Universidad CENFOTEC

Maestría en Ingeniería de Software con énfasis en Arquitectura y Diseño

Escuela de Ingeniería de Software

Trabajo Final de Graduación

“ISO/IEC 25000 Criteria in Assessing Cloud Native Application Quality”

Andrés Miranda-Arias

Marzo 2024

TRIBUNAL EXAMINADOR

Este proyecto fue aprobado por el Tribunal Examinador de la carrera: **Maestría Profesional en Ingeniería del Software con énfasis en Arquitectura y Diseño de Software,** requisito para optar por el título de grado de **Maestría,** para el estudiante: **Miranda Arias Andrés Felipe.**

CARLOS MARTIN FLORES GONZALEZ (FIRMA)
PERSONA FISICA, CPF-01-1145-0392.
Fecha declarada: 27/03/2024 10:56:36 PM
Razón: Firma de documentos
Lugar: San Jose, Costa Rica Contacto: cflores@ucenfotec.ac.cr

M.Sc. Carlos Martín Flores González
Tutor

JUAN DANIEL SANCHEZ CAMBRONERO (FIRMA)
Digitally signed by JUAN DANIEL SANCHEZ CAMBRONERO (FIRMA)
Date: 2024.04.03 08:39:10 -06'00'

M.Sc. Juan Daniel Sánchez Cambronero
Lector 1

IGNACIO TREJOS ZELAYA (FIRMA)
Firmado digitalmente por IGNACIO TREJOS ZELAYA (FIRMA)
Fecha: 2024.04.10 08:59:02 -06'00'

M.Sc. Ignacio Trejos Zelaya
Lector 2

San José, Costa Rica, 5 de marzo de 2024

Firmada digitalmente, de conformidad con la Ley de Certificados, Firmas Digitales y Documentos Electrónicos N° 8454, destacando el artículo 9°-

ISO/IEC 25000 Criteria in Assessing Cloud Native Application Quality

1st Miranda-Arias, Andrés
CENFOTEC University
San José, Costa Rica
amirandaa@ucenfotec.ac.cr

2nd Flores-González, Martín
Costa Rica Institute of Technology
Cartago, Costa Rica
cflores@tec.ac.cr

3rd Trejos-Zelaya, Ignacio
Costa Rica Institute of Technology and CENFOTEC University
San José, Costa Rica
itrejos@ucenfotec.ac.cr

***Abstract*—Technological updates, upgrades and trends on how software is developed have led to the introduction of a number of diverse architectural patterns, practices, and technologies, as well as to the obsolescence of standards, technology, patterns, and ways of creating software applications. Microservices architecture, DevOps, containerization, cloud computing, and many other new technologies have led to the introduction of Cloud Native Applications. Nonetheless, one crucial aspect that has been present regardless of technologies such as programming languages, or types of software applications, has been software product quality. This paved the way to the creation of ISO/IEC 9126 Software Product Quality, in 1991, and its successor, ISO/IEC 25000 System and Software Quality Requirements and Evaluation (SQuaRE) series in 2005. Our research assesses and analyzes this latter series of standards, aimed at creating an enriched version that comprises quality attributes concerning cloud native applications. After reviewing current literature on the subject, five different quality attributes were found that could be adopted into a Cloud Native Application focused ISO/IEC 25000 Standard Series: elasticity, loose coupling, observability, resiliency and scalability, along with one or more associated quality measures. The new measures, along with those that were considered applicable and relevant to the investigation, were applied to a case study to assess cloud native application quality with an extended ISO/IEC 25000.**

***Index Terms*—Cloud, Cloud Native, Cloud Native Applications, Elasticity, ISO, ISO/IEC 25000, Loose Coupling, Microservices, Observability, Resiliency, Scalability, Software Architecture, Software Quality, Software Quality Assessment.**

I. Introduction

Cloud Native Applications (CNAs) have been around for nearly 20 years in the software development industry. Nowadays, users can choose different cloud providers, mainly Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP). The modernization of current technology has led to an evolution from monolithic architectures to those organized via microservices, or from a waterfall software development cycle to agile cycles. All these recent changes on how software is architected and built has led to, in a way, an ISO/IEC 25000 standard which contains sections, or subsections, that can be considered relatively obsolete in present times.

Introduced in 2005, ISO/IEC 25000 series of standards serves as an assessment guideline of software product quality, with five different divisions and several documents within the entire series. Focusing mainly on quality tied to the software product itself, eight different quality attributes were selected and listed on this standard, specifically on the ISO/IEC 25010 document, along with several quality measures associated to each attribute:

Functional Suitability
Functional Completeness
Functional Correctness
Functional Appropriateness
Performance Efficiency
Time Behaviour
Resource Utilization
Capacity
Compatibility
Co-Existence
Interoperability
Usability
Appropriateness Recognizability
Learnability
Operability
User Error Protection
User Interface Aesthetics
Accessibility
Reliability
Maturity
Availability
Fault Tolerance
Recoverability
Security
Confidentiality
Integrity
Non-Repudiation
Authenticity
Accountability

Maintainability
Modularity
Reusability
Analysability
Modifiability
Testability
Portability
Adaptability
Installability
Replaceability

TABLE I
SYSTEM/SOFTWARE PRODUCT QUALITY MODEL

Despite the standards being adequate and relevant to the then current software applications, many concepts have appeared on the software engineering world since 2005. Concepts such as scalability, resiliency, loose coupling, to mention a few microservices-related characteristics, have been introduced and are not contemplated within the referred standard. The possibility of upgrading the standard, to include quality attributes tied directly to cloud native applications, motivated this investigation, which focuses on analyzing relevant, published literature and assessing which quality attributes, if any, that concern cloud native applications, should be included in a more current version of the standard.

Several research papers were found and studied in order to identify the different quality measures that appeared to be relevant for inclusion into a new release of the ISO/IEC 25000 standard, capable of encompassing cloud native software products, or even to be contemplated within development teams that implement the ISO/IEC 25000 standard within their software development cycles. As well, an analysis will take place to come up with valid quality measures for those quality attributes, following the standards within the ISO/IEC 25000 series, which can be seen in more detail further along. Prior to this investigation, a delimitation of the entire ISO/IEC 25000 series will take place, in order to select those documents, quality attributes and quality measures that can directly concern cloud native based software products. Once this investigation and evaluation has been done, they will be applied to a case study in order to assess ISO/IEC 25000 software quality attributes in a cloud native application, which will be an e-commerce application arbitrarily chosen.

The results obtained from this case study showed that there are different metrics that can be applied to any software product, therefore there can be applications that are "more cloud native" than others. As well, the overall investigation shows the necessity for an upgraded ISO/IEC 25000 standard, since there is an existing pool of quality attributes that are not mentioned on this 2005 version, maybe even taking the scope further since this investigation is based solely on cloud native applications.

It is important to note and clarify that, as part of the investigation, a new proposal of the ISO/IEC 25010 System/Software Product Quality Model has been found, designed on 2022, and which, at least at the time of this research, is under review, but no final updates and/or publications have taken place yet. Even with the ISO organization working on a new draft, the relevance of this investigation does not relapse but is rather boosted, taking into consideration the actual importance of both cloud native applications and a new version of this ISO/IEC standard in modern software development. A fully detailed model can be found later on on section III-C.

This article is organized as follows: sections II provides a background on Cloud Native Applications, the ISO/IEC 25000 Standard, software product quality attributes, and the concept of product vs. process quality. Section III describes the related work in software product quality assessment concerning cloud native applications, and any findings regarding the investigation topic. Section IV shows the proposed methodology and the upgrade proposal for the standard, taking into consideration the guidelines provided by the documents within the ISO/IEC 25000 series, in order to test within the case study. Section V shows the evaluation of the proposed quality attributes applied to a microservices application hosted on Amazon AWS services. Section VI show the results of said case study and present the final version of those quality attributes found that can potentially be added to the standard, concerning cloud native applications. Finally, section VII show the final conclusions obtained through the investigation, while section VIII suggests future research possibilities based on the findings of this paper.

II. BACKGROUND

A. ISO/IEC 25000: Systems and Software Quality Requirements and Evaluation - SQuaRE

Since its foundation in 1947, the International Organization for Standardization (ISO), has been in charge of developing international standards and standardization for different industries. In collaboration with the International Electrotechnical Commission (IEC), the ISO developed the ISO/IEC 25000 series in 2005, which has the goal of creating a framework for the evaluation of software product quality [7].

This standard, or series of standards, comprises five different divisions within the software product quality specification:

- ISO/IEC 2500n: Quality Management Division
- ISO/IEC 2501n: Quality Model Division
- ISO/IEC 2502n: Quality Measurement Division
- ISO/IEC 2503n: Quality Requirements Division
- ISO/IEC 2504n: Quality Evaluation Division

Each division contains one or more standards, or documents, with a specific objective, related to the software product quality area that is being analyzed. It is important to emphasize that there is an additional division, ranging from the ISO/IEC 25050 to ISO/IEC 25099, which "are reserved to be used for SQuaRE extension International Standards and/or Technical Reports" [8]. This extension division is made up of seven documents, however, this specific division is not within the scope of this investigation.

1) ISO/IEC 2500n: Quality Management Division: The first division of this series of standards contains the Quality Management Division, within the ISO/IEC 2500n. "The International Standards that form this division define all common models, terms and definitions referred to by all other standards from the SQuaRE series" [8]. This division is made up of two documents, *25000 - Guide to SQuaRE*, which "provides the SQuaRE architecture model, terminology, documents overview, intended users and associated parts of the series as well as reference models", and also the *25001 - Planning and Management* document, which "provides requirements and guidance for a supporting function, which is responsible for the management of system or software product requirements specification and evaluation" [8].

2) ISO/IEC 2501n: Quality Model Division: The second division of this series of standards contains the Quality Model Division, within the ISO/IEC 2501n. "The International Standards that form this division present detailed quality models for systems and software product, quality in use and data" [8]. This division contains itself two documents, first the *25010 - Quality Model* document, which "describes the model for system and software product quality and quality in use", and second the *25012 - Data Quality Model* document, which "defines a general data quality model for data retained in a structured format within a computer system" [8].

3) ISO/IEC 2502n: Quality Measurement Division: The third division of this series of standards contains the Quality Measurement Division, within the ISO/IEC 2502n. "The International Standards that form this division include a system and software product quality measurement reference model, mathematical definitions of quality measures, and practical guidance for their application" [8]. This division is divided into five different documents. First, the *25020 - Measurement Reference Model and Guide*, which "presents introductory explanation and a reference model that is common to quality measure elements, measures of internal software quality, external system and software quality and quality in use" [8]. Second, the *25021 - Quality Measure Elements* document, which "presents definitions and specifications of a set of recommended base and derived measures, which are intended to be used during the whole system or software development life cycle" [8]. The third document within this division is the *25022 - Measurement of Quality in Use*[1] document, that "describes a set of measures for measuring quality in use in terms of characteristics and subcharacteristics defined in ISO/IEC 25010, and is intended to be used together with ISO/IEC 25010" [8]. The fourth document is the *25023 - Measurement of System and Software Product Quality*, which "defines quality measures for quantitatively measuring system and software product quality in terms of characteristics and subcharacteristics defined in ISO/IEC 25010, and is intended to be used together with ISO/IEC 25010" [8]. The fifth and final document of this division is the *25024 - Measurement of Data Quality*, that "defines quality measures for quantitatively measuring data in terms of characteristics defined in ISO/IEC 25012" [8].

4) ISO/IEC 2503n: Quality Requirements Division: The fourth division of this series of standards contains the Quality Requirements Division, within the ISO/IEC 2503n. "The International Standard that forms this division helps specifying quality requirements" [8]. This division is made up of a single document, *25030 - Quality Requirements*, which "provides requirements and guidance for the process used to specify quality requirements, as well as requirements and recommendations for quality" [8].

5) ISO/IEC 2504n: Quality Evaluation Division: The fifth and final division of this series of standards contains the Quality Evaluation Division, within the ISO/IEC 2504n. "The International Standards that form this division provide requirements, recommendations and guidelines for product evaluation, whether performed by independent evaluators, acquirers or developers" [8]. This division is made up of three documents, the first being the *25040 - Evaluation Process* document, which "contains requirements and recommendations for the evaluation of system or software product quality and clarifies the general concepts" [8]. The second document is the *25041 - Evaluation Guide for Developers, Acquirers and Independent Evaluators*, which "contains specific requirements and recommendations for developers, acquirers and evaluators" [8]. Finally, the last document of this division is the *25045 - Evaluation Modules for Recoverability*, that "provides the specification to evaluate the subcharacteristics of recoverability defined under the characteristic of reliability of the quality model" [8].

Given the extension not only of the ISO/IEC 25000 series, but also of all its five divisions with several documents within, it is necessary to delimit the scope of the analysis, with relation to assessing cloud native application quality. This being said, the following documents will fall within the scope of this investigation, and will be used, totally or partially, to help assess quality of cloud native applications:

- ISO/IEC 2501n: Quality Model Division
  - ISO/IEC 25010 Quality Model
- ISO/IEC 2502n: Quality Measurement Division
  - ISO/IEC 25020 Measurement Reference Model and Guide
  - ISO/IEC 25021 Quality Measure Elements
  - ISO/IEC 25023 Measurement of System and Software Product Quality
- ISO/IEC 2504n: Quality Evaluation Division
  - ISO/IEC 25040 Evaluation Process

The ISO/IEC 25010 Quality Model document defines both the Quality in Use Model, and also the Product Quality Model. Both models are defined and developed under the quality model structure, which defines a quality, which in turn is composed of several characteristics, which they as well are composed of subcharacteristics or properties. The Quality in Use Model "defines five characteristics related to outcomes

[1]Degree to which a product or system can be used by specific users to meet their needs to achieve specific goals with effectiveness, efficiency, freedom from risk and satisfaction in specific contexts of use [8]

of interaction with a system [...]. Each characteristic can be assigned to different activities of stakeholders [...]" [9].

Given that the Quality in Use Model is composed of five characteristics that directly relate of stakeholder interaction with the system or software product, which are effectiveness, efficiency, satisfaction, freedom of risk, and context coverage, as well as one or more subcharacteristics each, and that do not directly relate to cloud native application quality, this model will not be taken into consideration in this investigation.

Nonetheless, the Product Quality Model does directly apply to the scope, given that this model "categorizes system/-software product quality properties into eight characteristics: functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability and portability. [...] The product quality model can be applied to just a software product, or to a computer system that includes software, as most of the subcharacteristics are relevant to both software and systems" [9]. The complete Product Quality Model, including the eight characteristics and each of their related subcharacteristics can be seen in table I on section I.

Once again, given the extension of this Product Quality Model, it will be reduced according to what characteristics, or subcharacteristics, will be or relevance to the assessing of cloud native application quality. Those characteristics and subcharacteristics that will be the base for the assertion are the following:

- Performance Efficiency
  - Time-behaviour: "degree to which the response and processing times and throughput rates of a product or system, when performing its functions, meet requirements" [9].
  - Resource Utilization: "degree to which the amounts and types of resources used by a product or system, when performing its functions, meet requirements" [9].
  - Capacity: "degree to which the maximum limits of a product or system parameter meet requirements".[2] [9].
- Reliability
  - Availability: "degree to which a system, product or component is operational and accessible when required for use" [9].
  - Fault Tolerance: "degree to which a system, product or component operates as intended despite the presence of hardware or software faults" [9].
  - Recoverability: "degree to which, in the event of an interruption or a failure, a product or system can recover the data directly affected and re-establish the desired state of the system" [9].
- Maintainability
  - Modularity: "degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components" [9].
  - Reusability: "degree to which an asset can be used in more than one system, or in building other assets" [9].
  - Analysability: "degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product of deficiencies or causes of failures to identify parts to be modified" [9].
  - Modifiability: "degree to which a product or system acn be effectively and efficiently modified without introducing defects or degrading existing product quality" [9].
  - Testability: "degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met" [9].
- Portability
  - Adaptability: "degree to which a product or system can effectively and efficiently be adapted for different or evolving hardware, software or other operational or usage environments" [9].

Despite there being other characteristics with subcharacteristics within the Product Quality Model, such as Functional Suitability, Compatibility, Usability, and Security, with their respective subcharacteristics, as well as Maturity as a subcharacteristic of Reliability, and Installability and Replaceability as subcharacteristics of Portability, will not be taken within the scope since they do not provide relevance to the cloud native application quality assertion. On a separate note, it is important to emphasize that cloud native applications might include more quality characteristics that are not part of the ISO/IEC 25010 document.

The ISO/IEC 25020 Measurement Reference Model and Guide document brings into the series the software product quality measurement reference model. This model "describes the relationship between a quality model, its associated quality characteristics (and subcharacteristics), and software product attributes with the corresponding software quality measures, measurement functions, quality measure elements, and measurement methods" [10]. This document will work as reference and guidance on how the assertion of the cloud native application quality will be done, it will also provide insights on the reference model, on selecting software quality measures, and also on constructing software quality measures.

On another hand, the ISO/IEC 25021 Quality Measure Elements document provides both guidance to understand how to measure quality in a software product, and a template to apply these concepts. The document presents two concepts, quality measure (QM) and quality measure element (QME). The first is described as "derived measure that is defined as a measurement function of two or more values of quality

[2]Parameters can include the number of items that can be stored, the number of concurrent users, the communication bandwidth, throughput of transactions, and size of database [9]

measure elements" [11], while a quality measure element is defined as "measure defined in terms of a property and the measurement method for quantifying it, including optionally the transformation by a mathematical function" [11]. The template to develop the quality measure elements is used to properly create and design a QME to provide helpful information on the QM. Basically, the quality measure elements provide a method to quantify quality characteristics, through the process of developing a QME, which in turn will provide a measurement function, who in the end will come up with a quality measure. The entire format for the QMEs can be seen in section IX-A.

The final document of the Quality Measurement Division to be considered into the scope will be ISO/IEC 25023 Measurement of System and Software Product Quality. This document defines the quality of a software product as the "degree to which it satisfies the stated and implied needs of its various stakeholders, and thus provides value. [...] The measurable quality-related properties of a system/software product are called properties to quantify and can be associated with quality measures. These properties are measured by applying a measurement method. A measurement method is a logical sequence of operations used to quantify properties with respect to a specified scale. The result of applying a measurement method is called a quality measure element." [12]. This particular document will also provide a format for the correct and precise documentation of the quality measures applied to a software product, which can be viewed in full in section IX-B. Finally, this document will also provide a complete set of measurements, with their respective name, description, and most importantly, mathematical formula to calculate and evaluate such measurements, crucial aspect in the assertion of cloud native application quality. The complete description and mathematical formula of the quality subcharacteristics and their measurements can be seen in full in section II-C1.

The last document that will be used and taken into consideration is the ISO/IEC 25040 Evaluation Process document, from the Quality Evaluation Division. This document presents the software product quality evaluation reference model, which basically explains that the evaluation process must take in constraints, inputs and resources for the evaluation, and through such process, outcomes for the evaluation will be obtained [13]. Additionally, this document will provide the basis for the software product quality evaluation process, establishing all the requirements to begin the evaluation process with its respective documentation to present the results, including how to specify the evaluation, define requirements, define decision criteria, designing the evaluation itself, amongst other crucial aspects.

B. Cloud Native Applications

According to the Cloud Native Computing Foundation (CNCF), "Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. [...] These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil" [1]. In 2015, the CNCF was founded by the Linux Foundation to support the open-source community in developing critical cloud-native components [2].

From this entire cloud native perspective, came cloud native applications. Such applications are software products which are built from multiple, small, interdependent services called microservices. Through this new approach, developers are able to break large functionalities into smaller microservices, thus making cloud native applications more agile, since these microservices work independently and take minimal computing resources to run [2].

With the introduction of cloud native, microservices, and cloud native applications into the software industry, the traditional way of building and developing applications has changed drastically. Despite this not being the only way of creating new software products, due to the existence of, for example, serverless applications, event-driven applications, and, of course, the traditional monolithic application, cloud native applications have become more and more popular due to the benefits they bring to the table. This type of development will allow for a collaborative approach, and will be highly scalable on different cloud platforms, giving developers the opportunity to heavily automate building, testing, and deploying cloud native applications, therefore allowing them to focus on developing functionalities rather than focusing on the CI/CD aspects of the process.

Cloud native applications also bring a new type of architecture to the software industry: the cloud native application architecture. The CNCF lists some key aspects of this architecture: immutable infrastructure, microservices, declarative APIs, containers, and service meshes [1]. According to Amazon Web Services (AWS), each concept is defined as follows:

- Immutable Infrastructure: "servers for hosting cloud-native applications remain unchanged after deployment. If the application requires more computing resources, the old server is discarded, and the app is moved to a new high-performance server" [2].
- Microservices: "small, independent software components that collectively perform as complete cloud-native software. Each microservice focuses on a small, specific problem. Microservices are loosely coupled, which means that they are independent software components that communicate with each other" [2].
- API: "[communication] method that two or more software programs use to exchange information. Cloud-native systems use APIs to bring the loosely coupled microservices together" [2].
- Service mesh: "software layer in the cloud infrastructure that manages the communication between multiple microservices" [2].
- Containers: "smallest compute unit in a cloud-native application. They are software components that pack the microservice code and other required files in cloud-native

systems. By containerizing the microservices, cloud-native applications run independently of the underlying operating system and hardware" [2].

This architecture certainly brings many benefits to developers and the software industry in general. Given that by implementing an immutable infrastructure, and thus avoiding manual upgrades and patches, the deployment of cloud native applications becomes a predictable process. As well, the implementation of a microservices architecture, will bring many key aspects as high availability, high fault tolerance, loosely coupled components, that will continue running and working despite the eventually potential failure of one or more components, and also allowing developers to run bug fixes or patches in one or more microservices, according to the specific requirements, without having the application down from production in the meantime [3].

A crucial aspect of cloud native application development has been the switch in mentality of developers and the transition from the different development processes, architectures, infrastructures, and deployment and packaging of applications throughout the years. This switch from waterfall processes, to agile methodologies, now into DevOps, or from monolithic applications into microservices, or from physical on-premises servers to containers, and finally from on-premises datacenters to cloud environments has allowed cloud native applications and development to come to existence and change how application development and deployment is done [3].

From an AWS perspective, continuous integration (CI), continuous development (CD), DevOps, and serverless [computing] are key and common cloud native development practices nowadays [2], and define them as follows:

- Continuous Integration (CI): "software practice in which developers integrate changes into a shared code base frequently and without errors. Small, frequent changes make development more efficient because you can identify and troubleshoot issues faster" [2].
- Continuous Delivery (CD): "software practice that supports cloud-native development. [...] development teams ensure that the microservices are always ready to be deployed to the cloud. [...] CI and CD work together for efficient software delivery" [2].
- DevOps: "software culture that improves the collaboration of development and operations teams. It is a design philosophy that aligns with the cloud-native model. DevOps practices allow organizations to speed up the software development lifecycle" [2].
- Serverless [Computing]: "cloud-native model where the cloud provider fully manages the underlying server infrastructure. [...] the cloud infrastructure automatically scales and configures to meet the application requirements. [...] The serverless architecture automatically removes computes resources when the app stops running" [2].

Taking all these aspects into consideration, cloud native has brought a new way of developing and deploying software applications into production environments, making it not only easier since the development teams do not necessarily need to focus on all these aspects, but also quicker as well. Certainly, this new technology and new concept will bring new quality criteria that may not necessarily be present in those defined in the system and software product quality model defined by the ISO/IEC 25010 in section II-A2, for example new terms like elasticity and resilience, or, on the other hand, quality criteria that come from a microservices perspective like loosely coupled components.

C. Quality Attributes of Cloud Native Applications

Given that the first edition of the ISO/IEC 25000 series predates the introduction of Cloud Native Applications into the software industry for approximately 10 years, many quality attributes and characteristics from a more modern software development world are part of CNA, but are not part of the Product Quality Model mentioned in the ISO/IEC 25010 document and seen in table I.

Evidently, CNAs includes the quality attributes mentioned in the ISO/IEC 25010, with all 8 characteristics and their respective subcharacteristcs being part of these applications. Nonetheless, with all the technological advances between the publication of the first edition of the series, and the introduction of CNA into software development, new quality attributes have been added as well into the new cloud native applications. These quality attributes include, as the most common ones, elasticity, scalability, and resilience, amongst others.

1) Cloud Native Application Quality Attributes within the ISO/IEC 25010 Scope: Being part of the software development industry, cloud native applications can also adhere to the Product Quality Model described in the ISO/IEC 25010 document [9], by taking into account all the eight quality characteristics and subcharacteristics described in it. CNAs do, in fact, take into consideration said characteristics, and can be measured as described in the ISO/IEC 25023 document [12].

The eight main quality characteristics are functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability and portability. Each one of them has one or more subcharacteristics, each with their own measurement and mathematical formula to quantify them. Despite all subcharacteristics having their own quantifiable method, it is important to recall that some of these subcharacteristics have been left outside the scope of this investigation, as stated in the list II-A5 found in section II-A.

With the subcharacteristics chosen to be inside the scope of this investigation, the mathematical way to quantify them, according to the ISO/IEC 25023 [12], is separated into categories, following the characteristic/subcharacteristic(s) format used in the ISO/IEC 25000 series, with each subcharacteristic having one or more measurements.

Performance Efficiency:

- Time-behaviour measures: "[...] are used to assess the degree to which the response and processing times and throughput rates of a product or system when performing its functions meet the requirements".

- Mean response time: “how long is the mean time taken by the system to respond to a user task or system task”.

$$X = \sum_{i=1 \to n} (A_i)/n$$

with A_i being the time taken by the system to respond to a specific user task or system task at the i-th measurement, and n being the number of responses measured.
- Response time adequacy: “how well does the system response time meet the specified target”.

$$X = A/B$$

with A being the mean response time measured by the formula above, and B being the target response time specified.
- Mean turnaround time: “what is the mean time taken for completion of a job or an asynchronous process”.

$$X = \sum_{i=1 \to n} (B_i - A_i)/n$$

with A_i being the time of starting a job i, B_i being the time of completing the job i, and n being the number of measurements.
- Turnaround time adequacy: “how well does the turnaround time meet the specified targets”.

$$X = A/B$$

with A being the mean turnaround time measured with the formula above, and B being the target turnaround time specified.
- Mean throughput: “what is the mean number of jobs completed per unit time”.

$$X = \sum_{i \to n} (A_i/B_i)/n$$

with A_i being the number of jobs completed during the i-th observation time, B_i being the i-th observation time period, and n being the number of observations.

- Resource utilization measures: “[...] are used to assess the degree to which the amounts and types of resources used by a product or system when performing its functions meet the requirements”.
  - Mean processor utilization: “how much processor time is used to execute a given set of tasks compared to the operation time”.

$$X = \sum_{i=1 \to n} (A_i/B_i)/n$$

with A_i being the processor time actually used to execute a given set of tasks in observation i, B_i being the operation time to perform the tasks in observation i, and n being the number of observations.
  - Mean memory utilization: “how much of memory is used to execute a given set of tasks compared to the available memory”.

$$X = \sum_{i=1 \to n} (A_i/B_i)/n$$

with A_i being the size of memory actually used to perform a given set of tasks for the i-th sample processing, B_i being the size of memory available to perform the tasks during i-th sample processing, and n being the number of samples processed.
  - Mean I/O devices utilization: “how much of I/O device busy time is used to perform a given set of tasks compared to the I/O operation time”.

$$X = \sum_{i=1 \to n} (A_i/B_i)/n$$

with A_i being the duration of I/O device(s) busy time to perform a given set of tasks during the i-th observation, B_i being the duration of I/O operations to perform the tasks for the i-th observation, and n being the number of observations.
  - Bandwidth utilization: “what proportion of the available bandwidth is utilized to perform a given set of tasks”.

$$X = A/B$$

with A being the bandwidth of actual transmission measured over time to perform a given set of tasks, and B being the bandwidth capacity available to perform a given set of tasks.
- Capacity measures: “[...] are used to assess the degree to which the maximum limits of a product or system parameter meet the requirements”.
  - Transaction processing capacity: “how many transactions can be processed per unit time”.

$$X = A/B$$

with A being the maximum number of transactions completed during observation time, and B being the duration of the observation.
  - User access capacity: “how many users can access the system simultaneously at a certain time”.

$$X = \sum_{i=1 \to n} A_i/n$$

with A_i being the maximum number of users who can simultaneously access the system at i-th observation, and n being the number of observations.
  - User access increase adequacy: “how many users can be added successfully per unit time”.

$$X = A/B$$

with A being the number of users successfully added during observation time, and B being the duration of observation.

Reliability:

- Availability measures: "[...] are used to assess the degree to which a system, product or component is operational and accessible when required for use".
  - System availability: "for what proportion of the scheduled system operational time is the system actually available".

    $$X = A/B$$

    with A being the system operation time actually provided, and B being the system operation time specified in the operation schedule.
  - Mean down time: "how long does the system stay unavailable when a failure occurs".

    $$X = A/B$$

    with A being the total downtime, and B the number of breakdowns observed.
- Fault tolerance measures: "[...] are used to assess the degree to which a system, product or component operates as intended despite the presence of hardware or software faults".
  - Failure avoidance: "what proportion of fault patterns has been brought under control to avoid critical and serious failures".

    $$X = A/B$$

    with A being the number of avoided critical and serious failure occurrences (based on test cases), and B being the number of executed test cases of fault pattern (almost causing failure) during testing.
  - Redundancy of components: "what proportion of system components is installed redundantly to avoid system failure".

    $$X = A/B$$

    with A being the number of system components redundantly installed, and B being the number of system components.
  - Mean fault notification time: "how quickly does the system report the occurrence of faults".

    $$X = \sum_{i=1 \to n} (A_i - B_i)/n$$

    with A_i being the time at which the fault i is reported by the system, B_i being the time at which the fault i is detected, and n being the number of faults detected.
- Recoverability measures: "[...] are used to assess the degree to which, in the event of an interruption or a failure, a product or system can recover the data directly affected and re-establish the desired state of the system".
  - Mean recovery time: "how long does it take for the software/system to recover from failure".

    $$X = \sum_{i=1 \to n} A_i/n$$

    with A_i being the total time to recover the downed software/system and re-initiate operation for each failure i, and n being the number of failures.
  - Backup data completeness: "what proportion of data items is backed up regularly".

    $$x = A/B$$

    with A being the number of data items actually being backed up regularly, and B being the number of data items requiring backup for error recovery.

Maintainability:

- Modularity measures: "[...] are used to assess the degree to which a system or computer program is composed of discrete components such that change to one component has minimal impact on other components".
  - Coupling of components: "how strongly are the components independent and how many components are free from impacts from changes to other components in a system or computer program".

    $$X = A/B$$

    with A being the number of components which are implemented with no impact on others, and B being the number of specified components which are required to be independent.
  - Cyclomatic complexity adequacy: "how many software modules have acceptable cyclomatic complexity".

    $$X = 1 - A/B$$

    with A being the number of software modules which have a cyclomatic complexity score that exceeds the specified threshold[3], and B being the number of software modules implemented.
- Reusability measures: "[...] are used to assess the degree to which an asset can be used in more than one system or in building other assets.
  - Reusability of assets: "how many assets in a system can be reusable".

    $$X = A/B$$

    with A being the number of assets which are designed and implemented to be reusable, and B being the number of assets in a system.
  - Coding rules conformity: "how many modules conform to required coding rules".

    $$X = A/B$$

    with A being the number of software modules conforming to coding rules for a specific system, and B being the number of software modules implemented.

[3]Such a threshold is used to determine whether a value of cyclomatic complexity is acceptable or not for each module. This is defined by each project or organization and is possibly a different value for a programming language, a type of module or function [12]

- Analysability measures: "[...] are used to assess the degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failure, or to identify parts to be modified".
  - System log completeness: "to what extend does the system record its operations in logs so that they are to be traceable".

    $$X = A/B$$

    with A being the number of logs that are actually recorded in the system, and B being the number of logs for which audit trails are required during operation.
  - Diagnosis function effectiveness: "what proportion of the diagnosis functions meets the requirements of causal analysis".

    $$X = A/B$$

    with A being the number of diagnostic functions useful for causal analysis, and B being the number of diagnostic functions implemented.
  - Diagnosis function sufficiency: "what proportion of the required diagnosis functions has been implemented".

    $$X = A/B$$

    with A being the number of diagnostic functions implemented, and B being the number of diagnostic functions required.
- Modifiability measures: "[...] are used to assess the degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality".
  - Modification efficiency: "how effectively are the modifications made compared to the expected time".

    $$X = \sum_{i=1 \to n} (A_i/B_i)/n$$

    with A_i being the total work time spent for making a specific type of modification i, B_i being the expected time for making the specific type of modification i, and n being the number of modifications measured.
  - Modification correctness: "what proportion of modifications has been implemented correctly".

    $$X = 1 - (A/B)$$

    with A being the number of modifications that caused an incident or failure within a defined period after being implemented, and B being the number of modifications implemented.
  - Modification capability: "to what extent are the required modifications made within a specified duration".

    $$X = A/B$$

    with A being the number of items actually modified within a specified duration, and B being the number of items required to be modified within a specified duration.
- Testability measures: "[...] are used to assess the degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met".
  - Test function completeness: "how completely are test functions and facilities implemented".

    $$X = A/B$$

    with A being the number of test functions implemented as specified, and B being the number of test functions required.
  - Autonomous testability: "how independently can the software be tested".

    $$X = A/B$$

    with A being the number of tests that can be simulated by stub[4] among the test which depend on other systems, and B being the number of tests which depend on other systems.
  - Test restartability: "how easily can the operation test be carried out from the restart point after maintenance".

    $$X = A/B$$

    with A being the number of cases in which maintainer can pause and restart executing test run at desired points to check step by step, and B being the number of cases in which executing test run can be paused.

Portability:

- Adaptability measures: "[...] are used to assess the degree to which a product or system can effectively and efficiently be adapted for different or evolving hardware, software or other operational usage environments".
  - Hardware environmental adaptability: "is software or system capable enough to adapt itself to different hardware environment".

    $$X = 1 - A/B$$

    with A being the number of functions which were not completed or results which were insufficient to meet requirements during testing, and B being the number of functions which were tested in different hardware environment.
  - System software environmental adaptability: "is software or system capable enough to adapt itself to different system software environment".

    $$X = 1 - A/B$$

[4]A stub is a skeletal or special-purpose implementation of a software module used to develop or test a module that calls or is otherwise dependent on it [12].

with A being the number of functions which were not completed or results which were insufficient to meet requirements during testing, and B being the number of functions which were tested in different system software environment.

- Operational environment adaptability: "is software or system capable enough to adapt itself to different operational environment".

$$X = 1 - A/B$$

with A being the number of functions which were not completed or results which were insufficient to meet requirements during operational testing with user's environment, and B being the number of functions which were tested in different operational environment.

2) Non-ISO/IEC 25010 Quality Attributes of Cloud Native Applications: With the introduction of cloud technologies into the software development industry, as well as other concepts like microservices[5], serverless[6] computing and others, new quality attributes have come into appearance. Some of these new attributes now are part of the non-negotiables in CNAs.

Since CNAs are based off a microservices application architecture [3], it brings into the scene the quality attributes a microservices architecture has. Some of these characteristics include agility, scalability, flexibility (or elasticity), reliability, availability, observability, resiliency, and loose coupling [19].

Despite some of these characteristics being already part of the Product Quality Model described on the ISO/IEC 25010 document [9], others are completely out of its scope. Other authors add some other quality attributes to CNAs, such as resiliency, but separated into both redundancy and adaptability, modularity, and scalability is mentioned again too [20], once again pin-pointing towards the importance of building scalable software products nowadays.

Finally, a third set of authors added new quality attributes to CNAs, after creating their own definition of what a cloud native application is, and they included automation, manageability, observability [and monitoring], containerization, and, once again, scalability [21].

In summary, several authors through either book or paper publications [14], [15], [17], [22], [24]–[47], [49], [52]–[54] have enlisted some of these new quality attributes attached to the new cloud native applications, and have given definitions of what each one of these characteristics is. Nonetheless, there might still not be a way to quantify all these attributes, as there is with one or more measure per subcharacteristic as defined in the ISO/IEC 25023 and seen in detail, at least those characteristics within scope, in section II-C1.

[5]"Microservices are an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs. These services are owned by small, self-contained teams" [4]

[6]"A serverless architecture is a way to build and run applications and services without having to manage infrastructure. [...] You no longer have to provision, scale, and maintain servers to run your applications, databases, and storage systems" [5].

Having a definition of each one of these new quality characteristics is just the beginning, since having a way to verify and validate their presence within cloud native applications is just as important as knowing what they are. As well, being able to quantify them, could be through a mathematical formula, a binary or boolean indicator of their presence or lack of it, or some other way, leads to knowing if these attributes are not only present within CNAs, but also works as a way to determine the level of presence, and even possibly identify improvement areas.

Being able to define, quantify, verify and validate all these new quality attributes and characteristics mentioned like agility, scalability, flexibility, availability, and others, is a crucial part to eventually modify the ISO/IEC 25000 series of standards for software product quality to properly include the technological advances seen in present days not only with the emergence of new programming languages, frameworks, or any other advances in either software or hardware, but also with cloud native applications themselves, with the continued growth in use in software development.

D. Product vs. Process: The 12 Factor App and Different Quality Attributes

Despite the main focus of this investigation being solely on those quality attributes described in the Product Quality Model, mentioned in the Quality Measurement Division on section II-A3, particularly on the ISO/IEC 25023 document, there are other quality attributes that could be related to the process of development, or to the process of acquiring and describing the requirements for a software system or product, similar to that mentioned in the Quality Requirements Division, on section II-A4.

A clear example of how this can be separated into quality attributes regarding the product, and quality attributes regarding the process, is the 12 Factor App concept. According to [58], this is a methodology "for building software-as-a-service apps that use declarative formats for setup automation, have a clean contract with the underlying operating system, are suitable for deployment on modern cloud platforms, minimize divergence between development and production, and can scale up without significant changes in tooling".

This differentiation between product and process quality attributes can potentially lead to describing and recording system or product requirements in a way to ensure the complete incorporation of the product quality attributes described by the ISO/IEC series, as well as those that can come up during this investigation. By doing so, as a previous stage of development, creating the software requirements in this way will guarantee that these attributes will not only be present, but will also adhere to the ISO/IEC standards of software quality.

Going back to the 12 Factor App, those factors mentioned by the different authors can relate both to product and to process as well, backing up, in a certain way, the importance of this differentiation as stated as well in the ISO/IEC 2502n division and the ISO/IEC 2503n division. These factors are, as listed in [58]:

- Codebase: one codebase tracked in revision control, many deploys.
- Dependencies: explicitly declare and isolate dependencies.
- Config: store config in the environment.
- Backing services: treat backing services as attached resources.
- Build, release, run: strictly separate build and run stages.
- Processes: execute the app as one or more stateless processes.
- Port binding: export services via port binding.
- Concurrency: scale out via the process model.
- Disposability: maximize robustness with fast startup and graceful shutdown.
- Dev/prod parity: keep development, staging, and production as similar as possible.
- Logs: treat logs as event streams.
- Admin processes: run admin/management tasks as one-off processes.

Overall, this separation of attributes into the product and process categories can lead to potential future work, given that the main focus of this investigation relies principally on those quality attributes tied directly to the product, as seen in the ISO/IEC 2502n division series.

E. Literature Selection Criteria

For our research, various literature sources will be taken into consideration for analysis. These sources will be taken from reliable sources, such as published books or papers from different authors. The main sources of information will be websites such as Google Scholar, IEEE Digital Library, Core, AWS, ISO/IEC, amongst others.

The main language to consider sources as valid will be English, nonetheless, since all authors of this research are native Spanish speakers, said language will be considered as a secondary one too. Considering all literature will have credibility from the software engineering and software development communities, as well as will be obtained from reliable sources, the main differentiation criteria in terms of selection will be from the experience and perspectives of the authors, from a point of view of their academic and professional backgrounds.

This selection of sources has a main goal of reducing, to the smallest possible degree, the total amount of literature that will be used, without affecting the overall credibility and reliability of this investigation, and taking into consideration only those sources considered relevant to cloud native applications and software product quality.

The overall research of sources will be carried out through keywords, were books, research papers, articles, websites, blogs, amongst others, will be a match, to a lesser or greater extent, of the main investigation topic, and excluding, evidently, those that do not match or do not contain relevant information.

After applying these filters and search criteria, different literature was obtained as basis for our investigation. From the ISO Organization, the entire ISO/IEC Standards Series was obtained, and was then granulated to what was considered relevant to this investigation. As well, from Google Scholar, IEEE Digital Library, IEEE Xplore, Springer, and other sources of information, various research papers and articles were obtained.

Sources such as the Cloud Native Computing Foundation, Amazon Web Services, RedHat, Google Cloud Platform, Microsoft Azure, and others, served as sources of definitions. Finally, books, blogs, and other literature was selected too, all which were considered relevant sources of information to our investigation. All sources contemplated and taken into consideration, to lesser or greater extent, can be seen listed in section VIII.

III. Related Work

Several studies have been published in the last couple of years, being mostly created by Robin Lichtenthäler and Guido Wirtz. These recent studies show the importance of investigation in the field of cloud native applications in relation to quality attributes, but not necessarily tied to the ISO/IEC 25010 Product Quality Model.

A. A New Quality Model Specific for Cloud Native Applications

In 2022, authors Lichtenthäler and Wirtz, came up with several research papers revolving around CNAs, quality attributes and models, as well as ways to validate this new quality model proposed.

Their research begins with a review of different approaches for a quality model in the context of CNAs. This initial research paper led to a validations of those different approaches used in literature, and concluded that "in an early design phase, surveys and expert interviews are suitable to validate quality attributes and their relations while for complete quality models quantitative validations through measures are advised" [18].

Consequently, a proposal for a new quality model for cloud native applications, "aligned with the Quamoco meta model and based on both practitioner books and scientific literature" [14] was established. This model is based off the ISO/IEC 25010 Product Quality Model as a basic standard for orientation and familiarity, and is an expansion of said model, including new quality attributes like scalability, elasticity, and other attributes mentioned before.

This led them, in collaboration with Fritzsch, to develop another research paper in which they validated the model proposed before through the conduction of a questionnaire-based survey to 42 software professionals, through which they aimed to update the quality model including these new updates gained after concluding said survey [15].

B. Current State of ISO/IEC 25010

Since its last publication and release in 2011, the ISO/IEC 25010 document has, as of 2022, began an update process. This new draft version will include several aspects that might be related to cloud native applications, however, they are not strictly tied to CNAs.

On his blog post, Dr. Gernot Starke [22], mentions that "Published in 2011, the ISO 25010 standard on software product quality lacks pragmatism and practical applicability. Terms like scalability, deployability, energy efficiency, safety, or code quality are missing", and, in reference to the new 2022 draft, states that there is still some "polishing" to do.

Currently, as of October 2023, the ISO/IEC FDIS 25010 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Product quality model document, is under the approval stages, as seen in the ISO website [23].

C. Current Research vs. Related Work

The recent publication of research articles, as well as the recent re-evaluation of the ISO/IEC 25010 standard, suggests and indicates clearly the importance of an eventual restructuring of such Product Quality Model, to finally include key quality attributes present in modern software development.

Despite the fact of a new ISO/IEC 25010 Product Quality Model, as seen in table II, which might already include key quality attributes like scalability, flexibility, availability, amongst others, there is currently nothing available nor closely similar to both the ISO/IEC 25010 and ISO/IEC 25023 documents.

Functional Suitability
Functional Completeness
Functional Correctness
Functional Appropriateness
Performance Efficiency
Time Behaviour
Resource Utilization
Capacity
Compatibility
Co-Existence
Interoperability
Usability
Appropriateness Recognizability
Learnability
Operability
User Error Protection
User Engagement
User Assistance
Self-Descriptiveness
Reliability
Faultlessness
Availability
Fault Tolerance
Recoverability
Security
Confidentiality
Integrity
Non-Repudiation
Accountability
Authenticity
Resistance
Maintainability
Modularity
Reusability
Analysability
Modifiability
Testability
Flexibility
Adaptability
Scalability
Instalability
Replaceability
Safety
Operational Constraint
Risk Identification
Fail Safe
Hazard Warning
Safe Integration

TABLE II
SYSTEM/SOFTWARE PRODUCT QUALITY MODEL 2022 PROPOSAL

This investigation aims to contribute and eventually expand the area of knowledge of software development, providing a combination of both documents mentioned before, thus delivering an updated version of the Product Quality Model, as well as providing new characteristics, subcharacteristics, metrics and formulas to quantify each one of them in the proposed model.

Being a current focus point in terms of research and investigation, as cloud native applications and quality attributes related to them, are right now, reiterates the importance this topic currently has, thus reinforcing and validating the scope of this investigation, as well as providing justification for undergoing this exploratory and empirical work, which is ultimately aimed at providing a new way in which to evaluate quality attributes and characteristics of cloud native applications specifically.

IV. METHODOLOGY & PROPOSAL

The initial proposal to achieve a similar standard as that defined in both the ISO/IEC 25010 and the ISO/IEC 25023 documents, with the Product Quality Model definition, as well as the mathematical formulas to quantify each subcharacteristic of said model, in those two documents respectively, includes the identification and definition of the new quality attributes related to cloud native applications as an initial

step. Afterwards, identifying a way to validate and verify the existence of these attributes, as well as a way to quantify them and creating the respective documentation will be part of the methodology of this investigation.

A. Identification & Definition

After having identified the new quality attributes present in cloud native applications through the work of several authors and in their published books, as well as other important characteristics identified in the proposed quality model for cloud native applications in [14], it is a crucial part to re-list and define all these characteristics.

1) Elasticity: In physics, elasticity has been defined as "a material property capturing the capability of returning to its original state after a deformation" [17]. Related to software development, elasticity is defined as "the ability to acquire resources as you need them and release resources when you no longer need them" [24], basically, this being the ability to increase your resources and there is a significant increase in requests, traffic, or any other interaction with servers that require more resources, and as well the ability of going back to the original configuration once this peak is over.

The ability of being elastic can apply to many different resources, depending on the non-functional requirements of your software product, or on what type of traffic or request increase your product is meeting. For example, compute or processing capacity, CPU memory, RAM, I/O bandwidth, or even storage should be able to be elastic, depending again on configuration and requirements.

Following the ISO/IEC 25010 definition format, then, and in agreement with [17], elasticity, in software development, can be defined as the degree to which a system, product or component is able to adapt to workload changes by provisioning and de-provisioning resources in an automatic manner, such that at each point in time, the available resources match the current demand.

2) Loose Coupling: This concept has been part of software development in more recent years, appearing with different types of architectures such as microservices or event-driven architectures. Unlike classic software architectures, loose coupling presents several benefits, since it relies on the independence each component has from the others.

Loose coupling can be described as "an approach to interconnecting the components in a [...] software application so that those components [...] depend on each other to the least extent practicable" [25]. The implementation of such concept within a software product reduces risks during updates, helps isolate issues, simplifies testing, amongst others. Being a crucial concept in cloud computing, as it is as well of course of software development, the inclusion of loose coupling as part of the software product quality model is fundamental.

Following the ISO/IEC 25010 definition format, and taking into consideration the definition of the CNCF [26], loose coupling can be defined as the degree to which a system or product is made up of independently built components.

3) Observability: Being able to monitor a software product, to determine whether it is working as expected, or on any other case, use this information to take corrective measurements, is a crucial attribute to have. Without observability, a development team has no way of determining the state of a software product, and this, in turn, could be eventually tied to the Functional Suitability characteristic defined in the Product Quality Model in the ISO/IEC 25010.

Aspects like CPU usage time, memory, disk space, API response times, amongst other factors, can be monitored to allow a potential maintenance team or development team to achieve expected outcomes from this product.

Following the ISO/IEC 25010 definition format, and taking into consideration the definition provided by the CNCF [27], observability can be defined as the degree to which a system, product, or component can generate actionable insights, allowing users to understand its state from external outputs, and, if needed, take corrective action.

4) Resiliency: Withstanding or recovering from tough times and difficulties, is an ability many people wish they had, and, as strange as it may seem, software products - or at least their designers and developers - would like to have this characteristic too.

Software products are not exempt of failure, errors or malfunctioning. This can be due to several factors, including, but not limited to, code bugs, hardware or software failures, or insufficient resources.

According to [30], "a system is resilient if it continues to carry out its mission in the face of adversity". Despite how well the product might be designed and engineered, at some point, something, somewhere, can and most probably will malfunction, causing errors to the product; if the product is resilient, or has resiliency, it will keep on working despite this eventual errors.

Following the ISO/IEC 25010 definition format, and in agreement with [30], resiliency can be defined as the degree to which a system, product, or component can rapidly and effectively protect its critical capabilities or functionalities from disruption caused by adverse events and conditions.

5) Scalability: Similar to elasticity, but different in some ways, scalability is another crucial factor to consider when building cloud native applications. As mentioned above in section IV-A1, elasticity mainly refers to how fast a software product can scale both up and down, depending on demand. Scalability, on another hand, refers to how the system can handle increased loads of traffic, requests, or others [28].

According to [29], scalability can be defined as "the measure of a system's ability to increase or decrease in performance and cost in response to changes in application and system processing demands".

Therefore, following the ISO/IEC 25010 definition format, and taking as base the definition provided by the Gartner Glossary [29], scalability can be defined as the degree to which a system, product, or component, can increase or decrease in performance and cost, in response to changes in application and system processing demands.

B. Quantification, Validation & Verification

After having identified and defined the proposed quality attributes in the section above (section IV-A), they can now be analyzed to develop ways in which to quantify, validate and verify them. In order to do this, it is first necessary to validate the implementation, given that these quality attributes can be present in different ways within cloud native applications. Similar to the quality measures defined by the ISO/IEC 25023 in section II-C1, these new quality attributes will have a quality measure, following the format described in annex IX-B.

1) Elasticity: Referring back to the definition provided in section IV-A1, elasticity can be defined as "the degree to which a system, product or component is able to adapt to workload changes by provisioning and de-provisioning resources in an automatic manner, such that at each point in time, the available resources match the current demand". This being said, elasticity is a non-negotiable in cloud native application development, and these applications must be designed to be elastic. This basically means, again, that they can and must scale up or down according to changes in demand of resources.

According to [31], cloud native applications can achieve elasticity "through the use of auto-scaling and load-balancing techniques. Auto-scaling allows the application to automatically scale up or down based on predefined metrics, while load balancing distributes incoming traffic across multiple instances of the application". In summary, this statement defines how elasticity should be implemented on a cloud native application, and the key aspect of it is the automatization of elasticity. Despite the fact that applications can manually scale up or down, as well as can be manually routed to different instances according to incoming traffic, cloud native applications must be enabled to do all this automatically, without the requirement of someone monitoring the application server and turning on and off these options.

After defining the how in terms of implementation, it is important to quantify elasticity in a cloud native application, following the quality measure format defined in annex IX-B. The following quality measure was either directly taken from or developed from the following sources [17], [24], [28], [31]–[33].

- Elasticity measures: are used to assess the degree to which a system, product or component is able to adapt to workload changes by provisioning and de-provisioning resources in an automatic manner.
  - Elasticity Index (EEi-1-G): how quickly can a system, product or component scale up or down in an automatic manner, in response to changes in resource demands.

    $$X = \frac{A}{B}$$

    with A being the number of resources added or removed, and B being the time taken to add or remove resources, and where a higher elasticity index would indicate a more elastic system, product, or component.

2) Loose Coupling: Having defined loose coupling in section IV-A2 as the degree to which a system or product is made up of independently built components, this quality attribute is another non-negotiable in cloud native applications. Having a common background and coming from a mixture of a microservices pattern architecture, modularity, and even containerization, implementing loosely coupled services is crucial in cloud native applications, and can be done through the segregation of functionalities, and, according to [34], it can be implemented through several ways, including, but not limited to, the use of schemas, shared data for APIs, asynchronous communication, separated environments, amongst others.

Once again, having defined a way in how to implement loose coupling in a cloud native application, quality measures can be defined. The following quality measure has been taken, in part or in total, from the following references [25], [31], [35]–[38].

- Loose Coupling measures: are used to assess the degree to which a system or product is made up of independently built components.
  - Coupling Factor (LCf-1-G): how interconnected or interdependent are the components of a system or product.

    $$X = \frac{A}{B}$$

    with A being the number of inter-component dependencies, and B being the total number of components within the system or product, and where a lower coupling factor would indicate a more loosely coupled system or product.

3) Observability: As defined in section IV-A3, observability is the degree to which a system, product, or component can generate actionable insights, meaning that the degree to which the system, product, or component can automatically generate logs to show the state and functioning it is at. This logs can work as metrics to trigger different responses according to what is being measured, for example, increasing or decreasing processing power or total number of processing units depending on demand, through elasticity, or indicate errors in the functionality of different APIs, to mention a few.

According to [31], observability, in cloud native applications, is directly tied to a monitoring and measuring of both performance and health of an application in real time, and this can be achieved through monitoring, logging, and tracing tools. It is important to note that cloud providers have their own services to do this, for example, AWS's CloudWatch service. Now, despite a cloud native application being able to log something, it is necessary to monitor relevant information and to be able to quantify how "measureable" is the application in terms of monitoring. This can be done with the following quality measures, developed through the references found in [39]–[44].

- Observability measures: are used to assess the degree to which a system, product, or component can generate actionable insights.

- Log Coverage (OLc-1-G): what percentage of the system or product's states are covered by logs.

$$X = \frac{A}{B}$$

with A being the total number of components that are logged, and B being the total number of components within the system or product, where a higher log coverage would indicate a more observable system.
- Log Completeness (OLc-2-G): how complete are the logs generated by the system or product.

$$X = \frac{A}{B}$$

with A being the total number of states logged, and B being the total number of states, where a higher log completeness would indicate a more observable system.

4) Resiliency: Another non-negotiable quality attribute identified for cloud native applications is resiliency. Defined as the degree to which a system, product, or component can rapidly and effectively protect its critical capabilities or functionalities from disruption caused by adverse events and conditions in section IV-A4, it is important to design resilient applications when creating a CNA.

The ability of recovering from failures and continuing operation without interruption can be achieved through the use of distributed architectures and the implementation of fault-tolerant mechanisms, as [31] suggests. Distributed architectures will ensure the continuous operation of a system or product even if one or more components fail, while the use or implementation of fault-tolerant mechanisms will ensure the CNA can recover from potential failures without data loss or downtime.

In order to quantify resiliency in cloud native applications, the following quality measures were developed with input from the following references [45]–[50].

- Resiliency measures: are used to assess the degree to which a system, product, or component can rapidly and effectively protect its critical capabilities or functionalities from disruption.
  - Resiliency Ratio (RR-1-G): how well can a system or product recover from failures.

$$X = \frac{A}{B}$$

with A being the number of successful recoveries, and B being the total number of failures, where a higher resiliency ratio would indicate a more resilient system.
  - Mean Time to Recovery (RMr-2-G): what is the average time it takes for the system or product to recover from failures.

$$X = \sum_{i=1 \to n} (A_i)/n$$

where A_i is the time it takes to recover from the i-th failure, and n being the total number of failures, where a lower mean time to recovery would indicate a more resilient system.

5) Scalability: Last but certainly not least, scalability has been identified through this investigation as one final crucial non-negotiable in cloud native applications. Arguably the most important quality attribute of a CNA, scalability has been defined as the degree to which a system, product, or component, can increase or decrease in performance and cost, in response to changes in application and system processing demands in section IV-A5.

Once again, the automatization behind scalability in cloud native applications is determining, given that a system or product can manually scale after a stakeholder can enable this option. Nonetheless, similar to elasticity in section IV-B1, it has to be done automatically depending on changing demands on the application's resources.

According to [51], this scalability in cloud native applications can be achieved and implemented through several considerations, including but not limited to the implementation of horizontal scaling rather than vertical scaling[7], avoid defaulting to physical servers and rather taking advantage of cloud services and providers, avoiding unnecessary bottlenecks through caching, non-blocking I/O calls, load balancing and redundancy.

In a final way to quantify scalability in a cloud native application, the following quality measures have been developed through consideration of the following references [52]–[57].

- Scalability measures: are used to assess the degree to which a system, product, or component, can increase or decrease in performance and cost, in response to changes in application and system processing demands
  - Scalability Coefficient (SSc-1-G): how well can a system or product handle an increase in load, traffic, or requests.

$$X = \frac{A}{B}$$

with A being the top performance value at an increased load state, and B being the top performance value at a baseline load state, and where a higher scalability coefficient would indicate a more scalable system.
  - Resource Utilization Ratio (SRu-2-G): what amount of resources are utilized compared to the amount of resources allocated during scaling.

$$X = \frac{A}{B}$$

with A being the total number of resources being utilized, and B being the total number of resources allocated during scaling, and where a higher resource

[7]Vertical scaling refers to scaling by adding more resources, while Horizontal scaling refers to scaling by adding more nodes to a distributed network [51].

utilization ratio would indicate a more scalable system.

V. Case Study: How Cloud Native can an Application Really Be?

After having evaluating the different sources and literature available as seen in section II-E, and coming up with different quality attributes as well as the quality measure to quantify them respectively, a combination of the Product Quality Model (those chosen and described in section II-C1) and these new attributes can be done to evaluate CNAs. Are there applications out there that are "more cloud native" than others? In order to determine to what extent can software systems or products be truly cloud native, to a greater or lesser degree, it is necessary to apply these quality measures selected and evaluate the results, doing so in a case study.

Taking into consideration costs of hosting cloud native applications in a cloud provider, time to develop the application regardless of its functionalities, and other aspects of the software development cycle, an application has been arbitrarily selected, after considering different software products publicly available for academic use, as a case study to evaluate to what extent can an application really be cloud native, according to the quality attributes chosen. In this particular case study evaluation, the selected software product to test is the "Sock Shop"[8] application, created by 55 different contributors listed on the GitHub repository [59], owned by WeaveWorks and Container Solutions [60].

As mentioned on [60], "*Sock Shop* simulates the user-facing part of an e-commerce website that sells socks. It is intended to aid the demonstration and testing of microservice and cloud native technologies". In order to get valuable, real results, and despite having access to the complete source code, no code modifications will be done, this with the idea of not manually altering the results and doing an as-is software product evaluation with the quality attributes and respective quality measures selected.

The architecture of this e-commerce application is based on microservices, as the name suggests, and can be seen in detail on figure 1. In synthesis, this architecture is made up of six different microservices: Order, Payment, User, Catalogue, Cart, and Shipping. The Order microservice is written with a Java and .NET Core combination, Payment, User and Catalogue are written with Go, Cart is written in Java, all with a MongoDB database, and the Shipping microservice is written in Java. As well, there is a queue implemented using RabbitMQ, with a Queue-Master service written in Java.

According to [60], "the architecture of the demo microservices application was intentionally designed to provide as many microservices as possible [...]. [...] the microservices are roughly defined by the function in an e-commerce site. [...] All the services communicate using REST over HTTP."

[8]The complete source code for *Sock Shop* can be found at https://github.com/microservices-demo/microservices-demo, while its complete documentation can be directly found at https://microservices-demo.github.io/.

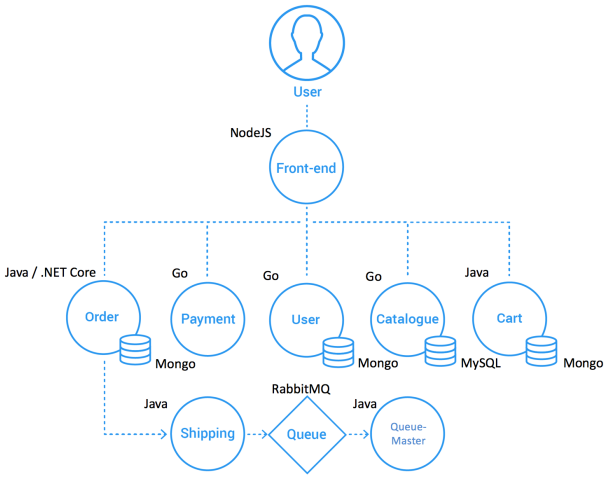

Fig. 1. Sock Shop Architecture [59]

Each microservice has its own API implementation using an API Gateway pattern, with different endpoints respectively.

A. Testing Environment and APIs

To test this *Sock Shop* application, an Amazon ECS container was created, and the application was provisioned as described in the documentation [61]. Other hosting possibilities, listed as well on the repository internal documentation, are a local machine, Kubernetes, Docker, Apcera, among others, however, due to familiarity, and following the cloud native application idea, it was hosted on AWS on ECS using the Docker images provided on the code repository. It is important to clarify that, given that the microservices are handled as Docker images, the resources allocated for each microservice cannot be determined individually.

As far as the APIs included in this application goes, there are several endpoints on the services listed above. Given the extent of the application, the services to be tested were arbitrarily chosen, and, taking into consideration that each service runs as its own, independent microservice, they can be considered a single responsibility application. These APIs chosen are: Catalogue, Orders, and Payment. They were chosen based on the fact that an e-commerce application relies mostly on the catalogue of products it will advertise to users, since if there are no products shown, there is nothing to sell. As well, in order to purchase products, users must be able to place orders and pay for their items, so these three APIs and their endpoints are thought to be crucial in any e-commerce application.

Note that the base URL for each endpoint will vary depending on where the application is hosted. For the sake of example, the localhost URL will be implemented to display the different endpoints, despite the fact that for testing, the application was hosted on AWS. The different services contain the following endpoints, as seen on table III:

Catalogue Service
[GET] http://127.0.0.1/catalogue/ [GET] http://127.0.0.1/catalogue/{id}/ [GET] http://127.0.0.1/catalogue/size/ [GET] http://127.0.0.1/tags/
Payment Service
[GET] http://127.0.0.1/health/ [POST] http://127.0.0.1/paymentAuth/
Orders Service
[GET] http://127.0.0.1/orders/ [POST] http://127.0.0.1/orders/

TABLE III
ENDPOINTS OF SERVICES CHOSEN FOR TESTING

VI. RESULTS

In order to test and simulate traffic to the *Sock Shop* website, the documentation includes a testing section where scripts are provided which will hit the different endpoints and APIs selected, associated to those microservices chosen. To execute these tests, they provide the generic Docker command, which must be updated depending on where the front-end is running, meaning IP address and port. Once again, for illustration purposes, the localhost address will be used. The command to be executed is the following:

```
docker run --net=host weaveworksdemos/
   load-test -h localhost
```

After testing, through the triggering of different HTTP requests of the different endpoints listed on table III, the following quality measures[9] were considered, according to the scope of the investigation:

- Performance Efficiency Quality Measures
  - Mean Response Time
  - Response Time Adequacy
  - Mean Turnaround Time
  - Turnaround Time
  - Mean Throughput
  - Mean Processor Utilization
  - Mean Memory Utilization
  - Mean I/O Devices Utilization
  - Bandwidth Utilization
  - Transaction Processing Capacity
  - User Access Capacity
  - User Access Increase Adequacy
- Reliability Quality Measures
  - System Availability
  - Mean Down Time
  - Failure Avoidance
  - Redundancy of Components
  - Mean Fault Notification Time
  - Mean Recovery Time
  - Backup Data Completeness
- Maintainability Quality Measures
  - Coupling of Components
  - Cyclomatic Complexity Adequacy
  - Reusability of Assets
  - Coding Rules Conformity
  - System Log Completeness
  - Diagnosis Function Effectiveness
  - Diagnosis Function Sufficiency
  - Modification Efficiency
  - Modification Correctness
  - Modification Capability
  - Test Function Completeness
  - Autonomous Testability
  - Test Restartability
- Portability Quality Measures
  - Hardware Environmental Adaptability
  - System Software Environmental Adaptability
  - Operational Environment Adaptability
- Elasticity Quality Measures
  - Elasticity Index
- Loose Coupling Quality Measures
  - Coupling Factor
- Observability Quality Measures
  - Log Coverage
  - Log Completeness
- Resiliency Quality Measures
  - Resiliency Ratio
  - Mean Time to Recovery
- Scalability Quality Measures
  - Scalability Coefficient
  - Resource Utilization Ratio

Apart from evaluating the case study, the final results of this evaluation, an "ISO/IEC 25000++ Series[10]" standard, will be also described in this section.

A. Sock Shop Case Study

After evaluating the tests that were provided by the same developers, different results were obtained. The command provided has four different flags that can be passed to the execution, and alter some parameters of the tests. The $-h$ flag sets the host to run the tests against, the $-c$ flag sets the amount of concurrent clients hitting the different endpoints, or

[9]The complete list, definition, and mathematical formula for their respective quantification can be found on section II-C for the ISO/IEC 25010 defined quality attributes, and those proposed in this investigation can be seen in full detail on section IV.

[10]This is not an official name, but rather simply a reference to the C/C++ naming idea.

at least simulates this; it is defaulted to 2 concurrent users. The $-r$ flag sets the total amount of requests made, and is defaulted to 100 requests. Finally, the $-d$ flag specifies a delay before hitting the endpoints.

Since the main focus of this investigation is not to validate the existing ISO/IEC 25000 criteria, but rather to test the proposed additional criteria, individual tests were not carried out, but rather took the results from the script given to users by the developers. Different combinations of users and requests were made, with the average one being the following command, with 500 requests and 100 different concurrent users.

```
docker run --net=host weaveworksdemos/
   load-test -h localhost -r 500 -c 100
```

The previous command will generate random users, with different names, and start hitting the general application with requests. Overall, the average results for the selected endpoints were the following:

Catalogue Service
Average request response time: 3.148ms
Payment Service
Average request response time: 3.720ms
Orders Service
Average request response time: 2.367ms

TABLE IV
AVERAGE RESPONSE TIMES OF SERVICES CHOSEN FOR TESTING

This information can be visualized through another service provided by the developers in charge of Sock Shop, through the following command, used to validate container health, and monitor the overall requests:

```
docker-compose -f ./deploy/docker
   -compose/docker-
   compose.monitoring.yml up -d
```

The Prometheus service, and Graphana service, can be accessed locally at the following addresses respectively:

```
Prometheus http://localhost:9090
Grapaha http://localhost:3000
```

Overall, the general ISO/IEC 25000 metrics could show that the application works, and has relatively good response times, can handle concurrent users and also simultaneous requests. However, these metrics can all by applied to any type of application, not necessarily to cloud native applications only.

The additional metrics, proposed in this investigation, apply specially to cloud native applications, and can be seen in this case study.

In terms of the fist quality measure, elasticity, the elasticity index for the Sock Shop application shows the following results. Since there is no option to dynamically add or remove resources depending on requests, but rather uses defined resources, regardless of were is the application running. For this elasticity quality measure, Sock Shop could either be a non-applicable case, or rather score a zero, since it has no elasticity whatsoever. It is important to point out, nonetheless, that elasticity attribute could be provided to the application through a manual setup, thus complying with this criteria. However, this manual implementation goes against that automation wanted in cloud native applications.

The next proposed quality measure is loose coupling, which, since the application was developed using microservices, should be a good scoring metric. The proposed quantification method for loose coupling was the following $X = \frac{A}{B}$, with A being the number of inter-component dependencies, and B the total number of components. In this case, there are a total of 15 different components within the entire Sock Shop Application, with the only dependencies being the services tied directly to their database, thus having 4 different inter-component dependencies. As a result, this gives a coupling factor of 0.267, and based on the definition for this quality measure, it would suggest a good loose coupling.

In terms of observability, there are two different quality measurements proposed. The first, log coverage, determines the total number of components that are logged or registered against the total number of components, were a result closer to 1 determines a better result. Sock Shop handles this pretty well, as all services have their respective logs, therefore giving a log coverage result of 1. The second quality measurement is log completeness, logging not the components themselves but rather their states. In this particular scenario, the services of Sock Shop, being Docker containers, are tied to the visibility given by Docker, so Sock Shop once again scores a perfect 1, since the container status of each microservice is always logged.

Next, resiliency is once again a rather complicated measure to test out, without having overall control of the application, but rather only the Docker images to create it. The first quality measurement is the resiliency ratio, but with the inability of causing the application to fail and recover, this cannot be tested out. Consequently, the second quality measurement, mean time to recovery, cannot be tested out either.

Finally, the last quality measure proposed is scalability, where two different quality measurements were proposed too: first, the scalability coefficient, and second the resource utilization ratio. The first measurement is tied to how well can an application handle an increase in load, traffic, or requests.

The scalability coefficient takes the performance of the application at an increased load state, and the top performance at a baseline load state. For the baseline load state, the default values were used, with 10 requests and 2 concurrent users. For the increased load state, without really knowing when will the application break, become slow, or require to scale, this was more of a trial and error situation, which ended up in using $1,000,000$ requests with $100,000$ users, trying to simulate a relatively busy e-commerce site; these numbers might be far off from a company like Amazon, or any other e-commerce giant, but try to simulate a small to mid size e-commerce site.

As a result, this ended up giving an average top performance

value at a baseline load state of 25.782ms, and an average top performance value at an increased load state of 22.497ms. This gives a result of 0.873, which indicates that the application is capable of handling such loads, but overall performance starts to decrease as a result of increased requests, users or traffic. Unfortunately, for the resource utilization ratio, since there is no visibility over the resource allocation or utilization, this quality measurement cannot be calculated for this Sock Shop case study.

Overall, results show that Sock Shop was well designed, being able to handle increased loads of requests, concurrent users and traffic, thus suggesting it is a scalable system. The microservices have observability and can log their state, as well as register any information derived from requests, again indicating an observable system. As far as loose coupling goes, the proposed architecture used a microservices approach, this tied to the idea that cloud native applications must use a microservices architecture approach, indicates that Sock Shop is a loosely coupled system. Finally, as far as elasticity goes, unfortunately, this case study does not provide the required information to actually determine if it is an elastic system or not. Nonetheless, with the intention of determining the current state of the application, without intervening the code or similar, the results are as realistic as possible.

B. Cloud Native Application ISO/IEC 25000++ Series

After a thorough investigation of reliable sources of information, as defined in section II-E, the proposed quality measures to a potential ISO/IEC 25000++ Series, with their respective quality measurements, are the following:

Elasticity
Elasticity Index
Loose Coupling
Coupling Factor
Observability
Log Coverage
Log Completeness
Resiliency
Resiliency Ratio
Mean Time to Recovery
Scalability
Scalability Coefficient
Resource Utilization Ratio

TABLE V
PROPOSED QUALITY MEASURES AND THEIR RESPECTIVE QUALITY MEASUREMENTS

The proposal of these quality measures, their quality measurements and ways to quantify, validate and verify them, will lead to a potential ISO/IEC 25000 standard series that takes into consideration new quality attributes and characteristics that have come into existence since 2005, when the standard was first published. All these new concepts not only show how technology has advanced, but also how standards, in some cases, have failed to keep on evolving with technology.

Given that there is a new ISO/IEC 25010 document being written and evaluated at the time of this investigation, shows how relevant the idea of adding new quality attributes, or reorganizing the current software product quality model is right now, thus validating our investigation too. In addition to a new draft being worked on, different books, articles and research papers evaluated as literature reinforces too this importance, not necessarily tied only to cloud native applications, but to new software product quality attributes in general.

The identification, definition, quantification, validation and verification of software product quality attributes, analyzed in section IV, is of great value and relevance to this ongoing discussion of updating the ISO/IEC 25000 Standards Series, since it has taken into account various different literature sources, and adapted them into an ISO/IEC 25000 format, providing a starting point for this new updated version.

VII. CONCLUSIONS

The results obtained, from both the investigation conducted, and the applied quality attributes proposed, to the Sock Shop case study, show the relevance of this topic in today's software development and software engineering communities. The necessity of architecting and building software applications and programs through a microservices, serverless, event-driven, or any other architectonic pattern, had led to new quality attributes that did not exist back in the days where monolithic architectures were the dominant style.

Consequently, the growth not only of applications, but their needs and capabilities of supporting large number of concurrent users, high traffic loads, and high processing power, to mention a few, has led to the need of scalable, elastic, and resilient software applications. Cloud native has been introduced as a solution, not necessarily the only one, but as a definitely good approach in solving these new requirements.

Our investigation shows the relevance of updating and upgrading software product quality attributes in new, current software applications, and how this can be taken from theory into practice through the implementation of this guideline in the Sock Shop case study.

The application of this guideline, despite Sock Shop being a software application designed for academic purposes, shows how key information can be obtained to model and potentially refactor applications according to business requirements and real life situation needs. This proposed standard can help to determine if the software application, regardless of the business area it will help, will suffice to satisfy those needs, or if it will require a more elastic, or more scalable, or a more loosely coupled architecture, to mention a few software quality attributes identified through our research.

The different limitations and restrictions of the ISO/IEC 25000 Standard Series, as described by [22], suggest that a change must be done, not only to any potential new additions in terms of quality attributes and/or measurements, but also to the existing ones.

As described in [23], the new version of the Product Quality Model, ISO/IEC 25010:2023 is already on the way, at least to the day of inquiry. This means that new quality attributes will already be taken into consideration, as suggested, at least in theory, in this investigation, though the quality attributes and quality measurements will not be the same, at least considering the new model described in [22]. Some of those quality measures, however, such as scalability, have been taken into account and will be included in this new version of the Software Product Quality Model. Nonetheless, other quality attributes, such as elasticity, loose coupling, or resiliency, which were considered relevant in our investigation, are not even mentioned or added in the new draft.

Overall, there is a clear necessity for a new version of this Product Quality Model, but believing it will take into account every quality attribute that has come to surface in the last 20 years is unreal. Software development has come a long way, new programming languages have been developed, new frameworks have been developed, new architectures, new concepts, and many different things have been developed over the past 20 years. But, what has happened to standards, guides, and assessment criteria?

At least from an ISO/IEC 25000 perspective, it has been left behind in comparison to those developments. On the same hand, the fact that there is a new draft of this model on the making, shows the crucial necessity for updating these guidelines and standards.

Cloud native applications will continue to be present in software development. Their low cost benefits, their SaaS approaches, and every other benefit provided to development teams and organizations, regardless of the cloud provider chosen, makes them attractive to the software development world. This being said, is there a clear necessity for a quality assessment guideline? The true answer is, it could depend. A small, family owned business, for example, might not be as thorough and strict with software quality as big enterprises, but these standards and guidelines do not target small applications but rather large applications that deal with thousands, or even millions of users and therefore must be scalable, must be elastic, must provide good performance levels at both baseline loads and increased loads; in summary, must be compliant with an ISO/IEC 25000 quality level.

Can an application be more cloud native than others? Certainly, yes. An application might not even be cloud native at all, since business requirements will vary and depend on the project's nature itself. Regardless, with the current dominance cloud native applications have on the market, the overall question of the need of an updated and refactored ISO/IEC 25000 Standard Series version could have been answered, and leads to a new idea: if the ISO/IEC 25000 Standard Series is, to a lesser or greater extended, outdated, what is the current situation of companies, organizations, or event governmental institutions that implement these kinds of standards, or even those that do not follow, to any extent, these same software product quality standards?

VIII. Future Work

Once the new version of the ISO/IEC 25010 Product Quality Model has been published, a revision of it must be done to validate if cloud native application quality attributes have been taken into consideration, and if so, to what extent. This will determine if a true, significant contribution to the standard was made, or if it still has some limitations and shortcomings.

On a separate note, and circling back to the Product vs. Process idea described on section II-D, a potential new research could take place, leading to a contribution where a guideline can be made. Through this guideline, a new way of defining and describing requirements, at both a functional and non-functional way, can conduct to ensuring the proper implementation of all the quality attributes described both in the ISO/IEC 25000 series, at least at its current version, and also those proposed in this investigation.

References

[1] *"Cloud Native Definition"*. Cloud Native Computing Foundation. https://www.cncf.io/about/who-we-are/ (accessed Oct. 1, 2023).

[2] *"What is Cloud Native?"*. AWS. https://aws.amazon.com/what-is/cloud-native/ (accessed Oct. 1, 2023).

[3] *"What is Cloud Native?"*. OCI. https://www.oracle.com/cloud/cloud-native/what-is-cloud-native/ (accessed Oct. 1, 2023).

[4] *"Microservices"*. AWS. https://aws.amazon.com/microservices/ (accessed Oct. 2, 2023)

[5] *"Building Applications with Serverless Architectures"*. AWS. https://aws.amazon.com/lambda/serverless-architectures-learn-more/ (accessed Oct. 2, 2023)

[6] *"Understanding cloud-native applications"*. RedHat. https://www.redhat.com/en/topics/cloud-native-apps (accessed Oct. 1, 2023).

[7] *"The ISO/IEC 25000 series of standards"*. ISO 25000. https://iso25000.com/index.php/en/iso-25000-standards (accessed Sep. 27, 2023).

[8] *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE*, ISO/IEC 25000:2014, 2014.

[9] *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*, ISO/IEC 25010:2011, 2011.

[10] *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Measurement reference model and guide*, ISO/IEC 25020:2007, 2007.

[11] *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Quality measure elements*, ISO/IEC 25021:2012, 2012.

[12] *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Measurement of system and software product quality*, ISO/IEC 25023:2016, 2016.

[13] *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Evaluation process*, ISO/IEC 25040:2011, 2011.

[14] R. Lichtenthäler, and G. Wirtz. "Towards a Quality Model for Cloud-native Applications", *In: Montesi, F., Papadopoulos, G.A., Zimmermann, W. (eds) Service-Oriented and Cloud Computing. ESOCC 2022. Lecture Notes in Computer Science*, vol. 13226, pp. 109-117, Apr. 2022. doi: 10.1007/978-3-031-04718-3_7. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-04718-3_7

[15] R. Lichtenthäler, J. Fritzsch, and G. Wirtz. "Cloud-Native Architectural Characteristics and their Impacts on Software Quality: A Validation Survey", *2023 IEEE Conference on Service-Oriented System Engineering (SOSE)*, Athens, Greece, 2023, pp. 9-18, doi: 10.1109/SOSE58276.2023.00008. [Online]. Available: https://ieeexplore.ieee.org/document/10254764

[16] K. Durr, and R. Lichtenthäler. "An Evaluation of Modeling Options for Cloud-Native Application Architectures to Enable Quality Investigations", *In: 2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*, Vancouver, WA, USA, 2022, pp. 297-304, doi: 10.1109/UCC56403.2022.00053. [Online]. Available: https://www.computer.org/csdl/proceedings-article/ucc/2022/608700a297/1LvAcBXTLEY

[17] N. Herbst, S. Kounev, and R. Reussner. "Elasticity in Cloud Computing: What It Is, and What It Is Not", *In: Proceedings of the 10th International Conference on Autonomic Computing (ICAC 2013)*, San Jose, CA, USA, 2013. [Online]. Available: https://sdq.kastel.kit.edu/publications/pdfs/HeKoRe2013-ICAC-Elasticity.pdf

[18] R. Lichtenthäler, and G. Wirtz. "A Review of Approaches for Quality Model Validations in the Context of Cloud-native Applications", *In: J. Manner, D. Lübke, S. Haarmann, S. Kolb, N. Herzberg, O. Kopp (eds) 14th ZEUS Workshop, ZEUS2022*, Bamberg, Germany, 2022. [Online]. Available: https://ceur-ws.org/Vol-3113/paper6.pdf

[19] B. Scholl, T. Swanson, and P. Jausovec. *Cloud Native: Using Containers, Functions, and Data to Build Next-Generation Applications*, 1st ed. Sebastopol, CA: O'Reilly Media Inc, 2019.

[20] C. Davis. *Cloud Native Patterns: Designing change-tolerant software*, 1st ed. Shelter Island, NY: Manning Publications Co, 2019.

[21] K. Indrasiri, and S. Suhothayan. *Design Patterns for Cloud Native Applications: Patterns in Practice Using APIs, Data, Events, and Streams*, 1st ed. Sebastopol, CA: O'Reilly Media Inc, 2021.

[22] G. Starke. *"Shortcomings of ISO 25010"*. innoq.com. https://www.innoq.com/en/articles/2023/02/iso-25010-shortcomings/ (accessed Oct. 16, 2023).

[23] ISO. *"ISO/IEC FDIS 25010 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Product quality model"*, iso.org. https://www.iso.org/standard/78176.html (accessed Oct. 16, 2023).

[24] AWS. *"Elastic"*. docs.aws.amazon.com. https://docs.aws.amazon.com/whitepapers/latest/reactive-systems-on-aws/elastic.html (accessed Oct. 23, 2023).

[25] P. Kirvan. *"Loose Coupling"*. techtarget.com. https://www.techtarget.com/searchnetworking/definition/loose-coupling (accessed Oct. 23, 2023).

[26] CNCF. *"Loosely Coupled Architecture"*. glossary.cncf.io. https://glossary.cncf.io/loosely-coupled-architecture/ (accessed Oct. 23, 2023).

[27] CNCF. *"Observability"*. glossary.cncf.io. https://glossary.cncf.io/observability/ (accessed Oct. 23, 2023).

[28] N. Peleg. *"Elasticity vs. Scalability AWS"*. cloudrice.co.il. https://www.cloudride.co.il/blog/elasticity-vs.-scalability-aws (accessed Oct. 23, 2023).

[29] Gartner Glossary. *"Scalability"*. gartner.com. https://www.gartner.com/en/information-technology/glossary/scalability (accessed Oct. 23, 2023).

[30] D. Firesmith. *"System Resilience: What Exactly is it?"*. insights.sei.cmu.edu. https://insights.sei.cmu.edu/blog/system-resilience-what-exactly-is-it/ (accessed Oct. 23, 2023).

[31] Noor. *"6 Key Features Of A Cloud-Native Applications"*. techbullion.com. https://techbullion.com/6-key-features-of-a-cloud-native-applications/ (accessed Nov. 7, 2023).

[32] T. Telang. "Cloud-Native Application Development", *In: Beginning Cloud Native Development with MicroProfile, Jakarta EE, and Kubernetes*, Berkeley, CA, 2022, doi: 10.1007/978-1-4842-8832-0_2. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4842-8832-0_2.

[33] W. Ai, K. Li, S. Lan, F. Zhang, J. Mei, K. Li, and R. Buyya. "On Elasticity Measurement in Cloud Computing", *In: Scientific Programming*, vol 2016 doi: 10.1155/2016/7519507. [Online]. Available: https://www.hindawi.com/journals/sp/2016/7519507/.

[34] J. Simpson. *"How to Design Loosely Coupled Microservices"*. nordicapis.com. https://nordicapis.com/how-to-design-loosely-coupled-microservices/ (accessed Nov. 7, 2023).

[35] J. Wang. *"Cloud Computing and Cloud Native Systems Lecture 18 Loose Coupling"*. [Online]. Available: http://www.ece.iit.edu/ jwang/ece473-2023f/ece473-lec18.pdf

[36] S. Kaushik. *"Microservices - What Is Loose Coupling?"*. alibabacloud.com. https://www.alibabacloud.com/blog/microservices—what-is-loose-coupling_597007 (accessed Nov. 7, 2023).

[37] C. Feling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter. *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*, 1st ed. Vienna, Austria: Springer Vienna, 2014.

[38] C. Richardson. *"Designing loosely coupled services"*. microservices.io. https://microservices.io/post/microservices/2020/12/14/designing-loosely-coupled-services.html (accessed Nov. 7, 2023).

[39] J. Kosinska, B. Balis, M. Konieczny, M. Malawski and S. Zielinski, "Toward the Observability of Cloud-Native Applications: The Overview of the State-of-the-Art," *In IEEE Access*, vol. 11, pp. 73036-73052, 2023, doi: 10.1109/ACCESS.2023.3281860. [Online]. Available: https://ieeexplore.ieee.org/document/10141603

[40] S. Gittlen. *"Observability is key to cloud-native transitions and modern application development"*. about.gitlab.com. https://about.gitlab.com/blog/2022/04/05/observability-is-key-to-cloud-native-transitions-and-modern-application-development/ (accessed Nov. 7, 2023).

[41] R. Shah. *"Observability for Cloud-Native Applications"*. codeburst.io. https://codeburst.io/observability-for-cloud-native-applications-cd7cf866514e (accessed Nov. 7, 2023).

[42] A. Rodriguez. *"Take a quantitiave approach to cloud application architecture"*. ibm.com https://www.ibm.com/cloud/architecture/architecture/practices/quantitative-approach-to-cloud-app-architecture/ (accessed Nov. 7, 2023).

[43] B. Doerrfeld. *"The State of Cloud-Native Observability Tools"*. cloudnativenow.com. https://cloudnativenow.com/features/the-state-of-cloud-native-observability-tools/ (accessed Nov. 7, 2023).

[44] B. Gangapadhyay. *"Observability in Cloud-Native Applications"*. [Online]. Available: https://www.happiestminds.com/wp-content/uploads/2020/11/Observability-in-Cloud-Native-Applications.pdf.

[45] S. Barve, and R. Shinde. *"A Four-Step Approach to Verifying the Resiliency of Cloud-Native Applications"*. ibm.com. https://www.ibm.com/blog/a-four-step-approach-to-verifying-the-resiliency-of-cloud-native-applications/ (accessed Nov. 8, 2023).

[46] Nix United. *"Cloud Native Architecture - Basics You Need To Know"*. nix-united.com. https://nix-united.com/blog/cloud-native-architecture-basics-you-need-to-know/ (accessed Nov. 8, 2023).

[47] R. Vettor, D. Pine, D. Coulter, M. Wenzel, and S. Smith. *"Cloud-native resiliency"*. learn.microsoft.com. https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/resiliency (accessed Nov. 8, 2023).

[48] Google Cloud.*"Google Cloud Architecture Framework: Reliability"*. cloud.google.com. https://cloud.google.com/architecture/framework/reliability (accessed Nov. 8, 2023).

[49] L. Atchison. *"Do Cloud-Native Architectures Make Apps More Reliable?"*. cloudnativenow.com. https://cloudnativenow.com/features/do-cloud-native-architectures-make-apps-more-reliable/ (accessed Nov. 8, 2023).

[50] N. Gerne, A. Dundurao, R. J. Jafarkhani, and F. Valles. *"The new era of resiliency in the cloud"*. mckinsey.com. https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-new-era-of-resiliency-in-the-cloud (accessed Nov. 8, 2023).

[51] Concepta. *"The Dos and Don'ts of Scalable Architecture"*. conceptatech.com. https://www.conceptatech.com/blog/dos-donts-designing-scalable-architecture (accessed Nov. 8, 2023).

[52] A. Al-Said, and P. Andras. Scalability analysis comparisons of cloud-based software services, *"In: Journal of Cloud Computing"*, vol 8, 2019, doi: 10.1186/s13677-019-0134-y. [Online]. Available: https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-019-0134-y.

[53] M. Bhandaru. *"Cloud-Native AI Workloads: Scalability, Sustainability and Security"*. cloudnativenow.com. https://cloudnativenow.com/features/cloud-native-ai-workloads-scalability-sustainability-and-security/ (accessed Nov. 8, 2023).

[54] R. Fellows, and M. Rabin. Massively Scalable Cloud Storage for Cloud Native Applications. 2019. [Online]. Available: https://www.evaluatorgroup.com/wp-content/uploads/2020/09/Cloud_Native_Application_Storage_final3.pdf.

[55] VMWare. *"What is Cloud Scalability?"*. vmware.com. https://www.vmware.com/topics/glossary/content/cloud-scalability.html (accessed Nov. 8, 2023).

[56] IBM. *"Nonfunctional requirements: A checklist"*. ibm.com. https://www.ibm.com/cloud/architecture/architecture/practices/nonfunctional-requirements-checklist (accessed Nov. 8, 2023).
[57] M. Abbott, and M. Fisher. *The Art of Scalability - Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise*, 1st ed. Boston, MA: Pearson Education Inc, 2009.
[58] *"The Twelve-Factor App"*. 12factor.net. https://12factor.net/ (accessed Nov. 24, 2023).
[59] *"microservices-demo"*. github.com. https://github.com/microservices-demo/microservices-demo (accessed Nov. 20, 2023).
[60] *"Sock Shop"*. github.io. https://microservices-demo.github.io/ (accessed Nov. 20, 2023).
[61] *"Sock Shop"*. github.io. https://microservices-demo.github.io/docs/index.html (accessed Nov. 20, 2023).
[62] *"Amazon EC2"*. aws.amazon.com. https://aws.amazon.com/ec2/instance-types/ (accessed Nov. 20, 2023).

IX. Annex

A. Table Format of QMEs

The following information is taken textually from the ISO/IEC 25021 document, and includes the required details on how to properly "define and/or design a QME to provide necessary or helpful information" [11].

- QME Name: A QME should have a unique name and should be identified with a serial number, if necessary. Most of the time it begins with "number of... (ratio scale)".
- Target entity: A QME shall have a target object that is to be characterized by measuring its property. Target entity should be a work product or behavior of a system, software, or stakeholders such as users, operators, developers, testers, or maintainers.
- Objectives and property to quantify: Identification of a property to quantify is usually related to the name of the QME. Selected property to quantify should be the one which is most relevant to the measurement of information needed. [...].
- Relevant quality measure(s): Reference to specific quality measure(s) which use this QME shall be specified.
- Measurement method: Measurement method explains how to collect data and how to transform it to a value quantifying the property through a numerical rule.
- List of sub properties related to the property to quantify (optional): An identified property to quantify can be related to different sub propertied, if necessary. This relationship between properties should be expressed as a schema or a formula. This constitutes the measurement method model.
- Definition of each sub property (optional): If there is a list of sub properties, each sub property should be defined.
- Input for the QME: The input shall be described in enough detail to identify what quantitative information is used to measure the QME. Any sources providing the input should also be identified such as the documented work products, behavior of systems and software, or human behavior of users, operators, developers, testers, or maintainers.
- Unit of measurement for the QME: The unit of measurement and, if appropriate, the formula used.
- Numerical rules: A numerical assignment rule shall be described from a practitioner view (generally a text form) or from a theoretical point of view (generally a mathematical expression).
- Scale type: Scale type shall be identified. [...] could be nominal, ordinary, interval or ratio.
- Context of QME: This gives information about the intended use of the measurement results.
- Software life cycle process(es): The typical appropriate life cycle process(es) that are suited for actual measurement of this QME with respect to a target entity should be identified here
- Measurement constraints (optional): If necessary, any constraints related to the measurement method should be described.

B. Format Used for Documenting the Quality Measures

The following information is taken textually from the ISO/IEC 25023 document [12], and provides the required details to properly define a Quality Measure.

- 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 subcharacteristics as one capital X followed by lowercase x (for example, "PTb" denotes "Time behaviour" measures for "Performance efficiency");
  - serial number of sequential order within quality subcharacteristic;
  - 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;
- Name: quality measure name;
- Description: the information provided by the quality measure;
- Measurement function: mathematical formula showing how the quality measure elements are combined to produce the quality measure.