be used for many purposes like definition, evaluation, and prediction of the quality of software products. Understanding the goal of using the proposed quality models in embedded systems can help us to improve their exploitation. Because of the specific development cycle of embedded systems, we think that using quality models to predict the quality of embedded systems in an early stage of development is more useful than using them to just define or assess the quality.

Table 3.2 summarizes the result of the comparative study. Despite the existence of several works that focus on the most important quality attributes and quality models in the context of embedded systems, we think that this field needs more investigations to fix its shortcomings. The main shortcoming identified by our comparative study consisted in:

- Unclear decomposition of the complex term quality due to the absence of a specific criterion that distinguishes between characteristics and sub-characteristics. Therefore, to reveal the ambiguity, it is important to determine which attributes are the main characteristics and which are the sub-characteristics. This key step will guarantee a sort of consistency between the attributes.
- The absence of a well-defined structure and relationship between the set of characteristics. This can be achieved through a quality model. We think that enhancing an existing quality model, especially the standard ones, with the specific characteristics of embedded systems is an attractive way.
- The most proposed quality models in this field are used only to specify or assess the quality. No work addressed the prediction of the quality of such systems.
- When we look back to the comparative study, it is interesting to say that we did not find any new proposed attributes, which reflect the specifications of embedded systems despite the absence of many such as adaptivity, autonomy, criticality, and complexity that sometimes can be considered as relevant to embedded systems.

Compared to similar studies that propose a literature review of quality attributes and models for embedded systems with our work, we presented various novelties. Besides including new papers in our work, the comparative study we proposed is based on several original criteria. We think that comparing existing works according to these criteria allows us to conclude the main challenges and shortcomings in this domain. Consequently, the obtained results in this work depict the main deficiency in this field, especially the lack of a well-defined quality model for defining, assessing, or predicting the quality of embedded systems. It is important to note that the following table represents the results presented in our first contribution, a survey on quality attributes and quality models in the context of embedded systems (Tamrabet et. al, 2018).

Table 3.2: Comparative study (Tamrabet et. al, 2018).

	Attribute Relationships	**Attribute Foundation**	**Attribute Decomposition**	**Product / Process Quality**	**Scope and application area**	**Standard Based Attributes**	**Purpose of quality model**
(Sherman, 2008)	Lack of relationship	Trade studies analyses	/	Both	Generic	/	/
(Wijnstra, 2001)	Not clear	/	Not clear	Product quality	Medical Products	/	/
(Choi et al., 2008)	Well defined	ISO 9126 Quality Model	Hierarchal	Product Quality	Digital Media Electronics	ISO/IEC 9126	Assessment
(Jeong & Kim, 2012)	Weak relationship	Based on DeLone and McLean Success Model, ISO/IEC 9126	Hierarchal	Product Quality	Lightweight Component Development	/	Definition
(Ahrens et al., 2013)	No clear relationship	Inspired from McCall, Boehm, ISO/IEC 9126	Hierarchal	Product Quality	Automotive Embedded Domain	/	Assessment
(Guessi et al., 2012)	Lack of relationship	Systematic review	/	Both	Generic	/	/
(Oliveira et al., 2013)	Lack of relationship	Systematic review	/	Both	Generic	/	/
(Carvalho & Meira, 2009)	No clear relationship	Based on SQuaRE project	Hierarchal	Product Quality	Component-Based Software Development	The SQuaRE project	Assessment
(Nakagawa et al., 2013)	Lack of relationship	Systematic review	/	Both	Software Architecture	/	/
(Bianchi et al., 2015)	Lack of relationship	Systematic Literature Review	/	Both	Generic	ISO/IEC 25010	/
(Jeong et al., 2014)	No clear relationship	Based on DeLone and McLean Success Model, ANP	/	Product Quality	Secure Embedded Systems with Sensor Network	/	Definition
(Garcés et al., 2017)	Well defined relationship	Systematic Mapping	Hierarchal	Both	Ambient Assisted Living	ISO/IEC 25010	Definition
(Kantarci et al., 2014)	Lack of relationship	/	/	Product Quality	Cloud-centric Internet of Things	/	/

(Sicari et al., 2016)	Lack of relationship	/	/	Product Quality	Internet of Things	/	/
(Kiran and Simons, 2016)	Lack of relationship	/	/	Product Quality	Cloud ecosystems	/	/

5. Conclusion

Embedded systems are systems consisting of hardware and software. They are becoming complex more than ever before, and as they grow, the traditional development process becomes insufficient. Today, the success of embedded systems is not measured by how they are managed to serve, but rather by the set of the non-functional requirements expressed through the concept of quality. Furthermore, quality is a confusing concept that cannot be comprehended without the aid of a suitable model. This concept should be modeled by breaking it down into a set of characteristics and sub-characteristics. As a result, managing this set of characteristics enables effective management of the overall system quality. In this chapter, we presented a survey on the most important works in the context of embedded software quality attributes and quality models. A comparative study is also provided to demonstrate the strengths and weaknesses of each work. Eventually, we believe that the identification of embedded software quality attributes and the adoption of quality models are both necessary to improve the quality of such systems. Despite the importance of the presented studies in this chapter, some tracks in this field need more investigation. Especially, proposing a generic quality model for embedded software using existing standards is an attractive goal. In the next chapter, we present our second contribution, a quality model for the embedded software.

Chapter 04

ESQuMo an Embedded Software Quality Model

1. Introduction

Quality has grown increasingly crucial in a variety of fields, particularly software development. As the application areas of the software are diversified, we should expect serious consequences if the quality was not there. The cost of the missing quality depends on which area the software is in use. Embedded systems, similar to other areas are affected by the absence of quality, this latter that is usually expressed with users' satisfaction. The best example of the user's satisfaction is the choice between computers and phones in the last ten years. We cannot easily choose which is the best choice for the user, except by looking at his satisfaction. In addition, the same users who preferred to use computers once, now their concerns became larger in using phones. This is a very natural phenomenon, and it can be explained by the quality that phones do provide in the current era as they have not provided before.

To fully comprehend the cost of the missing quality, it is necessary to first define quality. Therefore, the adoption of quality models is beneficial, as they are the foundation of quality modeling. Many works are interested in studying quality across many fields. However, quality studies in the context of embedded systems suffer from many shortcomings as discussed in the previous chapter.

It is important to note that the available quality engineering tools and methods are generally applicable to most known types of systems, even if they may require some adjustments in the development phase (Suryn, 2013). Because of that, several contributions in software quality assurance exploit this idea of identifying the different characteristics of the target product and inserting them into the right quality model. In the footsteps of these works, and in pursuance of building a suitable quality model for the embedded software, it is important to determine first what exactly should be added to a model in order to support the specifications of embedded software.

Through this chapter, we present our second contribution, which is mainly a quality model for embedded software. The remainder of this chapter is organized as follows. In section 2, we introduce ESQuMo, an embedded software quality model. Section 3 provides a comparative study of the proposed quality model with other quality assurance studies in the context of embedded systems. Section 4 presents the

application of the ESQuMo quality model to medical imaging embedded software. Conclusions are given in section 5

2. Quality Model for Embedded Software

Quality models are the first stage in quality assurance. They assist in the identification and definition of quality requirements. Quality models describe the concept of quality by decomposing it into a series of characteristics and sub-characteristics, culminating in an atomic characteristic that can be evaluated directly (Boehm et al., 1978; McCall, 1977). Moreover, quality models help to understand how exactly these characteristics contribute and influence the whole quality. Formally, the quality of the embedded software Q is defined by a set of characteristics (C_i):

$$Q = \{C_i / \ i \in N\}$$

Similarly, each characteristic $C_{i=1..n}$ is specified as a set of sub-characteristics that affect it.

$$C_i = \{SC_j / \ j \in N\}$$

In the context of embedded software, many works concerned with quality models and quality attributes have emerged (Choi et al., 2008; Carvalho & Meira, 2009; Guessi et al., 2012; Jeong & Kim, 2012; Ahrens et al., 2013; Oliveira et al., 2013; Nakagawa et al., 2013; Jeong et al., 2014; Bianchi et al., 2015; Kiran and Simons, 2016; Garcés et al., 2017). We presented a list of notable studies that addressed the quality models and quality attributes of embedded systems in our first contribution (Tamrabet et al., 2018). The findings of this paper might be summarized in terms of the lack of particular quality models for such systems. Besides, a number of works introduced the concept of quality through a set of quality characteristics, each according to their view of this concept (Rahman et al., 2018; Liebel et al., 2018; Vegendla et al., 2018; Akdur et al., 2018; Mohsin and Janjua, 2018; Kaur and Singh, 2019; Cadavid et al., 2020; Sales and Becker, 2021). Furthermore, several works share the same idea in the construction of an embedded system quality model. Begin by investigating the characteristics that reflect the unique properties of embedded systems. Then comes the process of identifying the most important characteristics. Afterward, subdivide the identified characteristics into more manageable sub-characteristics. Finally, these characteristics

are incorporated into an existing quality model to create a more suited one for embedded systems.

In this section, we will conduct our research to: i) Investigate quality models and quality attributes in the context of embedded systems. ii) Identify the most significant characteristics of the embedded software. iii) Propose an embedded software quality model based on the well-established ISO/IEC 25010 quality model.

2.1.Investigating quality models and quality attributes of embedded software

Several works in the context of embedded software focused on quality models and quality attributes. On the one hand, many works presented a list of quality characteristics based on their definition of quality as well as the application area of the targeted products (Guessi et al., 2012; Oliveira et al., 2013; Nakagawa et al., 2013; Bianchi et al., 2015; Kiran and Simons, 2016; Rahman et al,. 2018; Liebel et al,. 2018; Vegendla et al,. 2018; Akdur et al,. 2018; Mohsin and Janjua, 2018; Kaur and Singh, 2019; Cadavid et al,. 2020; Sales and Becker, 2021). On the other hand, a lot of works extended existing models to make them more suitable for embedded systems (Choi et al., 2008; Carvalho & Meira, 2009; Jeong & Kim, 2012; Ahrens et al., 2013; Jeong et al., 2014; Garcés et al., 2017).

In table 4.1, we summarize the characteristics identified in the works concerned with quality models and quality attributes for embedded systems. We have sorted all the characteristics to determine the ones considered relevant. Furthermore, since we are concerned with embedded software, we have omitted hardware-related characteristics (power consumption, physical size, weight, etc.). As a result, seventeen distinct characteristics were extracted. In addition, we have proposed *Complexity* and *Dependability*. The choice of these two characteristics is justified by their existence in the literature of embedded systems (Wagner, 2013; Suryn, 2013; Avižienis et al, 2004; Banker et al, 1989), as well as their absence in the works concerned with quality models and quality attributes while they do represent significant characteristics of embedded software. On the one hand, complexity is one of the most important properties of embedded software (Banker et al., 1989). On the other hand, dependability is considered an essential property that reflects the criticality of the embedded software (Avižienis et al, 2004).

Table 4.1: Results of the investigation on quality models and quality attributes (Tamrabet et al., 2022a).

	Reliability	Efficiency	Usability	Maintainability	Portability	Performance	Security	Safety	Availability	Fault tolerance	Functionality	Interoperability	Adaptability	Flexibility	Recoverability	Reusability	Testability	Complexity	Dependability
(Choi et al., 2008)	✓	✓	✓	✓	✓		✓			✓	✓		✓		✓	✓	✓		
(Carvalho & Meira, 2009)	✓	✓	✓	✓	✓		✓	✓		✓	✓				✓				
(Guessi et al., 2012)	✓			✓		✓		✓		✓		✓	✓						✓
(Jeong & Kim, 2012)	✓	✓	✓	✓			✓		✓	✓		✓	✓		✓				
(Ahrens et al., 2013)	✓	✓			✓			✓	✓		✓	✓	✓			✓	✓		
(Oliveira et al., 2013)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓		
(Nakagawa et al., 2013)	✓	✓	✓	✓		✓	✓	✓	✓	✓		✓	✓	✓			✓		
(Jeong et al., 2014)	✓	✓	✓	✓			✓		✓	✓		✓	✓		✓				
(Bianchi et al., 2015)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
(Kiran and Simons, 2016)						✓	✓				✓								

(Garcés et al., 2017)	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓		✓
(Rahman et al,. 2018)			✓			✓	✓		✓		✓	✓	✓						
(Liebel et al,. 2018)	✓	✓		✓				✓	✓							✓			
(Vegendla et al,. 2018)	✓		✓	✓		✓	✓	✓									✓		
(Akdur et al,. 2018)	✓			✓	✓											✓			
(Mohsin and Janjua, 2018)							✓					✓	✓						✓
(Kaur and Singh, 2019)				✓			✓				✓		✓	✓		✓	✓	✓	
(Cadavid et al,. 2020)	✓	✓	✓	✓	✓	✓	✓				✓								
(Sales and Becker, 2021)	✓		✓	✓	✓	✓		✓			✓								

2.2.Identifying the relevant characteristics of embedded software

The aim of this process is the identification of the characteristics relevant to embedded software. In addition to the establishment of a suitable quality model for such software based on the well-known quality model ISO/IEC 25010. After the extraction of the significant characteristics of the embedded software, we can notice that almost all the quality characteristics defined in the ISO/IEC 25010 product quality model are included in our extraction (*Reliability*, *Usability*, *Maintainability*, *Portability*, *Performance*, *Security*, and *Functionality*). This is considered a natural phenomenon since the global software category encompasses the range of embedded software. Furthermore, characteristics like: *Adaptability*, *Availability*, *Fault tolerance*, *Recoverability*,

Interoperability, *Reusability*, and *Testability* are included as well in the extraction process as they are considered sub-characteristics in the ISO/IEC 25010 product quality model.

For *flexibility,* we can proclaim that it is as same as *adaptability* according to the IEEE Standard Glossary of Software Engineering Terminology, which considers them synonyms (IEEE, 1990). Moreover, we are interested only in the *Product Quality* model since we are concerned with the early stages of software development. Thus, *Efficiency* and *Safety* are out of the scope of this work due to their presence in another quality model, the *Quality in use* model defined in ISO/IEC 25010.

Even though *Complexity* and *Dependability* are crucial characteristics in the context of embedded systems (Avižienis et al, 2004; Banker et al, 1989), none of the previous works identified them as relevant characteristics. In addition, they are completely absent from the actual quality models. Therefore, we believe that these characteristics have an impact on the quality level of embedded software. Hence, it is mandatory to pay more attention to these characteristics. In the next section, we will study the complexity and the dependability as relevant characteristics for the embedded software; together with their relationship to the ISO/IEC 25010 product quality model.

2.2.1. Complexity

Complexity is an important determinant of software quality. It influences the comprehension, the modification, the usage, and the resources used by the software (Banker et al, 1989; Chidamber and Kemerer, 1994; Lee et al., 2016). For this reason, more attention should be given to this characteristic. Therefore, in order to study complexity as a central characteristic of software quality, we should break it down into manageable parts. In the specialized literature, we can find that complexity is treated on McCall's quality model under the name of simplicity (McCall, 1977). Besides, it is considered a multidimensional concept according to Wake and Henry (1988). Furthermore, the complexity of software depends on its magnitude, the complexity of its structure, and the complexity of its data flows (Basili and Hutchens, 1983). Also, the complexity of software is related to the degree of its modularity (Bowen, 1978). From this point of view, we suggest that complexity should be studied through the following dimensions: *structural complexity*, *operational complexity*, and *resources complexity*.

Structural complexity

Structural complexity is closely related to maintainability. A poor software structure results in more errors, takes longer, and costs more to maintain (Banker et al, 1989). Contrariwise, a good software structure is easy and less expensive to maintain. Thus, to facilitate the study of structural complexity, we have to be aware of the factors behind it.

First, the size of the software is considered with no doubt as essential factor for the software structure. Big sized software is usually more complex, and even a tiny change in the software will cause an overall check to assure that the software works correctly (Banker et al, 1989). So, the software should be flexible for such changes.

Moreover, to avoid unnecessary checks every time changes happen, developers must decompose the software into smaller sub-software. This decomposition is a good way to improve the maintenance costs. It allows for reducing the maintenance efforts through the maintenance of smaller sub-software instead of the whole software (Banker et al, 1989).

Yet the decomposition of the software into smaller modules improves the maintainability. This decomposition, however, may lead to negative consequences if the software is decomposed into too many modules. In this case, even the smallest changes force the developers to deal with an immense number of modules. In contrast, if decomposed into a few modules, each module will be big-sized and difficult to comprehend and to be modified (Banker et al, 1989). The decomposition of the software should be done in a way to facilitate the maintenance and make it easy to locate the errors at the module level.

Operational Complexity

Operational complexity is more related to the usability of the software. It describes how well and with what satisfaction a user can operate the software. In this context, the software is declared of high complexity if it is hard to operate and control. In addition, the software can be declared complex if it prevents the users to learn how to use it in a specified context of use. Besides, if the software does not offer the required functionality, it crashes frequently, or it reacts slowly, then it is less usable and considered complex (Wagner, 2013).

Resources Complexity

The last dimension of complexity is expressed through the resources used by the software. Usually, software of high complexity consumes more amounts of resources compared to software of low complexity. The increased complexity means that the software takes a longer time to process and respond while performing its required functions. In addition, the increased complexity is a sign that the amounts of the physical resources while performing the required functions are bigger in most cases (Wagner, 2013).

To summarize, *Complexity* is expressed through *Maintainability, Usability, and Performance efficiency*. It is important to notice that all these characteristics are present in the ISO/IEC 25010 product quality model (ISO, 2011a).

$$Q_{complexity} = \{Maintainability\ , Usability\ , Performance\ efficiency\}$$

2.2.2. Dependability

Dependability is the characteristic related to specific software systems with very high and very strong requirements, known as dependable software systems or critical software systems (Wagner, 2013). A wide range of embedded software is also dependable software like systems controlling airbags in cars or business software performing bank transitions (Wagner, 2013).

According to Barbacci et al (1995), dependability is the property of a computer system to provide its services with confidence. Moreover, Laprie (1992) states that dependability is that property of a computer system such that reliance can justifiably be placed on the service it delivers and it comprises the next attributes: *availability*, *reliability*, *safety*, and *security*. Furthermore, Avižienis et al (2001) affirm that dependability is a system property that is defined by the ability to deliver service that can justifiably be trusted which integrates attributes such as *reliability*, *availability*, *safety*, *security*, and *maintainability*.

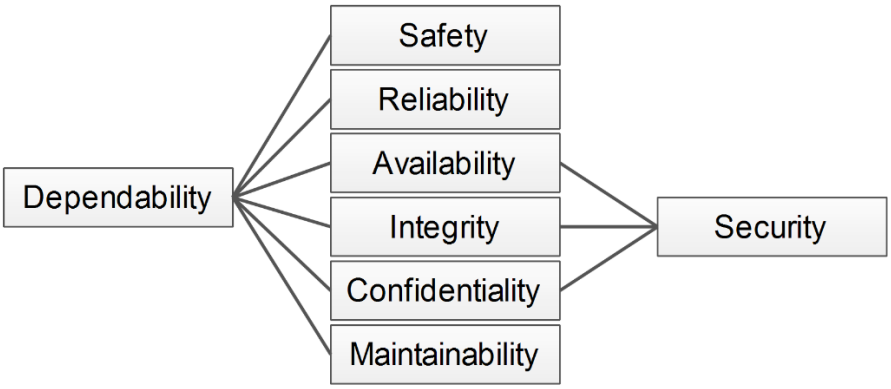

Figure 4.1: The Dependability classification (Avižienis et al, 2004).

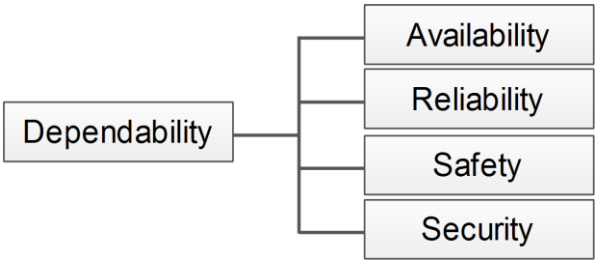

Figure 4.2: The Dependability attributes (Laprie, 1992).

As we can see, the dependability attributes used by both authors Laprie and Avižienis (Laprie, 1992; Avižienis et al, 2004) are very similar except for one attribute: *Maintainability*. This is due to the nature of maintainability itself, it is a general topic for software of any kind and not restricted to dependable software systems (Wagner, 2013).

As depicted in figure 4.2, the dependability attributes are all present in the ISO/IEC 25010 quality model (ISO, 2011a). Few exceptions can be noticed while mapping these factors to the ISO/IEC 25010 quality model. For instance, *Reliability* is considered among the main factors. *Availability* is a part of *Reliability* and not a main factor in itself. In addition, we notice that *Security* represents an important part of dependability; this is due to the nature of the dependable software systems, which should be of high security (Wagner, 2013). Lastly, *Safety* is out of the scope of this work as it represents a part of *freedom from risk* in the ISO/IEC 25010 quality in use model (ISO, 2011a). In our study, we will focus on the ISO/IEC 25010 product quality model (ISO, 2011a) since we are interested in the early stages of software specification.

To summarize, *Dependability* is expressed through *Reliability and Security*. Note that these characteristics are present in the ISO/IEC 25010 product quality model.

$$Q_{dependability} = \{Security\ , Reliability\}$$

As we have already seen, the different taxonomies of the quality characteristics are due to the multiplicity of fields in which embedded systems are used. We do believe that there are no generic characteristics that work for all kinds of embedded systems. Besides, not all of the defined quality characteristics in the ISO/IEC 25010 product quality model are suitable for embedded software. In the next section, a detailed discussion about characteristics that did not appear in our study either in *Complexity* or *Dependability*.

2.2.3. Discussion

In this section, we will discuss some of the characteristics that were not mentioned previously in our study but are still a part of the ISO/IEC 25010 product quality model. The discussion concerns: *Portability*, *Compatibility*, and *Functionality suitability*.

Portability

In usual programming, it is sufficient to compile the software to obtain an executable version of the software. The compilation is an automatic translation of a description written in one high-level language into another low-level language. So the software can take many forms and in many languages. However, the processor on which it is executed generally only understands a language based on sequences of 0 and 1, we speak of machine language. And as technology evolves, there are several processors that do not understand the same machine language. That is, "a program is only executable for one given processor".

There are also what are called virtual machines. A virtual machine is a software layer added to processors and includes the same language regardless of the processor on which it is installed. The most famous example is the Java language, which runs on a Java virtual machine. In this case, an executable program for the virtual machine is independent of the processor. The advantage is that the same program can run regardless of the processor. The downside is that these machines are usually very slow.

Moreover, embedded systems are designed for specific tasks and they are subject to a set of strong constraints (limited memory, small footprint, etc.), in this case, the traditional compilation is no longer efficient. For this, software targeting is proposed as a solution. Software targeting consists of producing an executable description of a software application for a given hardware architecture, that is, it adapts the software to a target and specific hardware architecture.

Additionally, some embedded systems have an operating system and others do not because all of their logic can be implemented in one program. Usually, the programs of an embedded system are stored in ROM, which is a non-volatile memory and permanently stores instructions of the programs in an unchangeable way.

From this point of view, we think that *Portability* is not important enough for embedded systems due to their technical properties and because it is defined as the ability of the software to be transferred from one hardware to another.

Compatibility

Note that an embedded system is present in various application areas such as computing systems, control systems, signal processing, telecommunications, and network. Several features characterize this system, but perhaps the most important feature is being designed for a specific task and not a general-purpose system. An embedded system does not have standard inputs/outputs, but it is equipped with sensors/actuators. As a result, an embedded system is simple and limited in terms of resources (memory, microcontroller, power ...etc.).

In addition, an embedded system is qualified to react to the environment. As already shown in figure 1.2 (Serpanos et al., 2011), the environment is moderated by a set of sensors; then the captured information goes to processing by the embedded system's core; then, the actuators execute the result in the environment.

According to the scheme in figure 1.2 (Serpanos et al., 2011), there is no possible interaction with the embedded system except through its sensors/actuators. And since the sensors/actuators represent the external view of the embedded system (i.e. the specific task for which the system is designed) the possibility that the embedded system interacts with another third party for purposes other than its task is totally neglected.

Therefore, the definition of *Compatibility* for embedded systems is useless and can be overlooked.

Functional suitability

The term embedded software is used to designate the software part of an embedded system. Embedded software is quite simply the set of instructions sequences called programs, which can be interpreted by the embedded system itself. The embedded software therefore determines the adequate task of the system, guarantees the correctness of its execution, and covers all possible scenarios of its operation.

Based on this definition, we find that embedded software has the same definition as ordinary software. This is quite normal because embedded software shares the same functional requirements with ordinary software, so the only difference is on the level of non-functional requirements. Consequently, *Functional suitability* represents the common part of different software and not the specifics of embedded software.

2.3.ESQuMo: an Embedded Software Quality Model

In embedded systems, quality characteristics can never be achieved in a completely separated way from the other characteristics. The achievement of some will affect the achievement of others. The traditional way to ensure the reliability of any type of system is through hardware redundancy, for example, two engines on an airplane instead of one. Nevertheless, this redundancy costs always twice in all scenarios (Koopman, 2007). From this point of view, we do believe that quality needs more structure in quality models than just characteristics and sub-characteristics.

Based on the findings of our study, a new quality model is created. This model is called ESQuMo shorthand for **E**mbedded **S**oftware **Qu**ality **Mo**del. As shown in figure 4.3, the model provides the definition of quality through three basic characteristics. The first is *Complexity*, which is an important indicator of quality for the embedded software, especially as they are developed to work on platforms with limited resources. The second is *Dependability*, which in turn represents the criticality of the embedded software, given that these systems are being developed to operate in environments that affect their outcome. The third characteristic is *Functionality*, which describes the similarities between the embedded software and the other regular software. It should be

noted that this ESQuMo model is a definition purpose quality model. The quality of the overall embedded software could be expressed through the next formula:

$$Q = Q_{complexity} \cup Q_{dependability} \cup Q_{functionality}$$

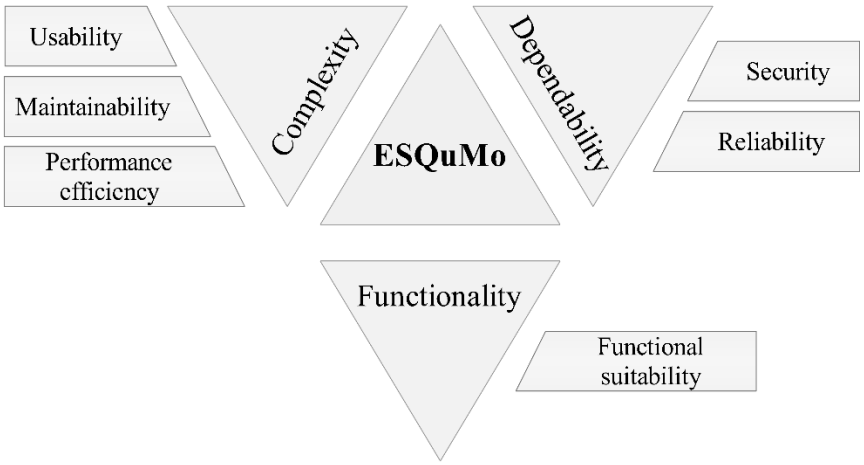

Figure 4.3: ESQuMo: an Embedded Software Quality Model (Tamrabet et. al, 2022a).

As previously indicated, a quality model is a set of interrelated characteristics. Naturally, a sub-characteristic influences the characteristics related to it. However, the representation of all the possible relations between the characteristics and the sub-characteristics generates a complex and difficult model to use. Therefore, we decide to present only the important relationships in order to obtain a comprehensible quality model. It is important to note that the modification applied to the characteristics level of the ISO/IEC 25010 quality model does not affect the spirit of the latter. Rather, a more focus on the characteristics that reflect the specificities of embedded systems.

The addition provided by our ESQuMo model in contrast to the aforementioned models can be summarized in the following points:

Our model is based on the famous ISO/IEC 25010 product quality model. The choice of the ISO/IEC 25010 quality model is not arbitrary, but rather thoughtful due to multiple reasons. First, the model is based on a meta-model, which in turn facilitates the definition of what quality is, and gives a more comprehensive interpretation and a clear view of quality. Second, as a standard model, it grants the possibility of comparability between different software products. This feature is as advantageous as it sounds especially with the intangible software products in the context of quality.

Lastly, since the ISO/IEC 25010 quality model belongs to the quality model division, which has other complementary divisions at the SQuaRE project. Consequently, the orientation of our proposed quality model towards other divisions is quite supported and possible.

In addition to the ability of our model to highlight the distinctive characteristics of the embedded software, it also highlights the general and common characteristics of this embedded software with other software. This is of prime importance since the embedded software represents a special range of the overall software category.

3. Comparative study

As previously stated in the state of the art, numerous works are concerned with quality in the context of embedded systems. After a thorough examination of such works, we can distinguish three distinct paths in this topic. Many of these works are restricted to proposing quality characteristics and do not provide the structure of quality models. Some works address quality by recommending a specific quality model. Only a few works propose quality models based on standard quality models. In this section, we will compare our proposed quality model to a sample of each of the aforementioned categories. The results of the comparative study are resumed in Table 4.2.

Table 4.2: Results of the comparative study (Tamrabet et. al, 2022a).

	ESQuMo	**SCQM (Choi et al., 2008)**	**QMSES (Jeong et al., 2014)**	**(Akdur et al., 2018)**
Model purpose	Definition purpose quality model	Assessment purpose quality model	Definition purpose quality model	/
Standard used	ISO/IEC 25010	ISO/IEC 9126	/	/
Meta-model	Meta-model Based (ISO/IEC 25010 meta-model)	Lack of an explicit meta-model	Lack of an explicit meta-model	/
Characteristics	Remove 02 characteristics Add 02 characteristics	Add 02 characteristics	Delone & Mclean characteristics Add 02 characteristics	A set of individual characteristics and sub- characteristics
Sub-characteristics	Use the same sub-characteristics of the standard	Remove/Keep/ and Propose new sub-characteristics	Propose new sub-characteristics	

Characteristics/Sub-characteristics relationship	Very simple connections and more comprehensive	More nested and complicated connections	Incomprehensible and unclear connections	Complete absence of connections
Application field	Generic embedded systems	Component based embedded systems	Secure embedded systems with sensor network	The embedded software industry

Foremost, SCQM (Samsung Component Quality evaluation Model) is based on the ISO/IEC 9126 quality model, which is the predecessor of the ISO/IEC 25010. In addition, SCQM does not have an explicit meta-model. Moreover, we think that using the ISO/IEC 9126 quality model as a base to build SCQM causes several shortcomings in its structure. In fact, using "modularity" which is relatively a simple concept as a characteristic, while using the "security" as a sub-characteristic is questionable (Choi et al., 2008).

Also, SCQM proposed new sub-characteristics. On the one hand, this step generates new relationships that make the connection between characteristics and sub-characteristics more nested and complicated. On the other hand, proposing new different sub-characteristics with new different metrics provides with no doubt different results. The main advantage of using standard models is the comparability aspect through the standard metrics while evaluating other software products.

For QMSES (Quality Model for Secure Embedded Systems) the author proposed a quality model for secure embedded systems with sensor networks, which uses an analytic network process. The proposition of this model is based on DeLone and McLean's model (Jeong et al., 2014) which presents influential characteristics and their relationships. Additional characteristics and sub-characteristics have been proposed to support the specifications of secure embedded systems with sensor networks.

The main shortcoming that this model suffers from is the difficulty of understanding the decomposition of the concept of quality into a set of characteristics due to the absence of a clear criterion for the decomposition, as well as the absence of a meta-model that explains the complex interrelationships of these characteristics. Furthermore, using a specific quality model as a base for this work, instead of using standard quality models, causes a real usability challenge.

For the last category, all we can say is that it considers the concept of quality as a set of individual characteristics, which it is not. Quality is more than just characteristics

and sub-characteristics. Once again, we emphasize the need for correlations between characteristics because they do represent a large part of the overall quality.

In contrast to the previous works, our proposed quality model ESQuMo presents several additional points, the most important of which are: Based on the standard ISO/IEC 25010 quality model, which in turn provides a meta-model that makes it easier to read and understand what quality is. In addition to the ability to pass from the quality definition provided by the model to other quality purposes (such as quality assessment and quality prediction) through the complementary standards of ISO/IEC 25010. Focuses more on the characteristics that reflect the relevant properties of the embedded software without overlooking the common properties of the ordinary software. Uses the same sub-characteristics of the standard used for comparability purposes. Presents a simple and well-defined structure of the quality concept through fewer and simpler connections between characteristics and sub-characteristics. Finally, it is important to note that our proposed quality model can be applied to various embedded systems in general. Contrary to the previous works, which are restricted by targeting a specific type of these systems. The proposition of this model is due to the lack of a standard quality model that unifies and covers the specific characteristics of the embedded software, as well as the common characteristics of the ordinary software.

For demonstration reasons, we devote the next section to the application of the ESQuMo quality model to medical imaging embedded software by presenting a set of quality measures.

4. Applying the ESQuMo Quality Model to Medical Imaging Embedded Software

Embedded systems can be found in a wide range of fields. For instance, the medical field is littered with many examples such as *Defibrillators*, *Oximeters*, *Fetal monitors*…etc (Dere, 2021). These medical devices made the lives of people related to the medical field much easier as well as their profession more efficient. Medical imaging devices are no less important than the aforementioned devices, as medical imaging is so sensitive, that it requires the use of new imaging technologies and relies more on this kind of system (Lee, et al., 2016; Dere, 2021).

Medical imaging devices are considered the most commonly used embedded systems since they are able to see inside a patient's body without actually having to cut them open. This is extremely important as it allows doctors to identify any abnormalities and it helps to decide the right medical diagnosis as well. However, the real challenge with these types of systems is to ensure that all the provided results are dependable and can be relied on (Dere, 2021). In addition, the images taken in the medical imaging process must be not only accurate but also in real-time. For example, in ultrasound imaging, any delay in the process of displaying images affects the diagnosis process and may completely conceal the real issue to be imaged in the first place (Niederhauser et al., 2005).

Furthermore, medical imaging devices are among the embedded devices that necessitate prior knowledge before performing any imaging process. This is because medical imaging devices require certain procedures during imaging that if not followed would completely disrupt the imaging, such as the patient's continuous movement or carrying any type of metal that would perturb the imaging process and at times jeopardize the patient's safety, particularly when using CT or MRI. Therefore, the choice of the right medical imaging modality is related to several points such as the patient's age, the type of injury, the organ or tissue to be imaged, the cost of imaging in addition to the amount of radiation that is sometimes present in the medical imaging process. In the following, we will present the most popular modalities of medical imaging (Ahmad et. al, 2014).

4.1.Modalities of Medical Imaging

4.1.1. X-ray

An x-ray is an image of a specific area of the body, such as the chest, abdomen, or a specific joint. With an x-ray, highly charged electrons are shouted through a specific area of the body, the image is then formed according to the scatter of these electrons. Thus, high-density objects will appear white because electrons scatter everywhere against them, and low-density objects like air will appear black since there is no scatter at all. As a result, everything in between is going to be some shade of grey. For the radiation of the x-ray, there is some radiation. However, it is very low and no need to worry about it, and in terms of cost, x-rays are among the very cheap tests.

4.1.2. CT scan

A CT scan is a little bit the same technology as an x-ray, in which high-energy electrons are shouted through the body, except for taking cross-sectional images in the case of a CT scan. The images are taken from a 360-degree view around the patient and this creates a slice-by-slice, a layer-by-layer map of the patient. This technology allows for visualizing bones, soft tissues, and vessels all at once. CT scan is a very powerful imaging technique. However, it is not without issues. It is a much more radiation-intensive process since it is a series of cross-sectional x-rays. compared to an x-ray, a CT scan is probably three or four times as intense in terms of radiation exposure, and because it is more sophisticated, it costs a little bit more.

4.1.3. Ultrasound

The ultrasound is a very useful imaging modality, which contains much interesting physics behind it. The main idea is to take high-frequency sound waves and bounce them all over the tissues, and the way that they echo back to the ultrasound probe is what produces the images used for the density assessment of the tissues. It is a great, easy and effective imaging modality for visualizing body cavities, fluids, tissues, and blood vessels. As sound waves are used, no radiation exposure adds to the usability of an ultrasound. In addition, it is a very cheap test compared to a CT scan or MRI, similar to probably the cost of an x-ray but there is no radiation cost so ultrasound is much more frequently used.

4.1.4. MRI

The MRI is also a common imaging modality but it is a little bit different from the other imaging modalities. Instead of using radiation or sound waves, the MRI uses magnetic fields. Two perpendicular magnetic fields that are acting in opposite directions to each other are used to manipulate hydrogen atoms that are spinning in the body. These atoms release signals that are captured, and then used to determine their location in the body, and display it on a gradient of gray colors indicating the strength of the signal, as well as the different tissues of the body (the brain, the spinal cord, the liver, the pancreas, ... etc.).

Typically, an MRI is used to visualize the tissues of the body in order to detect tissue irregularities or tumors, and because it uses magnets, there is obviously no radiation exposure which is greatly beneficial, especially in the pediatric population. Rather than exposing children to CT scans where they are going to get a significant dose of radiation, the images could be friendly obtained using an MRI. Unfortunately, this magnificent imaging modality comes with a cost, it is definitely the most expensive of the four basic imaging modalities.

4.2.Quality Measurements of Medical Imaging Embedded Software

Before proceeding to provide quality measurements of the medical imaging embedded software, it is important to note that the SQuaRE provides a general reference model as shown in figure 4.4, to facilitate the navigation through the different complementary standards (ISO, 2005).

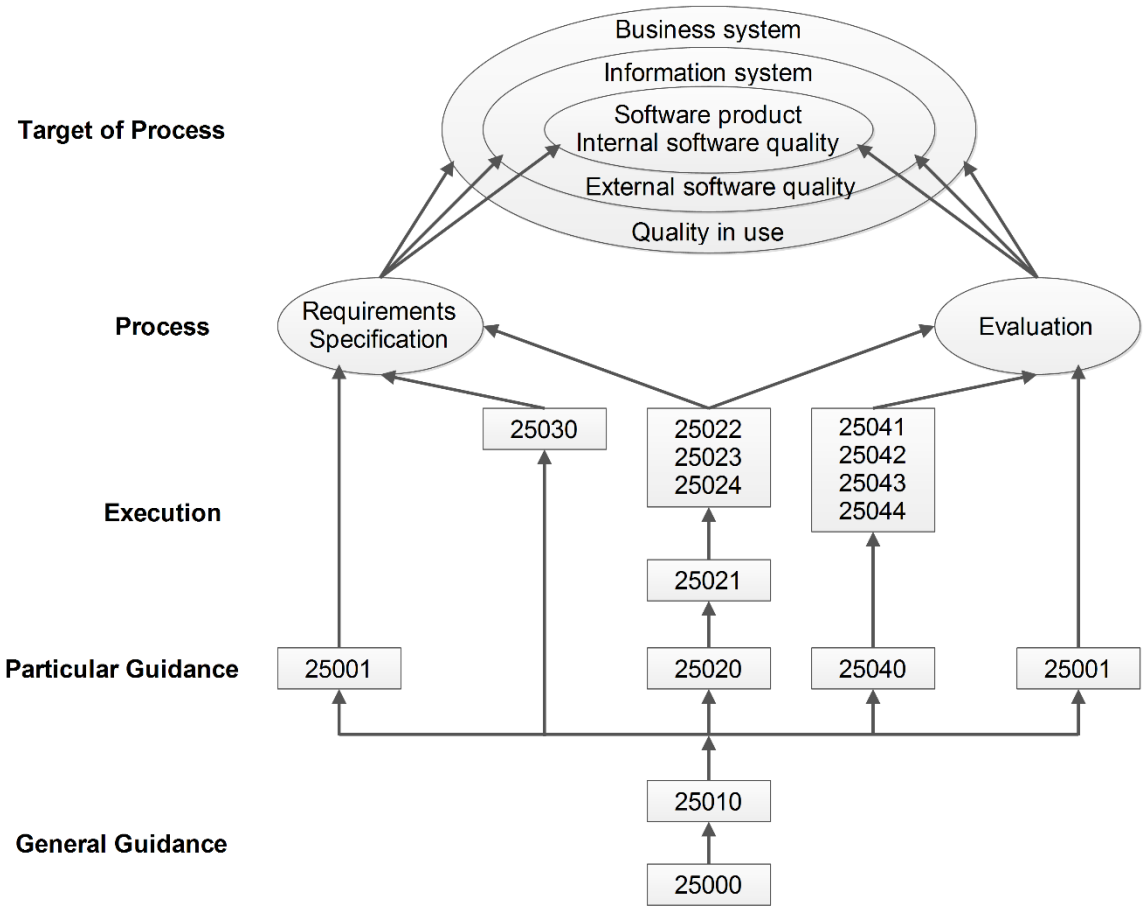

Figure 4.4: SQuaRE general reference model (ISO, 2005).

Through the SQuaRE general reference model (ISO, 2005), it seems necessary to review the ISO/IEC 25020 Measurement reference model and guide straightaway after obtaining the definition of quality through the standard quality model ISO/IEC 25010. Note that the ISO/IEC 25020 includes ISO/IEC 25021, ISO/IEC 25022, ISO/IEC 25023, and ISO/IEC 25024 respectively for Quality measure elements, Measurements of internal quality, Measurements of external quality, and Measurements of quality in use. The orientation of our study is concerned only with the ISO/IEC 25023 Measurements of external quality, where the software is evaluated and measured according to an external point of view. In the next section, quality sub-characteristics and quality measures related to the medical imaging embedded software are presented along with their detailed definitions, as well as their evaluation methods.

The measuring process is very important as it helps to control and supervise the overall software quality. In this process, before proposing a set of quality measurements for each of the identified characteristics in the Embedded Software Quality Model (ESQuMo), it is important to break down these characteristics into measurable sub-characteristics. Moreover, as stressed in the SQuaRE main guide (ISO, 2005); the quality model can be used as a guideline or checklist where the selection of the concerned characteristics is possible. From this point of view, the proposition of quality measurements for all the sub characteristics is not mandatory. In addition, since the Embedded Software Quality Model is based on the ISO/IEC 25010 standard quality model, three distinguished cases are possible for the sub-characteristics:

- Specific sub-characteristics are proposed for medical imaging embedded software, as well as their subsequent measures.
- Some sub-characteristics can be drawn from the ISO/IEC 25010 quality model, extended with new defined measures in order to support the medical imaging embedded software.
- The remaining sub-characteristics of the ISO/IEC 25010 quality model can be overlooked.

4.3.Specific Sub-characteristics of Medical Imaging Embedded Software

4.3.1. Cost of Use

Cost of Use: "*Degree to which a system costs while performing its operations*".

Price has always been an important indicator in any type of product. As we have already mentioned, the modality of medical imaging controls the cost of the imaging process. This sub-characteristic is part of the *Usability* characteristic, where it can be measured qualitatively as follows:

CoU ∈ {Cheap, Economic, Expensive, Extremely Expensive}

4.3.2. Vulnerability of Data

Vulnerability of Data: "*Degree to which a system ensures that data is not influenced by patient factors*".

The medical imaging process may be affected by factors that the patient often unintentionally commits. For example, movement during the imaging process or wearing metallic jewelry may affect the results of imaging and make it vulnerable. This sub-characteristic is part of the *Security* characteristic, and it is measured as follows:

VoD = the probability of patient factors occurrence.

4.4.Extended ISO/IEC 25010 Sub-Characteristics

4.4.1. Time Behaviour

Medical imaging devices are among the real-time embedded systems. Thus, the time aspect is the most susceptible feature for these kinds of devices. A real-time embedded system is defined as a system in which the time at each produced output is significant. Hence, the lag from input time to output time must be sufficiently small. Therefore, in the context of embedded systems, it is not only the outcome of the output that matters but also the time of its occurrence. From this point of view, the *Time Behaviour* sub-characteristic is extended with new measures of Timeliness and Responsiveness.

- Timeliness: "*is the quality of being done or occurring at a favorable or useful time*".

$$T = \begin{cases} 1, & if\ Response\ Time\ <=\ Processing\ Time\ +\ Latency \\ 0, & else \end{cases}$$

  - Latency is the delay incurred in communicating a message.
  - Processing time is the amount of time a system takes to process a given request.

- Response time is the total time it takes from when users make a request until they receive a response.

- Responsiveness: "*is the ability to detect change of system's response over time*".

Responsiveness is often expressed as a Standardized Response Mean (Husted et al., 2000). In order to calculate the Standardized Response Mean, first we have to make sure that we already have two groups of response times, and then we apply the next formula, where:

SRM = (M1 – M2) / SD

- M1 is the mean of the first group.
- M2 is the mean of the second group.
- SD is the standard deviation of the population from which the groups were sampled.

After the SRM index is calculated, we use the Cohen's benchmarks (Husted et al., 2000) for better interpretation where:

- Small change: SRM <= 0.2
- Moderate change: 2 < SRM < 0.8
- Large change: SRM >= 0.8

4.4.2. Resource Utilization

Medical imaging devices are among the most energy-consuming embedded systems, and this is mainly due to their use of GPUs as well as CPUs to meet the requirements of the system along with increasing the level of performance. However, this performance comes with a cost. Since the amount of resources used is enormous, it is natural to have a massive energy consumption. This side effect is originally a result of the extensive use of processors in addition to excessive access to the memory. Hence, it is necessary to determine the level of energy consumption of the embedded software. In order to evaluate this latter, the software energy consumption measure is proposed as an extension of the *Resource utilization* sub-characteristic.

- Software Energy Consumption: "*is the amount of energy consumed by the software, given the amount of resources used from processors and memory access*".

  SEC = EIC × MAC

  - Executed Instruction Count (EIC), which corresponds to the number of executed instructions by the processing unit (Furber, 2000).
  - Memory Access Count (MAC), which is equal to the number of memory accesses (Chatzigeorgiou and Stephanides, 2002).

4.4.3. Maturity

The determinism of medical devices is of prime importance because it represents the system's next behavior. The behavior of a non-deterministic system can never be predicted and is always questionable. From this point of view, we do believe that determinism influences the *Maturity* of the overall system.

- Determinism: "*A deterministic system is a system in which a given initial state or condition will always produce the same results*".

In the following section, we adopt the cyclomatic number as a measure of determinism for medical imaging embedded software:

- CN = E − N + 2P, where :
  E = the number of edges of the graph,
  N = the number of nodes of the graph,
  P = the number of connected components.

The cyclomatic number is a famous software measure used to indicate the number of linearly independent paths through a program (McCabe, 1976). This measure is worth at least "1" since there is always at least one path. Mathematically, a cyclomatic number that is too high indicates that the number of decision points of the system is high, which potentially translates to several behaviors of the very same system. Theoretically, a low cyclomatic number indicates that the system is well deterministic.

4.4.4. Fault Tolerance

Robustness is considered a special case of *Fault tolerance*. Since Fault tolerance corresponds to the ability of the system to operate as intended despite the presence of faults, the use of invalid inputs similarly must be tolerated and continue to work normally. The definition of this measure is provided in the following.

- Robustness: "*The degree to which a system continues to function in the presence of invalid inputs*". where it can be measured as follows:
  R = Number of invalid inputs / Number of stops

4.5.Format of the Proposed Quality Measures

For better conformance during the measuring process, and in order to maintain a unified format with the rest of the quality measurements predefined in the international standard ISO/IEC 25023, we must support the following information (ISO, 2016):

a) ID: identification code of quality measure; each ID consists of the following three parts:
- abbreviated alphabetic code representing the quality characteristics as capital X and sub characteristics as one capital followed by lowercase x;
- serial number of sequential order within quality sub characteristic;
- G (Generic) or S (Specific) expressing potential categories of quality measure; where, generic measures can be used whenever appropriate and specific measures could be used when relevant in a particular situation;

b) Name: quality measure name;
c) Description: the information provided by the quality measure;
d) Measurement function: mathematical formula showing how the quality measure elements are combined to produce the quality measure.

The following table summarizes all of the previously mentioned characteristics, sub-characteristics and measurements in the context of medical imaging embedded software.

Table 4.3: Quality measures of medical imaging embedded software (Tamrabet et. al, 2022b).

ID	Characteristic	Sub-Characteristic	Name	Description	Measurement function
UCu-1-S	Usability	Cost of Use	Cost of Use	Degree to which a system costs while performing its operations	CoU ∈ {Cheap, Economic, Expensive, Extremely Expensive}
SVu-1-S	Security	Vulnerability of Data	Vulnerability of Data	Degree to which a system ensures that data is not influenced by patient factors	VoD = the probability of patient factors occurrence.
PTb-5-S	Performance efficiency	Time Behaviour	Timeliness	the quality of being done or occurring at a favorable or useful time	T=1 if Response Time <= Processing Time + Latency; 0 else
PTb-6-S	Performance efficiency	Time Behaviour	Responsiveness	the ability to detect change of system's response over time	SRM = (M1 – M2) / SD
PRu-5-S	Performance efficiency	Resource utilization	Software Energy Consumption	the amount of energy consumed by the software, given the amount of resources used from processors and memory access	SEC = EIC × MAC
RMa-5-S	Reliability	Maturity	Determinism	A deterministic system is a system in which a given initial state or condition will always produce the same results	CN = E − N + 2P
RFt-4-S	Reliability	Fault tolerance	Robustness	The degree to which a system continues to function in the presence of invalid inputs	R = Number of invalid inputs / Number of stops

5. Conclusions

The designers of embedded systems are continually confronted by the increasing complexity of algorithms, as well as the constraints imposed by the multidisciplinary fields of such systems. Thus, developing embedded systems with an adequate quality level is of prime importance; otherwise, the outcome could produce serious consequences in the case of the missing quality. For this reason, quality assurance has taken place.

One of the most promising ways to improve quality in quality assurance is the adoption of quality models. We note that it is important to use standard models, which cover and unify the different aspects of quality. In the context of embedded software, many works

are concerned with quality attributes and quality models. However, the actual quality models neglect some features of the embedded software like *Complexity* and *Dependability* although they do represent relevant characteristics for embedded software.

This chapter aims to propose a quality model for specifying the quality of embedded software. Based on the well-established standard ISO/IEC 25010 product quality model, ESQuMo an embedded software quality model is established. As a standard-based quality model, ESQuMo gives us the possibility to cover the common software characteristics defined in ISO/IEC 25010 through the *Functionality* characteristic. As well as the relevant characteristics of the embedded software, which are presented through *Complexity* and *Dependability*.

In addition to the proposition of a quality model for the embedded software, we have proposed a set of quality measures for the Medical Imaging Embedded Software as an application of the ESQuMo quality model. Since we cannot manage what we cannot measure, it appears necessary to be able to quantify the proposed characteristics in order to properly manage the quality of the embedded software. This step is of paramount importance because the process of measuring helps to control and supervise software quality. Besides, it provides rapid identification of problems and their solutions. It is important to note that these quality measures are conforming to the international standard ISO/IEC 25023 that works in conjunction with other international standards such as ISO/IEC 25010.

Conclusion and Future Works

1. Conclusion and Perspectives

This thesis falls within the field of software engineering of embedded systems. Specifically, it targets the issue of quality assurance of embedded software. We are mainly interested in developing quality models specific to embedded software because they represent the basis of specification, assessment, and quality prediction.

We began our research with a state of the art covering the domains related to our problem in order to provide valuable contributions. In the first chapter, we presented a general overview covering the field of embedded systems through the various terms, definitions and basic concepts related to embedded systems. As well as the specification and design of such systems. The study of this field has shown us the evolution of embedded systems since their appearance, in addition to the intrinsic characteristics of these systems.

In the second chapter, we presented the second domain attached to our research, namely software quality assurance. Since the term quality in itself is considered an ambiguous term in the field of quality assurance, we have devoted this second chapter to presenting the basic concepts in the context of software quality assurance. This activity is a part of a larger task called software quality management. In addition, this chapter focuses on quality models as a promising tool to achieve software quality assurance. Several quality models are present in the case study in the context of quality assurance; the most important and well-known are mentioned in this chapter, with more emphasis on standard quality models, especially the ISO/IEC 25010 standard quality model.

The study of works devoted to the quality of embedded systems was presented in chapter three. We have classified these studies into two classes: Studies interested in quality attributes and quality characteristics of embedded systems. While other studies are interested in quality models of embedded systems. In the second class, we distinguish the proposal of models specific to the different categories of embedded software, and the extension of existing models in order to support the particularities of embedded software. We also noticed that only a few works were interested in the standardization of quality models. This chapter was crowned by a comparative study, which showed us the limits of the work presented. These limits consist of the unclear decomposition of the complex term of quality. The absence of a well-defined structure and relationship between the set of quality characteristics. Nevertheless, perhaps the

most essential limitation is the lack of a standard quality model that encompasses the common characteristics of the software as well as the distinctive properties of embedded software. These remarks reflect the results of our first contribution; a Survey on Quality Attributes and Quality Models for Embedded Software presented throughout this chapter (Tamrabet et. al, 2018).

In chapter four, we presented our second contribution that is placed in the context of modeling the quality of embedded software. Based on the findings of the previous survey, a model called ESQuMo an Embedded Software Quality Model is established (Tamrabet et. al, 2022a). This model, which consists of an extension of the ISO/IEC 25010 standard model, encompasses at the same time the distinctive characteristics of embedded software as well as the common characteristics of ordinary software. Contrary to works that extend the standard models in the literature, which only add the basic notion of the target products to the standard models, our model also eliminates the non-important characteristics of the embedded software. In addition to establishing this quality model, the third contribution applied the model to a specific type of embedded system, medical imaging embedded systems, by providing a set of external measurements that conform to the ISO/IEC 25023 standard (Tamrabet et. al, 2022b).

2. Future Works

Of course, our contributions remain open for extensions in several directions. First of all, the quality models are classified according to their objectives either for the definition, the assessment, and/or the prediction of the quality. Our contributions mainly focus on the first objective of defining quality for embedded software. In addition, it superficially evaluates the quality of embedded software through the external measures obtained by the application of the proposed model. Indeed, establishing models for assessing and predicting the quality of embedded software is a very interesting subject. Naturally, the quality assessment or prediction will be based on the definition presented by the proposed model. However, assessment or prediction models increase the management and control of embedded systems' quality. An evaluation model presents a framework for the measurement of the overall quality and the interpretation of the results obtained. Whereas a prediction model allows the estimation of the quality of embedded systems even before the implementation phase of these systems.

Another possible direction for the extension of our contributions is the proposal of internal quality measures. As mentioned in the fourth chapter, all of the quality measures proposed are external measures that comply with the ISO/IEC 25023 standard. It is possible to propose internal measures that comply with another complimentary standard the ISO/IEC 25022. This step emphasizes the importance of standardization and the use of standard quality models as mentioned earlier in several places of the thesis. It is possible to take advantage of other complementary standardizations without the need to build elements from zero scratches.

References

Aaronson, S. (2016) 6.045J Automata, Computability, and Complexity: Lecture 3, Spring 2011. Retrieved from http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-045j-automata-computability-and-complexity-spring-2011

Ahrens, D., Frey, A., Pfeiffer, A., & Bertram, T. (2013). Objective evaluation of software architectures in driver assistance systems. Computer Science-Research and Development, 28(1), 23-43.

Airiau, R., Bergé, J. M., Olive, V., & Rouillard, J. (1998). VHDL: langage, modélisation, synthèse. PPUR presses polytechniques.

Akdur, D., Garousi, V., & Demirörs, O. (2018). A survey on modeling and model-driven engineering practices in the embedded software industry. Journal of Systems Architecture, 91, 62-82.

Alonso. F., Fuertes. J. L., Montes C., Navajo. R. J. (1998). A Quality Model: How to Improve the Object Oriented Software Process. IEEE International Conference on Systems, Man, and Cybernetics.

Avižienis A, Laprie JC, Randell, B. (2001). Fundamental Concepts of Dependability. Technical Report 739.

Avizienis, A., Laprie, J. C., Randell, B., & Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. IEEE transactions on dependable and secure computing, 1(1), 11-33.

Banker, R. D., Datar, S. M., & Zweig, D. (1989). Software complexity and maintainability. Age, 11(5.6), 3.

Bansiya, J., & Davis, C. G. (2002). A hierarchical model for object-oriented design quality assessment. IEEE Transactions on software engineering, 28(1), 4-17.

Barbacci, M. et al., 1995, "Quality Attributes, Technical Report CMU/SEI-95-TR-021", Pittsburgh, USA: Software Engineering Institute/Carnegie Mellon University, 56 p.

Barr, M. (2007). Embedded systems glossary. Neutrino Technical Library.

Basili, V. R., and Hutchens, D. H. "An Empirical Study of a Syntactic Complexity Family." IEEE Transactions on Software Engineering, Volume SE-9, Number 6, November 1983, pp. 664-672.

Bedoya, A. E., Perez, Y. M., & Marin, H. A. P. (2016). A review on verification and validation for embedded software. IEEE Latin America Transactions, 14(5), 2339-2347.

Bergé, J. M., Levia, O., & Rouillard, J. (Eds.). (2012). High-level system modeling: specification languages (Vol. 3). Springer Science & Business Media.

Bianchi, T., Santos, D. S., & Felizardo, K. R. (2015, May). Quality attributes of systems-of-systems: a systematic literature review. In Software Engineering for

Systems-of-Systems (SESoS), 2015 IEEE/ACM 3rd International Workshop on (pp. 23-30). IEEE.

Boehm, B. W. (1981). Software engineering economics (Vol. 197). Englewood Cliffs (NJ): Prentice-hall.

Boehm, B. W., Brown, J. R., & Kaspar, H. (1978). Characteristics of software quality.

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

Booch, G. (1990). Object oriented design with applications. Benjamin-Cummings Publishing Co., Inc.

Bowen, J. B. "Are Current Approaches Sufficient for Measuring Software Quality?" Proceedings of the ACM Software Quality Assurance Workshop, November 1978, pp. 148-155.

Cadavid, H., Andrikopoulos, V., & Avgeriou, P. (2020). Architecting systems of systems: A tertiary study. Information and Software Technology, 118, 106202.

Carvalho, F., & Meira, S. (2009, June). Towards an embedded software component quality verification framework. In Engineering of Complex Computer Systems, 2009 14th IEEE International Conference on (pp. 248-257). IEEE.

Chidamber, S. R., &Kemerer, C. F. (1994). A metrics suite for object oriented design. IEEE Transactions on software engineering, 20(6), 476-493.

Chiodo, M., Giusto, P., Hsieh, H., Jurecska, A., Lavagno, L., & Vincentelli, A. S. (1993, October). Synthesis of mixed software-hardware implementations from CFSM specifications. In International Workshop on Hardware-Software Co-design.

Choi, Y., Lee, S., Song, H., Park, J., & Kim, S. (2008, February). Practical S/W component quality evaluation model. In Advanced Communication Technology, 2008. ICACT 2008. 10th International Conference on (Vol. 1, pp. 259-264). IEEE.

Crosby, P. B. (1979). Quality is free: The art of making quality certain. New York: New American Library.

Delone, W. H., & McLean, E. R. (2003). The DeLone and McLean model of information systems success: a ten-year update. Journal of management information systems, 19(4), 9-30.

Dere, G. (2021). Biomedical Applications with Using Embedded Systems. In Data Acquisition-Recent Advances and Applications in Biomedical Engineering. IntechOpen.

Derr, K. W. (1995). Applying OMT: A Practical step-by-step guide to using the Object Modeling Technique. SIGS Publications, Inc.

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

Emilio, M. D. P. (2015). Embedded systems design for high-speed data acquisition and control. Springer International Publishing.

Gajski, D. D., Vahid, F., Narayan, S., & Gong, J. (1994). Specification and design of embedded systems. Prentice-Hall, Inc.

Ganssle, J. (2008). The art of designing embedded systems. Newnes.

Garcés, L., Ampatzoglou, A., Avgeriou, P., & Nakagawa, E. Y. (2017). Quality attributes and quality models for ambient assisted living software systems: A systematic mapping. Information and Software Technology, 82, 121-138.

Garvin, D. A., & Quality, W. D. (1984). Really mean. Sloan management review, 25.

Gauthier, L. (2001). Génération de système d'exploitation pour le ciblage de logiciel multitâche sur des architectures multiprocesseurs hétérogènes dans le cadre des systèmes embarqués spécifiques (Doctoral dissertation, Institut National Polytechnique de Grenoble-INPG).

Georgiadou, E. (2003). GEQUAMO—a generic, multilayered, cusomisable, software quality model. Softw. Qual. J. 11, 313–323.

Giusto, D., Iera, A., Morabito, G., & Atzori, L. (Eds.). (2010). The internet of things: 20th Tyrrhenian workshop on digital communications. Springer Science & Business Media.

Grady, R.B., Caswell, D.L. (1987). Software Metrics: Establishing a Company-Wide Program. Prentice Hall, Englewood Cliffs.

Guessi, M., Nakagawa, E. Y., Oquendo, F., & Maldonado, J. C. (2012, June). Architectural description of embedded systems: a systematic review. In Proceedings of the 3rd international ACM SIGSOFT symposium on Architecting Critical Systems (pp. 31-40). ACM.

Gupta, R. K. (2012). Co-synthesis of hardware and software for digital embedded systems (Vol. 329). Springer Science & Business Media.

Heath, S. (2002). Embedded systems design. Elsevier.

Herrera. M., Moraga. M. A., Caballero. I., Calero. C. (2010). Quality in Use Model for Web Portals (QiUWeP). In 10th International Conference on Web Engineering ICWE 2010 Workshops.

Highsmith, J. A., & Highsmith, J. (2002). Agile software development ecosystems. Addison-Wesley Professional.

IEEE Standard 1061–1998. (1998). IEEE Standard for a Software Quality Metrics Methodology. New York: IEEE Computer Society.

Institute of Electrical and Electronics Engineers. (1990). IEEE Standard Glossary of Software Engineering Terminology. USA.

Institute of Electrical and Electronics Engineers. (1998). IEEE Standard for a Software Quality Metrics Methodology.

International Electrotechnical Commission. (2006). IEC 62304: Logiciels de dispositifs médicaux - Processus du cycle de vie du logiciel. Geneva, Switzerland.

ISO, I. (2001). ISO/IEC 9126-1 Software Engineering – Product Quality – Part 1: Quality Model. International Organization for Standardization, Geneva, Switzerland.

ISO, I. (2003a). ISO/IEC 9126-2 Software Engineering – Product Quality – Part 2: External Metrics. International Organization for Standardization, Geneva, Switzerland.

ISO, I. (2003b). ISO/IEC 9126-3 Software Engineering – Product Quality – Part 3: Internal Metrics. International Organization for Standardization, Geneva, Switzerland.

ISO, I. (2004). ISO/IEC 9126-4 Software Engineering – Product Quality – Part 4: Quality in Use Metrics. International Organization for Standardization, Geneva, Switzerland.

ISO, I. (2005a). IEC 25000 Systems and software engineering–Systems and software Quality Requirements and Evaluation (SQuaRE)—Guide to SQuaRE. International Organization for Standardization.

ISO, I. (2005b). ISO 9000 Quality Management Systems—Fundamentals and Vocabulary. International Organization for Standardization, Geneva, Switzerland.

ISO, I. (2007a). IEC 25001 Systems and software engineering–Systems and software Quality Requirements and Evaluation (SQuaRE)— planning and management. International Organization for Standardization.

ISO, I. (2007b). 25020: Systems and Software Engineering-Systems and software Quality Requirements and Evaluation (SQuaRE)- measurement reference model and guide. International Organization for Standardization, Geneva, Switzerland.

ISO, I. (2007c). 25030: Systems and Software Engineering-Systems and software Quality Requirements and Evaluation (SQuaRE)- quality requirements. International Organization for Standardization, Geneva, Switzerland.

ISO, I. (2008). 25012: Systems and Software Engineering-Systems and software Quality Requirements and Evaluation (SQuaRE)- data quality model. International Organization for Standardization, Geneva, Switzerland.

ISO, I. (2010a). IEC/IEEE 24765: 2010 Systems And Software Engineering-Vocabulary. IEEE Standards Association, 24765, 418.

ISO, I. (2010b). 25045: 2010-Systems and Software Engineering-Systems and software Quality Requirements and Evaluation (SQuaRE) - evaluation module for recoverability. International Organization for Standardization, Geneva, Switzerland.

ISO, I. (2011a). 25010: 2011-Systems and Software Engineering-Systems and software Quality Requirements and Evaluation (SQuaRE)-System and Software Quality Models. International Organization for Standardization, Geneva, Switzerland.

ISO, I. (2011b). 25040: 2011-Systems and Software Engineering-Systems and software Quality Requirements and Evaluation (SQuaRE)- evaluation process. International Organization for Standardization, Geneva, Switzerland.

ISO, I. (2011c). ISO 26262-1: Road vehicles -- Functional safety' International Standard ISO/FDIS 26262. International Organization for Standardization, Geneva, Switzerland.

ISO, I. (2012a). 25010: 2012-Systems and Software Engineering-Systems and software Quality Requirements and Evaluation (SQuaRE)- quality measure element. International Organization for Standardization, Geneva, Switzerland.

ISO, I. (2012b). 25041: 2012-Systems and Software Engineering-Systems and software Quality Requirements and Evaluation (SQuaRE) - evaluation guide for developers, acquirers and independent evaluators. International Organization for Standardization, Geneva, Switzerland.

ISO, I. (2016). ISO/IEC 25023: Systems and software engineering – systems and software quality requirements and evaluation (SQuaRE) – Measurement of system and software product quality. International Organization for Standardization, Geneva, Switzerland.

Jeong, H. Y., & Kim, Y. H. (2012). A Quality Model of Lightweight Component for Embedded System. In Applied Mechanics and Materials (Vol. 121, pp. 4907-4911). Trans Tech Publications.

Jeong, H. Y., Park, J. H., & Jeong, Y. S. (2014). An ANP-based practical quality model for a secure embedded system with sensor network. International Journal of Distributed Sensor Networks.

Kalavade, A. P. (1997). System-level codesign of mixed hardware-software systems.

Kandt, R. K. (2005). Software Engineering Quality Practices. CRC Press.

Kantarci, B., & Mouftah, H. T. (2014). Trustworthy sensing for public safety in cloud-centric internet of things. IEEE Internet of Things Journal, 1(4), 360-368.

Kaur, S., & Singh, P. (2019). How does object-oriented code refactoring influence software quality? Research landscape and challenges. Journal of Systems and Software, 157, 110394.

Khaddaj, S., Horgan, G. (2005). A proposed adaptable quality model for software quality assurance. J. Comput. Sci. 1(4), 482–487.

Khadidja, Y. (2013). L'apport des outils de l'Intelligence Artificielle dans les systèmes temps réel: Ordonnancement des tâches (Doctoral dissertation, Université d'Oran).

Kiran, M., & Simons, A. (2016). Testing Software Services in Cloud Ecosystems. International Journal of Cloud Applications and Computing (IJCAC), 6(1), 42-58.

Kitchenham, B. (1987). Towards a constructive quality model. Part I: Software quality modelling, measurement and prediction. Softw. Eng. J. **2**(4), 105–113.

Kitchenham, B., & Pfleeger, S. L. (1996). Software quality: the elusive target [special issues section]. IEEE software, 13(1), 12-21.

Koopman, P. (2007). Reliability, safety, and security in everyday embedded systems. Lecture Notes in Computer Science, 4746, 1.

Kopetz, H. (2011). Real-time systems: design principles for distributed embedded applications. Springer Science & Business Media.

Kumar, S., Aylor, J. H., Johnson, B. W., & Wulf, W. A. (1995). The Codesign of Embedded Systems: A Unified Hardware/Software Representation: A Unified Hardware/Software Representation. Springer Science & Business Media.

Laprie, J. L. (1992). Dependability: Basic Concepts and Terminology. Dependable Computing and Fault-Tolerant Systems, Vol. 5.

Lee, E. A. (1999). Embedded Software: An Agenda for Research. Electronics Research Laboratory, College of Engineering, University of California.

Lee, E. A., & Seshia, S. A. (2016). Introduction to embedded systems: A cyber-physical systems approach. Mit Press.

Li, Q., & Yao, C. (2003). Real-time concepts for embedded systems. CRC press.

Liebel, G., Marko, N., Tichy, M., Leitner, A., & Hansson, J. (2018). Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice. Software & Systems Modeling, 17(1), 91-113.

Lyu, M.R. (ed.) (1996). Handbook of Software Reliability Engineering. IEEE Computer Society Press/McGraw-Hill, Silver Spring/New York.

Marir, T., Mokhati, F., Bouchlaghem-Seridi, H., Acid, Y., & Bouzid, M. (2016). QM4MAS: a quality model for multi-agent systems. International Journal of Computer Applications in Technology, 54(4), 297-310.

MARTE, U. (2015). UML profile for MARTE: modeling and analysis of real-time embedded systems.

Marwedel, P. (2003). Embedded System Design. Kluwer Academic Publishers, Dordrecht.

Marwedel, P. (2021). Embedded system design: embedded systems foundations of cyber-physical systems, and the internet of things (p. 433). Springer Nature.

Mäunch, J., Kläs, M. (2008). Balancing upfront definition and customization of quality models. In: Workshop-Band Software-Qualitätsmodellierung und -bewertung (SQMB 2008). TechnischeUniversitätMünchen.

McCall, J. A. (1977). Factors in software quality. US Rome Air development center reports.

McCall, J. A., Richards, P. K., & Walters, G. F. (1977). Factors in software quality. volumei. concepts and definitions of software quality. GENERAL ELECTRIC CO SUNNYVALE CA.

Mohsin, A., & Janjua, N. K. (2018). A review and future directions of SOA-based software architecture modeling approaches for System of Systems. Service Oriented Computing and Applications, 12(3), 183-200.

Muhammad, N., Vandewoude, Y., Berbers, Y., & van Loo, S. (2010). Modelling embedded systems with AADL: a practical study. INTECH.

Nakagawa, E. Y., Gonçalves, M., Guessi, M., Oliveira, L. B., & Oquendo, F. (2013, July). The state of the art and future perspectives in systems of systems software architectures. In Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems (pp. 13-20). ACM.

Nelson, G., Saunders, A., & Playter, R. (2019). The petman and atlas robots at boston dynamics. Humanoid Robotics: A Reference, 169, 186.

Nicolescu, Gabriela, et al. "Desiderata pour la spécification et la conception des systèmes électroniques." TSI-Technique et Science Informatiques-RAIRO 21.3 (2002): 291-314.

Nikolaus, D. (2015). New software problem at Volkswagen. https://www.welt.de/print/die_welt/wirtschaft/article149285119/Neues-Software-Problem-bei-Volkswagen.html.

Oliveira, L. B. R., Guessi, M., Feitosa, D., Manteuffel, C., Galster, M., Oquendo, F., & Nakagawa, E. Y. (2013). An investigation on quality models and quality attributes for embedded systems. ICSEA, 13, 1-6.

Olsen, A. (2012). Systems engineering using SDL-92. Newnes.

Oman, P., & Hagemeister, J. (1992, November). Metrics for assessing a software system's maintainability. In Software Maintenance, 1992. Proceerdings., Conference on (pp. 337-344). IEEE.

Paulk, M. (1993). Capability maturity model for software. Encyclopedia of Software Engineering.

Pfleeger, S. L., & Atlee, J. M. (1998). Software engineering: theory and practice. Pearson Education India.

Plösch, R., Gruber, H., Hentschel, A., Körner, C., Pomberger, G., Schiffer, S Saft, M., Storck, S., (2008). The EMISQ method and its tool support-expert-based evaluation of internal software quality. Innovations in Systems and Software Engineering, 4(1), 3-15.

Pressman, R. S. (2010). Software engineering: a practitioner's approach (7th Edition). McGraw-Hill.

Pries, K. H., & Quigley, J. M. (2008). Project management of complex and embedded systems: Ensuring product integrity and program quality. Crc Press.

Quality. (2017). Oxford dictionaries. https://en.oxforddictionaries.com/definition/quality.

Rahman, L. F., Ozcelebi, T., & Lukkien, J. (2018). Understanding IoT systems: a life cycle approach. Procedia computer science, 130, 1057-1062.

Renaudin, M. (2000). Asynchronous circuits and systems: a promising design alternative. Microelectronic engineering, 54(1-2), 133-149.

RTCA (Firm). SC 167. (1992). Software considerations in Airborne Systems and equipment certification. RTCA, Incorporated.

Sales, D. C., & Becker, L. B. (2021). Approach for Evolving Sensing and Actuation Devices in Cyberphysical Systems Architectures. In MODELSWARD (pp. 306-313).

Serpanos, D., & Wolf, T. (2011). Architecture of network systems. Elsevier.

Shah, S. M. A., Sundmark, D., Lindström, B., & Andler, S. F. (2016). Robustness testing of embedded software systems: An industrial interview study. IEEE Access, 4, 1859-1871.

Sherman, T. (2008). Quality attributes for embedded systems. In Advances in Computer and Information Sciences and Engineering (pp. 536-539). Springer Netherlands.

Sicari, S., Cappiello, C., De Pellegrini, F., Miorandi, D., & Coen-Porisini, A. (2016). A security-and quality-aware system architecture for Internet of Things. Information Systems Frontiers, 18(4), 665-677.

Sommerville I. (2007). Software Engineering (8th Edition). China Machine Press.

Stankovic, J. A. (1988). Misconceptions about real-time computing: A serious problem for next-generation systems. Computer, 21(10), 10-19.

Suryn, W. (2013). Software quality engineering: a practitioner's approach. John Wiley & Sons.

Suryn, W., Abran, A., & April, A. (2003). ISO/IEC SQuaRE. the second generation of standards for software product quality.

Tamrabet, Z., Marir, T., & Mokhati, F. (2018). A Survey on Quality Attributes and Quality Models for Embedded Software. International Journal of Embedded and Real-Time Communication Systems (IJERTCS), 9(2), 1-17.

Tamrabet, Z., Marir, T., & Mokhati, F. (2022a). ESQuMo: An Embedded Software Quality Model. International Journal of Embedded and Real-Time Communication Systems (IJERTCS), 13(1), 1-18.

Tamrabet, Z., Marir, T., & Mokhati, F. (2022b, May 24th). Applying the ESQuMo Quality Model to Medical Imaging Embedded Software. The first International Conference on Autonomous Systems and their Applications (ICASA'22). Chadli Bendjedid El-Taref University. Algeria.

Vegendla, A., Duc, A. N., Gao, S., & Sindre, G. (2018). A systematic mapping study on requirements engineering in software ecosystems. Journal of Information Technology Research (JITR), 11(1), 49-69.

Wagner, S. (2013). Software product quality control. Berlin: Springer.

Wake, S., and Henry, S. "A Model Based on Software Quality Factors which Predict Maintainability." Proceeding of the conference on Software Maintenance, 1988, pp. 382-387.

Wijnstra, J. G. (2001, January). Quality attributes and aspects of a medical product family. In System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on (pp. 10-pp). IEEE.

Xu, H., & Zhuang, Y. (2014). A Formal Transformation Approach for Embedded Software Modeling. JSW, 9(4), 807-813.