Evaluation of Non-functional Requirements: Case study of the AHFE Open Access System

André Luís P. Vasconcelos Jr.[1]

[1]Centro de Informática (CIn) – Universidade Federal de Pernambuco (UFPE)
Caixa Postal 7851 – 50732-970 – Recife – PE – Brazil

alpvj@cin.ufpe.br

Abstract. *This article presents a comprehensive quality assessment of the AHFE Open Access system using an ISO/IEC 9126-1 derived quality model and the IQMC method. We analyze core non-functional attributes, including search usability, page load performance, codebase quality, and responsiveness. Through a systematic case study, we reveal insights into the system's strengths, limitations, and opportunities for improvement, providing a valuable guide for software practitioners. Our findings underscore the system's competitive search usability, while identifying optimization potential in page load times, code modularity, and test coverage. Notably, AHFE Open Access showcases rapid recovery times and consistent responsiveness. This study informs future enhancements, contributing to a higher quality and user-centric evolution of the system.*

Resumo. *Este artigo apresenta uma avaliação da qualidade do sistema AHFE Open Access usando um modelo de qualidade derivado da ISO/IEC 9126-1 e o método IQMC. Analisamos atributos não funcionais essenciais, incluindo usabilidade de busca, desempenho de carregamento de páginas, qualidade do código e responsividade de tela. Através de um estudo de caso sistemático, revelamos insights sobre as forças, limitações e oportunidades de melhoria do sistema, fornecendo um guia para profissionais da área. Nossas descobertas destacam a usabilidade de busca competitiva do sistema, identificando potencial de otimização em tempos de carregamento de páginas, modularidade do código e cobertura de testes. O AHFE Open Access demonstra tempos de recuperação rápidos e responsividade consistente. Este estudo informa melhorias futuras, contribuindo para uma evolução com maior qualidade e foco no usuário.*

1. Introduction

Non-functional Requirements (NFRs) have gained significant attention due to their high impact on modern software systems, influencing both their quality and user satisfaction [1]. While the Functional Requirements (FRs) define what the software does, it is the NFR that determines how the software behaves and the characteristics it possesses.

In today's digital landscape, the success of software systems relies not only on their core functionalities (FRs), but also on their ability to meet non-functional requirements. These requirements encompass crucial aspects such as performance, security, usability, and maintainability, play a critical role in determining the overall quality and user experience of a system. In this article, we delve into the evaluation of non-functional requirements through an insightful case study of the Applied Human Factors and Ergonomics (AHFE) Open Access System.

The AHFE Open Access serves as a prime example of a software system that caters to a diverse user base, offering access to a large range of scholarly resources, e.g. books and

articles, in a seamless and user-friendly manner. As the system is already in use, in this study,we aim to shed light on the evaluation process of non-functional requirements and the significance of incorporating them into the software evolution lifecycle.

Throughout this article, we explore methodologies and best practices for assessing the non-functional aspects of the AHFE Open Access System. Guided by the principles outlined by Robert K. Yin [2], we conduct a single case study. We also address the challenges that arose during NFR evaluation and share the strategies we employed to overcome them. Notably, we utilized system mocking to perform production-like tests without disruption and established internal routes to evaluate specific behaviors. By dissecting this real-world case study, we provide valuable insights into the practical application of non-functional requirements evaluation in web-based systems. This, in turn, empowers the enhancement of software performance, security, and overall quality.

By the end, as we uncover the intricacies of evaluating how non-functional requirements are being satisfied by the AHFE Open Access System, we aim to achieve three overarching goals: firstly, to construct a robust quality model tailored for a real-world, in-use system; secondly, to illustrate how the evaluation of Non-Functional Requirements (NFRs) can seamlessly integrate into the software development lifecycle, amplifying the system's overall excellence; and thirdly, to provide a guide for individuals seeking to replicate this process for their own software systems. By sharing our insights, we empower the community with the knowledge and tools to not only elevate software performance, security, and quality, but also to forge a user-centric approach that underpins the modern software landscape.

2. Background

In this section we provide a context for this paper giving a brief description of four main streamlines that support the development of our work: FRs/NFRs, ISO/IEC 9126-1, MCCall's quality model in SWE and the IQMC method. And we also provide an overview of our object of study: the AHFE Open Access System.

2.1. Functional and Non-functional Requirements (FRs/NFRs)

Requirements Engineering plays a pivotal role in the software development lifecycle by capturing and defining what a software system needs to accomplish to satisfy stakeholders requirements [3]. This process involves understanding stakeholder needs, translating them into specific requirements, and ensuring that the resulting system meets those needs effectively. Within requirements engineering, two fundamental types of requirements emerge: functional requirements (FRs) and non-functional requirements (NFRs).

Functional requirements encompass the 'what' of a software system. They delineate the desired behaviors and functionalities that the system must exhibit. These requirements delve into the specifics of tasks, actions, and services that the system should execute [4]. Expressed as statements, use cases, or user stories, functional requirements detail how the

system should respond to diverse inputs or events. Notably, functional requirements steer clear of the 'how' and concentrate solely on defining the system's intended behavior. They encompass the features, capabilities, and functionalities that are vital for the system to fulfill its designated purpose.

Contrastingly, non-functional requirements, often referred to as quality attributes or system qualities, encompass the 'how' of a system's behavior. These requirements encapsulate the attributes or properties that dictate the system's performance and conduct [5]. While functional requirements center on the system's functionalities, non-functional requirements pivot towards the system's overarching quality and constraints. These requirements span concerns such as system performance, security, scalability, usability, and availability. Their role is to ensure the system aligns with defined standards, regulatory norms, and delivers a satisfactory user experience.

Crucially, it is important to recognize that functional and non-functional requirements are inherently interconnected. They collaboratively shape a holistic set of requirements for a system. Functional requirements meticulously outline specific features and functionalities, while non-functional requirements intricately address the system's broader quality attributes and constraints. Together, these categories of requirements synergize to craft a system that effectively caters to the needs and expectations of users and stakeholders alike.

2.2. The ISO/IEC 9126-1 quality standard

A quality model is defined by means of general characteristics of software, which are further refined into sub-characteristics, which in turn are decomposed into attributes, yielding to a multilevel hierarchy quality [6]. One of the most relevant and which we use in this work is ISO/IEC 9126-1 quality standard [7].

The ISO/IEC 9126-1 quality standard provides a framework for defining and evaluating software quality. It focuses specifically on software product quality and is divided into several characteristics and sub-characteristics that describe different aspects of quality. This model is also structured in 4 parts: 1 – quality model, part 2 – external metrics, part 3 – internal metrics and part 4 – quality in use metrics. In Table I. we show the six quality characteristics defined in ISO/IEC 9126-1 quality standard and their decomposition into sub-characteristics.

Characteristics	Sub Characteristics
Functionality	Suitability
	Accuracy
	Interoperability
	Security

Reliability	Maturity
	Fault Tolerance
	Recoverability
	Availability
Usability	Understandability
	Learnability
	Operability
	Attractiveness
	User error protection
Efficiency	Time behavior
	Resource utilization
	Capacity
Maintainability	Analyzability
	Changeability
	Stability
	Testability
Portability	Adaptability
	Installability
	Coexistence
	Replaceability

Table I. ISO/IEC 9126-1 characteristics and sub characteristics.

These quality characteristics and respective sub-characteristics help evaluate the overall quality of a software product. They provide a comprehensive framework for identifying and defining the specific non-functional requirements that need to be met. We provide a quick summary of each of these characteristics in the following way:

1. Functionality: This characteristic relates to the system's ability to provide the functions that meet specified needs.
2. Reliability: Refers to the software's ability to perform its intended functions without failures or errors under specific conditions.
3. Usability: Usability focuses on the software's ease of use and the effectiveness of user interactions.
4. Efficiency: Relates to the software's ability to use resources effectively in order to perform its functions.
5. Maintainability: Refers to the ease with which the software can be modified, enhanced, or repaired.
6. Portability: Refers to the software's ability to be transferred from one environment to another.

By following the ISO/IEC 9126-1 standard, we can ensure that the software product meets the defined quality requirements and addresses the various aspects of quality that are important for its intended use [8]. Is important to note that we use this standard as basis for developing the quality model presented in this paper, based on the IQMC method, for evaluating the AHFE Open Access system.

2.3. The IQMC method (Individual Quality Model Construction)

The IQMC method is a valuable approach for defining quality models in various software domains. By adopting a mixed model approach, it provides a starting model that can be customized for specific domains [9]. In order to achieve this customization, it is necessary to select a quality framework or perspective that will influence the resulting catalog. In this case, we use the ISO/IEC 9126-1 quality standard framework, as described in section 2.2.

The IQMC method offers a set of guidelines and techniques aimed at identifying the appropriate quality features to be included in an individual quality model. It consists of seven well-defined steps:

1. Investigation of the domain of interest to gain a comprehensive understanding of the required tasks.
2. Determination of which quality sub-characteristics from the standards are applicable to the domain of interest.
3. Decomposition of the selected sub-characteristics into a hierarchy of sub-characteristics.
4. Decomposition of lower-level sub-characteristics, which are used for classification rather than measurement, into measurable attributes.
5. Recursive decomposition of derived attributes into basic ones.
6. Establishment of relationships between quality attributes, such as logical dependencies, synergies, and trade-offs.
7. Determination of appropriate metrics for the basic attributes.

The IQMC method has undergone validation in numerous domains, both within academic research and industrial applications. Its effectiveness and practicality have been demonstrated in these real-world scenarios [10].

By employing the IQMC method, researchers and practitioners can systematically define and refine quality models tailored to specific software domains. This approach enhances the understanding of quality requirements and contributes to the development of high-quality software solutions.

2.4. Overview of AHFE Open Access System

Applied Human Factors and Ergonomics (AHFE) [11] is an international conference company that is also a publisher for its generated material specialized in academic journals, books, and other scholarly publications. The company provides researchers, scholars, and the general public with access to a wide range of scientific and technical resources. This company offers an Open Access (OA) platform called AHFE Open Access [12] that was first released for public use on January/2022 and now stands with over 110 published volumes wrapping up a total of 4000+ articles from 10 different conferences and now made freely accessible for the academic community.

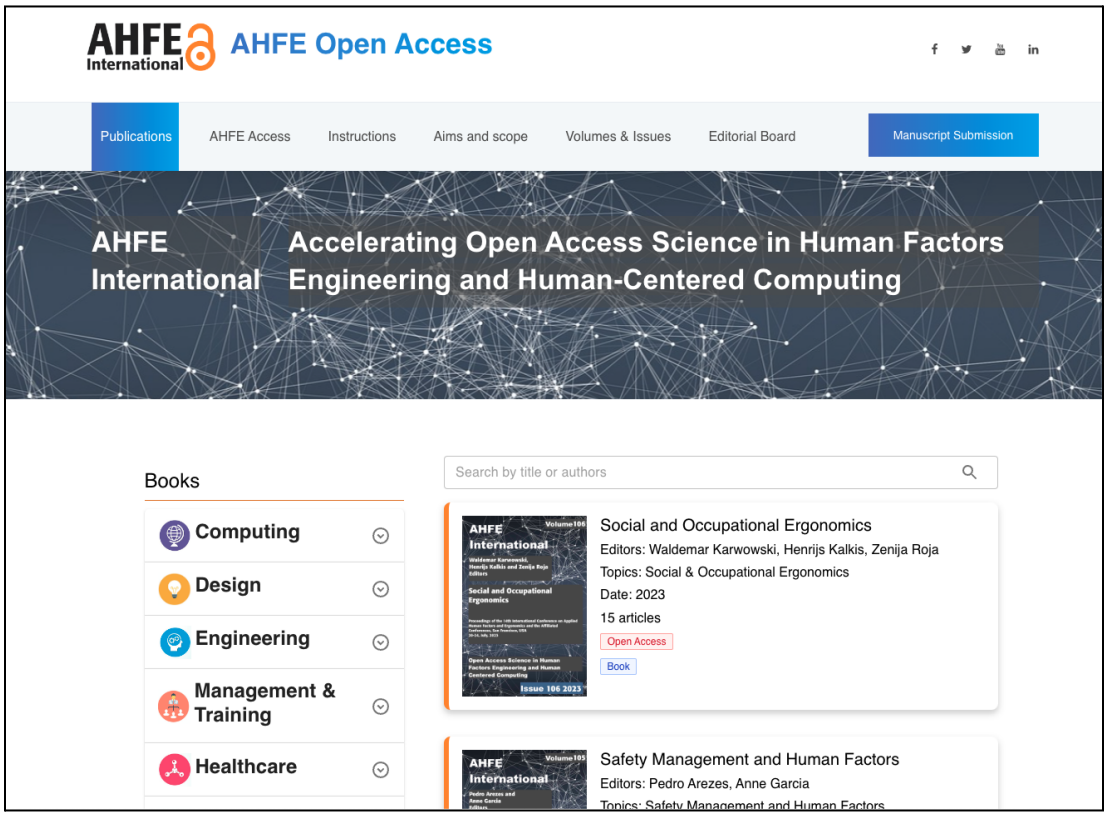

Figure 1. AHFE Open Access Homepage

AHFE Open Access is an initiative by AHFE that aims to promote open access publishing. Open access refers to the practice of making scholarly research freely available to the public, removing barriers such as subscription fees or paywalls. AHFE Open Access allows authors to publish their research articles under an open access license, making them freely accessible and downloadable by anyone. Figure 1 provides an overview of the website.

Authors who choose to publish through AHFE typically retain the copyright to their work while granting others the right to use, distribute, and build upon their research, provided proper attribution is given. This helps to increase the visibility and impact of the research by making it accessible to a broader audience.

AHFE Open Access covers a wide range of disciplines and offers open access journals, books, and conference proceedings. The platform follows a rigorous peer-review process to ensure the quality and integrity of the published research. It also provides various services and features, such as online submission systems, article-level metrics, and enhanced digital formats.

Overall, AHFE Open Access serves as a platform for researchers to disseminate their work openly and contribute to the global knowledge base. It aligns with the principles of open science and facilitates the sharing of research findings with a wide audience of 65 thousand impressions from 85+ countries just over the period of July/2023.

3. Methodology

The methodology for the development of this case study followed the principles of the classic book written by Robert K. Yin: “Case Study Research: Design and Methods”[2].

3.1 Research project components

According to the author, for case studies, five components of a research project are particularly important. They are: the study's questions (1); the propositions (2); units of analysis (3); linking the data to propositions (4); and criteria for interpreting the findings (5).

This first component suggests that the formulation of research questions provides an important key to establishing an appropriate case study strategy, particularly for questions of "how" and "why." In our case, the research questions revolved around the non-functional requirements of the system:

RQ1 - What are the main non-functional requirements for the AHFE system?

RQ2 - How can the non-functional requirements be observed in the system?

RQ3 - Why are these non-functional requirements important for the AHFE system?

RQ4 - What metrics can be used to evaluate these non-functional requirements?

RQ5 - How can we collect these metrics?

RQ6 - How can the results highlight areas for the system improvement?

As for the propositions, each of them focuses on something that should be examined within the scope of the study. The propositions for this study were:

- The system under consideration will have a level of compliance with a non-functional requirement proportional to the result of the collected metrics.
- The system will be considered compliant with the industry standards when it achieves satisfactory metrics for all identified non-functional requirements.

The units of analysis component relates to the fundamental problem of defining what a "case" is. The definition of the units of analysis is related to how the initial research questions were formulated. Therefore, our unit of analysis were the attributes of the AHFE Open Access web service, which included the source code and the web interface.

The fourth and fifth components represent the steps of data analysis in the case study research, and a research design should provide the foundation for this analysis. We linked the data to the propositions using the "fit to pattern" approach described by Donald Campbell [13]. Through this approach, we identified reference system patterns for comparison. We also applied this idea to the fifth component, as the data fit one pattern much better than another. A question that could be asked is: What level of fit is required to be considered a match? The results found in this article are that different patterns contrasted clearly enough, allowing the interpretation of findings in terms of comparing at least two competing propositions.

3.2 The Case Study Research Stages

In order to conduct this case study research, we divided the work in 4 stages. They are: preparation for data collection (1); collection of evidence (2); analysis the evidence from the case study (3); composing the case study report (4);

All data collection was extracted from the application’s source code and its web interface using a variety of methods such as technology analysis and online forms involving the system's stakeholders. It used the IQMC method previously mentioned in section 2.3 for constructing the web service’s quality model. This guided what kind of metrics were relevant to collect from the system and its relevant similar systems for comparisons.

After stage 1, we had a set of attributes and their respective metrics. So, for the second stage, we justified how each metric was going to be collected and applied the framework and methods based on both literature and techniques used in the market. This study involved various sources of evidence: Analysis of documentation, Interviews with stakeholders, Direct/Participant observations, and Analysis of physical artifacts. All records were saved in files in the form of maps and tables. We also followed the 3 principles mentioned by Yin [2]: (1) use multiple sources of evidence; (2) create a database for the case study; (3) maintain the chaining of evidence;

For the stage of analyzing the evidence from the case study, the main method was the suitability to the pattern described by Donald Campbell [13], which involved comparing an essentially empirical pattern with a prognostic-based pattern. Whenever the patterns coincided, its results helped reinforce the internal validity of the case study.

At last, for the fourth stage, it was time to compose the case study report. We used a simple narrative to describe and analyze the case. The narrative information was enhanced with tables, graphs, and images. Some of the illustrative structures used are comparative and linear analytical approaches.

The focus was on reporting the case study in a way that adheres to each of the five general characteristics described by the author. These characteristics include being meaningful, comprehensive, considering alternative perspectives, providing sufficient evidence, and being elaborated in an engaging manner.

4. Quality Model for AHFE Open Access System

In this section we present the core of our work, the construction of a quality model for the AHFE Open Access System, applying the IQMC method described in section 2.3.

4.1. Study of the domain

Step 1 of IQMC consists of the analysis of the domain of interest, i.e. AHFE Open Access in our case. We based this analysis on the understanding and evaluation of this system. We studied this system using a range of sources including: documentation, demos, presentations and architecture models. This was possible because the author of this paper have access to the source code and stakeholders of AHFE Open Access.

To consolidate the most representative information in the scope of this work and to end up Step 1 of IQMC, we identified the core functional requirements as a preliminary step to the quality model construction. Table II shows its content. This information will be later included in the final quality model in a systematic way, but the purpose at this stage was to provide a general landscape of the domain whilst learning which type of information is crucial in this system.

Reference	Description
#1	Articles are freely available to readers.
#2	Articles and Books are easily found and follows a structure.
#3	System administrators can update, create or delete books and articles.
#4	Provide article-level metrics and analytics.

#5	Provide crawlable articles metadata by third-party engines.

Table II. AHFE Open Access core functional requirements

4.2. Identification of sub characteristics

In this subsection we present steps 2 and 3 from the IQMC method. Although not explicitly stated in the ISO/IEC 9126-1 standard, in our work with quality models we have considered sub-characteristics as classifiers of quality concepts, whilst attributes have measures to allow evaluating particular aspects of the domain. According to Step 2, we start by selecting the sub-characteristics from the ISO/IEC 9126-1 quality model that apply to our domain. Ideally, all sub-characteristics would apply, which is quite reasonable since modern software systems are usually quite complex. But in order to limit the amount of work in this article we will only to the core functionalities described in Table II. Therefore, we go over each of the sub-characteristics and analyze if they relate to any of our functional requirements, excluding the ones that does not.

Next, we take these high-level sub-characteristics and decompose them into a second level. We present this second level in the rest of this subsection, although we cannot develop the detail of all selected first-level sub-characteristics of the ISO/IEC 9126-1, we focus only on selected sub-characteristics that we find as the most important ones.

Correctness:

- *Article Search*: One of the most fundamental properties of the system is enabling users to find articles of interest.
- *Administrator operability*: The system is an interface to interact with the open access database. So it is crucial that administrators can update these system informations.

Reliability:

- *System downtime*: This system is utilized by multiple users all over the world, with multiple timezones. And the utilization involves multiple and consistent accesses, therefore it needs to be available with zero downtime.
- *Error handling*: Given that data can be modified, old links and references can become outdated. The system should take care of guiding the user in those flows.

Efficiency:

- *Page Load Time*: It is super important for the system to be indexed by third party solutions. Most of this is done by crawling bots that scan the system and extracts its metadata. To do that, the system needs to present a consistent response time over all pages. But, instead, we will be using the Page Load Time as it is easier to compare with different systems and is the one that will affect the final user the most.

- Page Size: Similarly, we will also consider the page size as it is an important factor, especially when the system is designed to work on mobile devices and possible restrained internet data caps have to be considered.

Integrity:

- *Protected internal features*: Since the platform accommodates both users and administrators, it is essential that certain features are only accessible given the right permissions.

Maintainability:

- *Runtime resource utilization*: Since this system continuously executes and requires resources, the measurement of the resources required during its normal operations is relevant.

Flexibility:

- Code modularity: It refers to the practice of breaking down our software into small, independent, and reusable modules, each responsible for a specific set of functionalities. By adhering to code modularity, we can achieve higher levels of scalability. As our project grows, we can easily add or replace modules without disrupting the entire codebase. This flexibility allows us to adapt to changing requirements and accommodate future expansions without significant refactoring.

Testability:

- Codebase tests coverage: Test coverage refers to the extent to which our code is exercised by automated tests. By maintaining comprehensive test coverage, we can build a more reliable and maintainable software system. Automated tests rigorously evaluate our code, helping to identify and rectify defects, bugs, or unexpected behavior. With a robust suite of tests, we can catch issues early in the development process, minimizing the risk of critical bugs reaching production.

Portability:

- Multi-device interface responsiveness: This refers to the capability of our applications and websites to adapt seamlessly to various devices, including desktops, laptops, tablets, and smartphones. By ensuring responsiveness across different screen sizes and resolutions, we deliver an optimal user experience to all our users, regardless of the device they use.

Reusability:

- Client assets: In our web-based system, we utilize a variety of assets such as buttons and text fields to enhance the user interface. By standardizing these assets across the platform, we aim to significantly improve the user experience while also streamlining the development process.

4.3. Identification of attributes

In this subsection we present steps 4 and 5 of IQMC. The main goal here is to convert these lower level sub-characteristics into basic attributes like String, Number, Date or a primitive so it can be measurable. Some limitations were applied for the sake of this work, so we are only investigating the following sub-characteristics:

Article Search: Defines one attribute (Search usability) that records wether the users can efficiently use the search tool to fulfill their needs.

Administrator operability: It refers to how much of the system content can be modified by a system administrator directly on the platform.

System downtime: It offers two attributes. Amount of system downtime at all times or in fragments of time (Downtime) and the time to recover from a downtime (Time to recover).

Error handling: Similarly, we have two attributes. System Mapped Errors and User recoverability from errors, defining how much of the routes are handles unexpected behaviors and the user response on how to deal with those handlings.

Page Load Time: Two attributes are offered. Being the Average and the Maximum load time for articles pages on the system. We only take the load time of articles pages as they're where the actual information is available.

Page Size: One attribute is described, stating the average data present on the articles page measured in kilo bytes Kb.

Code modularity: A single attribute is offered: Modularity score. This will describe how well structured the codebase is.

Codebase tests coverage: Describes one attribute, stating the amount of coverage on the codebase.

Multi-device interface responsiveness: Provides a responsiveness score that measures how well the system responds to different devices interfaces.

Client assets: Offers a single attribute describing how much of the assets used on the client web interface is reusable.

The decision behind each basic attributes takes into account which framework or technology will be used to collect the data, therefore we more details in Section 5.1. In order to complete this step, Table III presents the characteristics and respective derived and basic attributes.

Characteristic	Sub-characteristic	Derived Attribute	Basic attribute
Correctness	Article Search	Search usability points	Number

	Administrator operability	Administrator operability coverness	Number (Percentage)
Reliability	System downtime	Downtime	Number / Number
		Time to recover	Number
Efficiency	Page Load Time (PLT)	Average PLT	Number
		Maximum PLT	Number
	Page Size	Page Average Size	Number
Flexibility	Code modularity	Modularity score	Number
Testability	Codebase tests coverage	Codebase testing coverage	Number (Percentage)
Portability	Multi-device interface responsiveness	Responsiveness score	Number
Reusability	Client assets	Reusable assets	Number (Percentage)

Table III. Decomposition of the characteristics into attributes.

4.4. Dependencies and metrics

In this subsection we sketch some examples of steps 6 and 7 of IQMC.

(1) Concerning dependencies among non-functional requirements, we can point out some interesting relations. So, for instance, the attribute *Reusable assets* will have a positive influence over the *Responsiveness score*, since using standardized assets will prevent unexpected components behaviors. Constantly updating the history of the monitoring process will consume computational resources. In addition to that, the *Time to Recover* and the *Downtime* have a logical dependency as the faster the recoverability of the system is, less time will be spent in an unavailability state. We can also establish two relationships for the *Page Load Time*: the first one with *Search usability points* as the user searching scenario involves opening multiple pages until the ideal is found and doing it in a faster way will improve the overall user experience, thus the search usability; the second is with the *Page Size* which forms a direct proportionality because larger page sizes require more data to be downloaded from the web server to the user's browser, and the transfer time adds to the overall load time.

(2) Exploring the interdependencies among non-functional requirements unveils intriguing relationships. To illustrate, the *Reusable assets* attribute wields a favorable impact on the *Responsiveness score*. The employment of standardized assets curtails

unforeseen component behaviors. It's worth noting that continually updating the monitoring process history can exact a computational resource toll. In tandem, the *Time to Recover* and *Downtime* exhibit logical interdependence—an expedited recovery mechanism correlates with reduced unavailability duration. Additionally, we can explore two connections involving the *Page Load Time*: firstly, its correlation with the *Search usability points*. Given that the user search scenario often entails navigating numerous pages before reaching the desired outcome, expediting this process amplifies the overall user experience and, consequently, search usability. Secondly, a direct proportionality emerges between *Page Load Time* and *Page Size*. Bulkier page sizes necessitate greater data transfer from the web server to the user's browser, thereby augmenting the overall load time.

As for the metrics, all of them are of three basic types: number; number over number; number percentage. Some examples are:

- *Number*. An example is the average *Page Load Time* that declares how long it takes for a page to load, measured in milliseconds.
- *Number percentage*. An example is Codebase testing coverage stating the percentage of code that is included in at least one automated test.
- *Number over number*. The most usual case, for instance Downtime stating the amount of unavailability over a period of time.

5. Case Study - Application of Quality Model to AHFE Open Access System

In this section, we present the results of the case study that focuses on the evaluation of non-functional requirements within the AHFE Open Access System. In the next subsections, we present the stages of the case study.

5.1. Preparation to collect data

In the preparation for data collection stage of a case study, it is important to select the frameworks, techniques and methods that will be used to measure each of the sub-characteristics basic attributes from the Quality Model built during section 4. Once again, some further limitations were applied minding the scope of this article. Therefore, we arbitrarily selected only a set of the attributes to effectively investigate on this case study.

Search usability points: We prepared a practical survey for users to see how long it takes for them to find a relevant article of a given topic, using only the website search bar. It will be considered relevant if the person downloads the article complete file. The results will be compared against the average session duration in minutes [14] of top similar websites with a search engine: Google Scholar, Open Access Library (OALib) [15] and Springer Open [16].

Time to recover: For data collection, we will deploy a mirrored version of the system, complete with identical resources and a selection of deliberately detrimental endpoints designed to provoke system failures. By analyzing system logs, we can pinpoint

the timestamps corresponding to the initiation and resolution of these breakdowns. Subsequently, we will compute the current time-to-recovery by calculating the delta difference between these timestamps. While numerous studies delve into determining the optimal Maximum Tolerable Downtime (MTD) [17], we maintain a focus on conciseness in this article. Our evaluation strategy involves a comparative analysis against open-source codebases utilizing akin technologies.

Page Load Time: To collect this, we first have to clarify how the page loading of the system works: it uses a technology called SSR (Server-side rendering) meaning that the content of the website is generated on the server, then sent to the browser [18]. All analyzed platforms also uses this kind of technology as it is a standard on SEO (Search Engine Optimization) webpages. Based on that, we will take the timestamp of when the server returns the page content and compare to the timestamp of when the GET request was made. To evaluate if the results from this metric are on toes with the industry standards and the case studied system's core functionality described at Reference #5 of Table II, we will compare the Maximum PLT to to the maximum waiting time of Google crawlers since Google Search is the leading search engine on the actuality [19] and the Average PLT with similar platforms mentioned when describing the plan for the *Search usability points*.

Page Size: The same approach used for Page Load Time will be replicated on this metric but instead of time, we will be getting the page data size.

Modularity score: This will be collected through the help of SonarQube - a widely-used platform for continuous code quality inspection. It provides various code analysis features, including code modularity checks. It can identify issues related to package dependencies, cyclic dependencies, and overall code structure [20]. The Maintainability score reported takes the modularity in consideration therefore will be the one used for this metric. The platform will also be used on popular open source codebases to get an estimate of the real world application.

Codebase testing coverage: Similarly, we will be using a code coverage reports generator for Java projects solution to collect this metric: JaCoCo [21]. And similarly to the Modularity score mentioned above, we will also use JaCoCo against popular and well-founded open source Java codebases to compare.

Responsiveness score: Our approach entails conducting a field study to gauge this aspect. The methodology involves enlisting a group of participants to execute a series of tasks on both desktop computers and mobile devices. Subsequently, we will compute the average deviation as a quantifiable metric and juxtapose it against analogous platforms—akin to the methodology proposed for the *Search usability points* assessment.

5.2. Data collection

In this stage, we will be applying the techniques discussed in the previous sub-section. It is worth to mention that for all of the sub-characteristics metrics that involves user data collection, we will be using the standardized recommended number of 5 participants in our qualitative study [22]. These participants, aged from 25 to 35, are all familiar with academic

routines and have completed their higher-level studies. Importantly, they are not avid users of any of the measured platforms, which ensures that we collect data from unbiased individuals.

Commencing in a chronological order, we proceeded to assess the *Search Usability Points*. To this end, we designated a task for each participant to undertake—a task involving the search for a pertinent article of interest, ideally suited for their respective final theses. Notably, all five participants are well-versed in article research practices, but have never used none of the three platforms. Capitalizing on this familiarity, we executed a hands-on survey, wherein each participant engaged with the task using a desktop computer. The study was conducted across three distinct platforms: AHFE Open Access, Springer Open, and OALib. Pertinently, we recorded the time taken by each participant to accomplish this task—for every second passed, one point was added. The timer commenced the moment the platform's homepage loaded and ceased upon the participant's commitment to read the article—typically signaled by clicking to download the full text. Consequently, we captured five instances for each platform, culminating in the generation of average times for comparison. The findings, showcased in Table IV, encapsulate the average time per platform. Naturally, a lower value indicates superior performance in this context.

WebSystem	Search Usability Points
AHFE Open Access	97
Springer Open	67
OALib	81

Table IV. Data collection for Search Usability Points

The subsequent two sub-characteristics, namely *Page Load Time* and *Page Size*, were jointly assessed. Leveraging an online tool, we gauged load times across diverse global locations. This tool also furnished page size metrics, simplifying data collection. The challenge lay in amassing sufficient data to derive a credible average for each location, followed by consolidating these figures into unified averages for page load time and size. This comprehensive data acquisition unfolded across three distinct websites: (1) AHFE Open Access; (2) Springer Open; (3) Open Access Library - OALib. A Python script, crafted by the author, facilitated the random selection of 25 articles from each site. This script automated the online tool, transcending the need for the web interface. The script mimicked requests and substituted them with multiple parametrized CURL commands, tailored for various locations and URLs. Detailed results are accessible in Table V.

Websystem	Average Page Size	Average PLT	Maximum PLT

AHFE Open Access	1400 KB	1.52s	2.54s
Springer Open	438.7 KB	1.35s	2.46s
OALib	656.7 KB	3.1s	3.86s

Table V. Data collection for Page Load Time and Page Size

Likewise, the *Modularity Score* and *Codebase testing coverage* underwent simultaneous evaluation. The initial step encompassed the identification of open-source codebases sharing a similar tech stack with AHFE Open Access. Given the system's bifurcation into backend (Java with SpringBoot) and frontend (Next.JS), our focus narrowed onto the backend, housing the crux of server logic and intricate structures ripe for analysis. In this context, our scrutiny embraced solely the backend. Among the systems available for comparison, we chose two: (1) ThingsBoard—an open-source IoT platform for data collection, processing, visualization, and device management [23]; (2) Apollo—a reliable configuration management system. It can centrally manage the configurations of different applications and different clusters [24]. These platforms were chosen because both uses Java with Springboot on the backend and follows very similar code structure as our AHFE Open Access. Employing the JaCoCo tool for assessing codebase test coverage proved to be a streamlined process, entailing repository cloning and the addition of a Maven plugin to the package. The focal metric for each system rested on total *Lines coverage*, extracted from the analytical insights of the tool's report. Similarly, leveraging the SonarQube platform, a parallel methodology was applied to the trio of repositories. The outcome materialized in the form of the *Code Smell* metric, embedded within the generated Maintainability report—where a lower value signifies enhanced code quality. The detailed results for both evaluations are comprehensively presented in Table VI.

WebSystem Repository	Lines coverage (JaCoCo)	Code Smell (SonarQube)
AHFE Open Access (Backend)	57%	1320
Thingsboard	93%	520
Apollo	89%	650

Table VI. Data collection for Modularity Score and Codebase testing coverage

Continuing our evaluation, we turn our attention to the *Time to Recover* sub-characteristic. Adhering to the suggested approach, we established a malicious endpoint within AHFE Open Access, mirroring the procedure for the remaining two systems, namely Thingsboard and Apollo. The shared utilization of the Spring Boot

framework across these systems facilitated a streamlined process, ensuring a more authentic comparison. We used the native */shutdown* endpoint available under *Spring Boot Actuator* and measured this metric. Expanding on the details provided in Section 5.1, the collected results are available for reference in Table VII.

WebSystem Repository	Time to Recover
AHFE Open Access (Backend)	22s
Thingsboard	37s
Apollo	33s

Table VII. Data collection for Time to Recover

Finally, we assess our metric for the *Responsiveness score*. To achieve this, we delineated a task for our participants, tailored to various devices, thereby illuminating the influence on task completion times. Our chosen devices encompassed the Apple MacBook Air M2, symbolizing the desktop computing realm, and the Apple iPhone 14 Pro Max, emblematic of mobile interactions. Adhering to our earlier stipulation of involving a group of five participants, they underwent this testing protocol. Once more, we harnessed the platforms of OALib and Springer Open as our testing grounds, allowing us to parallel the tasks and scrutinize outcomes. These tasks encompassed downloading a specific article's PDF based solely on its title and extracting the corresponding DOI link. Notably, our participants boasted familiarity with Apple products, and we gave them a brief five-minute window to navigate each platform using the desktop device prior to task initiation. Additionally, a strategic sequencing was employed—participants commenced tasks on the desktop device before transitioning to the mobile counterpart, with the intention of potentially garnering more favorable mobile browsing results. A comprehensive tabulation of our final results is available within Table VIII.

WebSystem	Multi-device Deviation
AHFE Open Access	1.5s
Springer Open	7.0s
OALib	1.5s

Table VIII. Data collection for Responsiveness score

5.3. Analyzing the data

Within this subsection, we analyze the data gleaned from our case study. Our exploration aims to extract meaningful insights that illuminate the performance and characteristics of the AHFE Open Access System. By scrutinizing this data, we uncover valuable trends and correlations, contributing to a comprehensive understanding of the system's non-functional requirements and overall user experience.

Upon analyzing the outcomes gleaned from the *Search Usability Points* assessment, a noteworthy observation surfaces: AHFE Open Access exhibits a disadvantage concerning its search feature. This divergence in user experience can be attributed, in part, to the system's notably smaller article library in comparison to the other two systems. While the efficiency of the search tool itself may not be at fault, the scarcity of content emerges as a plausible source of the challenge. Despite this setback, upon filtering out instances where topics were less densely populated within the system, slightly elevated completion times persist. Notably, this phenomenon can be attributed to the impact of page loading time, which occasionally extended participant waiting periods before engaging with full article abstracts. Further elaboration on this aspect is slated for the subsequent paragraph.

Based on the data gathered for *Page Load Time* and *Page Size*, it is evident that our analyzed system aligns with industry expectations concerning the average page load time. However, a swift data scrutiny reveals an unusually high maximum PLT compared to the average, attributable to the geographical location of the system's servers in North America. Insights gleaned from website traffic disclose that while 20% of visits originate from North America, over 30% originate from European nations that don't enjoy proximity to the server. Furthermore, the Average Page Size deviates considerably from the norm, surpassing the compared system's average by more than double. Delving into the rationale behind this discrepancy, it becomes apparent that the other systems have judiciously streamlined their pages by omitting resource-intensive elements such as images.

The data collection for the Modularity Score yielded a compelling revelation, shedding light on the studied system's deficiency in terms of code modularity. A comparative analysis against elevated industry benchmarks positions AHFE Open Access at a discernible disadvantage. Nonetheless, despite the system's inherent simplicity, the SonarQube platform underscored several recommended code modifications. This underscores a pertinent area for enhancement, beckoning focused efforts to bolster the system's modularity.

With the outcomes of the *Codebase testing coverage* analysis in hand, a conspicuous departure from established industry patterns, that were collected within this study, becomes evident. This deviation may find its roots in the scale of the projects under comparison, where the open-source landscape beckons numerous developers, inherently mandating extensive test coverage. However, it's worth noting that AHFE Open Access falls significantly below the threshold that might warrant leniency in this regard.

Examining the metrics for the *Time to Recover* sub-characteristic, a noteworthy distinction emerges, placing AHFE Open Access at a pronounced advantage in comparison

to the other assessed systems. Despite sharing a common technological framework, our studied case distinguishes itself as a simpler construct, boasting reduced intricacies in terms of logic and third-party dependencies. Notably, AHFE Open Access leverages deployment within a Kubernetes network, a facet that underpins its resilience. In the event of a failure, the impact is localized to a single operational pod, potentially translating to near-zero downtime. This attributes a compelling real-world edge to our testing outcomes.

Shifting our focus to the data compilation for the *Responsiveness score*, a revealing panorama comes into view. AHFE Open Access solidifies its stance in a notably advantageous position, showcasing minimal deviation between mobile and desktop devices. This distinction is especially pronounced when juxtaposed with Springer Open, where the mobile experience is cleary affected by interface limitations. Notably, despite the mobile experience bearing striking semblance to desktop usage, the statistics from July 2023 illuminate an interesting facet: a mere 15% of website interactions occurred via smartphones or tablets. This intriguing statistic gives rise to a hypothesis—conducting article research on mobile devices might not be a prevailing norm, with stakeholders seemingly accustomed to engaging in such tasks primarily via desktop computers.

5.4. Final Remarks

The comprehensive examination of the gathered data has unveiled a wealth of insights into the performance and other non-functional requirements of the AHFE Open Access System. This section presented a detailed exploration of the findings, highlighting significant observations and their implications.

Overall, our approach of constructing a quality model to guide the case study has proven highly effective. Despite the article's inherent limitations in scope, we successfully delved into six key sub-characteristics, assessing the system's response to pertinent non-functional requirements. The analysis reveals that while the AHFE Open Access System does address the NFRs, it becomes apparent that certain requirements teeter on the edge of acceptability. Identifying these specific gaps offers an opportunity for targeted improvements, aiming to elevate the system to meet industry standards and stakeholder needs.

In essence, this in-depth analysis provides a multifaceted view of AHFE Open Access, underlining both its strengths and areas ripe for improvement. The insights gleaned from these data-driven assessments pave the way for informed decisions and targeted enhancements, fostering a more refined and user-centric software ecosystem.

6. Conclusions, limitations and future work

In this study, we conducted a comprehensive assessment of the AHFE Open Access system's non-functional attributes using a quality model derived from the ISO/IEC 9126-1 standard. The application of the IQMC method allowed us to systematically evaluate the system's performance across various sub-characteristics and attributes. The findings offer

valuable insights into the system's strengths, areas for improvement, and potential avenues for future enhancement.

6.1. Conclusions

Our analysis revealed several noteworthy conclusions:

Search Usability: AHFE Open Access demonstrated competitive search usability; however, its smaller article library may impact user experience. Further investigation into expanding content could contribute to enhancing search efficiency.

Page Load Time and Size: The system's average page load time aligned with industry norms, while the maximum load time and page size reflected geographical disparities. Addressing these disparities and optimizing content delivery could lead to improved user experience.

Modularity and Code Coverage: The system exhibited potential areas for improvement in terms of code modularity and test coverage. Enhancing these aspects could lead to more maintainable and reliable codebases.

Time to Recover: AHFE Open Access demonstrated a commendably rapid recovery time, positioning it advantageously against comparable systems. Leveraging its Kubernetes deployment, the system's resilience contributed to minimal downtime.

Responsiveness: The system showcased consistent responsiveness across various devices, with minimal deviation between mobile and desktop interfaces. Insights from user behavior patterns suggest potential opportunities for responsive design enhancements.

6.2. Limitations

Despite the comprehensive nature of this study, several limitations need to be considered. Firstly, the scope of the case study is limited to a specific system, AHFE Open Access, and may not generalize to other contexts. Additionally, the study focused on a subset of attributes due to constraints in resources and time. The reliance on qualitative user data collection methods introduces subjectivity and potential biases. Furthermore, the study's effectiveness is contingent on the selected benchmarks and similar systems used for comparison.

6.3. Future Work

To build upon the insights gained from this study, several directions for future work emerge:

Enhanced Search Functionality: Investigate strategies to augment the system's search usability, potentially through content expansion, advanced search algorithms, or user behavior analysis.

Optimized Page Loading: Address geographical disparities in page load times and sizes, exploring content delivery networks (CDNs) and image optimization or lazy loading techniques.

Codebase Quality: Pursue initiatives to improve code modularity and test coverage, leveraging insights from industry best practices and open-source projects.

Responsive Design Refinement: Further analyze user behavior patterns and preferences to refine responsive design strategies and enhance the user experience across various devices.

Benchmark Diversification: Expand the scope of benchmark systems and attributes for a more comprehensive and robust comparison, accommodating diverse technology stacks and usage scenarios.

Quantitative Validation: Combine qualitative insights with quantitative data, employing larger participant samples and statistical analysis to validate and strengthen the findings.

Long-Term Performance Monitoring: Implement continuous monitoring and assessment of the system's non-functional attributes to track improvements and address potential regressions over time.

In conclusion, this study has provided valuable insights into the non-functional requirements of the AHFE Open Access system, shedding light on its performance, strengths, and areas for optimization. While acknowledging its limitations, this work serves as a foundation for future endeavors aimed at enhancing the system's quality, user experience, and overall impact.

References

[1] Chng, L., et al.: Non-functional Requirements in Software Engineering, vol. 5. Springer Science and Business Media, Berlin (2012)

[2] Yin, R. K. (2003). Case study research: design and methods. 3rd ed. Thousand Oaks, Calif., Sage Publications.

[3] S. Wagner, D. Méndez-Fernández, M. Kalinowski and M. Felderer, "Agile requirements engineering in practice: Status quo and critical problems", CLEI Electron. J., vol. 21, no. 1, 2018.

[4] Supplement 4-A, A Procedure for Requirements Analysis". Systems Engineering Fundamentals (PDF). United States Government US Army. 2001. ISBN 978-1484120835

[5] Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. International Series in Software Engineering, vol. 5, p. 476. Springer, Heidelberg (1999)

[6] X. Franch, J. Carvallo, “A Quality-Model-Based approach for describing and evaluating software packages,” In Proceedings 10th IEEE Joint Conference on Requirements Engineering (RE), 2002.

[7] International Organization for Standardization, ISO/IEC standard 9126: Software Engineering – Product Quality, part 1. 2001.

[8] Olsina L., Lew P., Dieser A., Rivera B.: Updating Quality Models for Evaluating New Generation Web Applications. Journal of Web Engineering, Special issue: Quality in new generation Web applications. Rinton Press. USA. 11:(3), (2012)

[9] E. Demidenko, "Mixed models: theory and applications," 1st ed., Ed. Wiley Series in Probability and Statistics, 2004.

[10] J. Carvallo, X. Franch, C. Quer, “Determining criteria for selecting software components: lessons learned,” IEEE Software, 2007, pp. 84-94.

[11] [Applied Human Factors and Ergonomics International Conference]. (2023). AHFE Conference. Retrieved July 30, 2023, from http://ahfe.org/

[12] AHFE Open Access. (2023). AHFE Open Access. Retrieved July 30, 2023, from https://openaccess.cms-conferences.org/

[13] Campbell, D. T. (1975). Degrees of freedom and the case study. Comparative Political Studies, 8, 178-193.

[14] Rajesh Kumar Goutam (2018). Correlation Based Evaluation for Search Tools. 2018 5th International Conference on Signal Processing and Integrated Networks (SPIN).

[15] Open Access Library (OALib). (2023). OALib. Retrieved August 3, 2023, from https://www.oalib.com/

[16] *Springer Open Access*. (2023). SpringerOpen. Retrieved August 3, 2023, from https://www.springeropen.com/

[17] NIST SP 800-34, REV 1; Contingency Planning Guide for Federal Information Systems; National Institute of Standards and Technology; U.S. Department of Commerce: Gaithersburg, MD (May 2010).

[18] What is server-side rendering: definition, benefits and risks. (2022, July 21). Solutions Hub. Retrieved July 7, 2023, from https://solutionshub.epam.com/blog/post/what-is-server-side-rendering

[19] "Search Engine Market Share Worldwide | StatCounter Global Stats". StatCounter Global Stats. Archived from the original on December 10, 2020. Retrieved April 9, 2021.

[20] Code Quality Tool & Secure Analysis with SonarQube. (2023). Sonar. Retrieved August 10, 2023, from https://www.sonarsource.com/products/sonarqube/

[21] Intro to JaCoCo. (2023, May 5). Baeldung. Retrieved July 8, 2023, from https://www.baeldung.com/jacoco

[22] Nielsen, Jakob, and Landauer, Thomas K.: "A mathematical model of the finding of usability problems," Proceedings of ACM INTERCHI'93 Conference (Amsterdam, The Netherlands, 24-29 April 1993), pp. 206-213.

[23] T. (2016). GitHub - thingsboard/thingsboard: Open-source IoT Platform - Device management, data collection, processing and visualization. GitHub. Retrieved August 10, 2023, from https://github.com/thingsboard/thingsboard/

[24] A. (2016). GitHub - apolloconfig/apollo: Apollo is a reliable configuration management system suitable for microservice configuration management scenarios. GitHub. Retrieved August 10, 2023, from https://github.com/apolloconfig/apollo