

Research Article

An Integrated Approach to Managing Software Quality in Complex Systems

Félix Témolet^{1,*} , Desislava Atanasova² 

¹Chief Information Officer (CIO) Advisory Services, Capgemini Germany GmbH, Hamburg, Germany

²Department of Informatics and Information Technologies (IIT), University of Ruse “Angel Kanchev”, Ruse, Bulgaria

Abstract

This paper addresses the persistent challenge of implementing software quality models in complex organizational environments such as a large insurance company undergoing agile transformation. The paper posits that a universally accepted definition of software quality is inadequate and must be operationalized using a quality model that refines the characteristics in accordance with dynamic stakeholder expectations for the software product within their individual context. To ascertain the quality of software products, it is not sufficient to determine the quality of individual characteristics; rather, they must be combined in a system and classified. Quality models guarantee that all quality characteristics are considered and provide a set of quality characteristics that are of interest to different stakeholders. When introducing a suitable quality model in a company, it is essential to consider the possibility of measurement, as only using metrics and defined quality indicators is an objective evaluation possible. This gap can be closed using the pragmatic and value chain-oriented Quality Gate Framework developed as part of this study. It ensures both product quality and process quality across all phases of software development. The framework is empirically grounded through 58 expert interviews conducted using the Grounded Theory method. This approach enabled the extraction of context-specific themes on the one hand, and the validation of the model's relevance on the other. This document highlights the effectiveness of the model in improving transparency, stakeholder alignment, and quality assurance without compromising agility. It also provides a transferable and easily adaptable framework that integrates quality assurance mechanisms with agile practices, offering practical guidance to organizations undergoing similar transformations. Furthermore, the model's compatibility with DevOps approaches, CI/CD, and automated testing environments is discussed, as well as its cultural and managerial implications. This document contributes to the discourse on software quality by demonstrating how systemic integration along the value chain can lead to sustainable improvements in software development outcomes.

Keywords

Software Quality Model, Product Quality, Process Quality, Quality Characteristics, Quality Gates, Software Development, Value Chain

1. Introduction

The software quality model discussed in this paper has been developed and introduced in the company by the author. This

model considers not only the various stages of software development but also all sub-processes that contribute to the

*Corresponding author: felix.temole@capgemini.com (Félix Témolet)

Received: 16 June 2025; **Accepted:** 30 June 2025; **Published:** 22 July 2025



Copyright: © The Author(s), 2025. Published by Science Publishing Group. This is an **Open Access** article, distributed under the terms of the Creative Commons Attribution 4.0 License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution and reproduction in any medium, provided the original work is properly cited.

value chain. The fundamental objectives are to guarantee product quality, enhance process quality, and regulate software development costs. Software quality models have been used to evaluate software product quality since the 1970s. Over time, these models have evolved, with many designed to address specific aspects of software quality. However, this approach limits their ability to assess the quality of software products holistically [1]. Previous publications have not provided a comprehensive examination and listing of all models that attempt to describe the various properties of software product quality.

Recent research underscores the significance of a model-based approach for attaining both high software quality and high efficiency in the development of software systems [2, 3]. With respect to the intricacy of the value chain within the organization, a coherent and readily comprehensible software quality model should offer indispensable guidance for all stakeholders in delineating their perceptions of software quality. A software quality model thus provides a framework for the sub-processes within the value chain, allowing for the verification of the expected software quality.

The most recent research findings on software quality models are presented by Champion et al. [4], Lenarduzzi et al. [5], Izurieta et al. [6], Foidl & Felderer [7], and Yadav & Kishan [8]. These findings corroborate those of earlier publications, indicating that research on software quality models remains a pertinent area of investigation.

2. Overview on Software Quality Models

2.1. Software Quality

Software quality is defined by how well a software program or system meets specified requirements and fulfills its intended purpose. Ensuring high software quality is crucial, as low-quality software can result in numerous issues, including security vulnerabilities, suboptimal performance, and system crashes. Various factors contribute to software quality, including reliability, usability, efficiency, maintainability, testability, traceability, availability, reusability, and readability [4, 5, 1].

By ensuring software quality, it is possible to enhance the overall performance of the company, and the value of the software product manufactured by that company [7]. However, due to the complexity and multi-layered nature of software quality, the economic impact is often not directly

tangible. It is therefore important to determine which aspects of software quality are crucial regarding the product as well as the business model in a long-term perspective [9, 10]. The substantial economic benefit of software quality is contingent upon the capacity to effectively address quality concerns [11]. The monitoring, tracking, and measurement of software quality can be achieved through the implementation of standardized processes and the provision of consistent visibility of workflows.

2.2. Different Approaches

The ISO/IEC 25010 quality model addresses the needs of various stakeholders, as outlined in ISO/IEC 250101 [12]. Quality models can be categorized into several approaches: product-based, view-based, process-based, and value-based. Each approach offers a unique perspective on assessing and ensuring software quality, catering to different aspects and requirements of software development and evaluation [2].

1. **Product-based models:** In product-based models, the distinctive characteristics of quality are typically delineated during the modeling process. Such characteristics can be employed to ascertain whether the desired objectives have been attained by the software product. During the software development process, customer requirements are considered and incorporated into the software product [11]. Accordingly, the software product comprises the transformation of customer requirements into functional specifications, the translation of these specifications into a design, the implementation of the design, and the testing of the implementation [2]. This aligns with the product-based perspective on quality, which emphasizes attributes such as performance, reliability, maintainability, usability, portability, compatibility, security, and functionality [5].
2. **View-based models:** In view of the multiplicity of perspectives from which software quality can be viewed by different stakeholders [4], view-based models are of particular interest. Consequently, both product and process quality can be emphasized, as well as the perspectives of the user and developer. To evaluate the quality of a product from the user's perspective, view-based models may be employed. These models concentrate on attributes such as satisfaction, preference, perception, and opinion [2, 11].

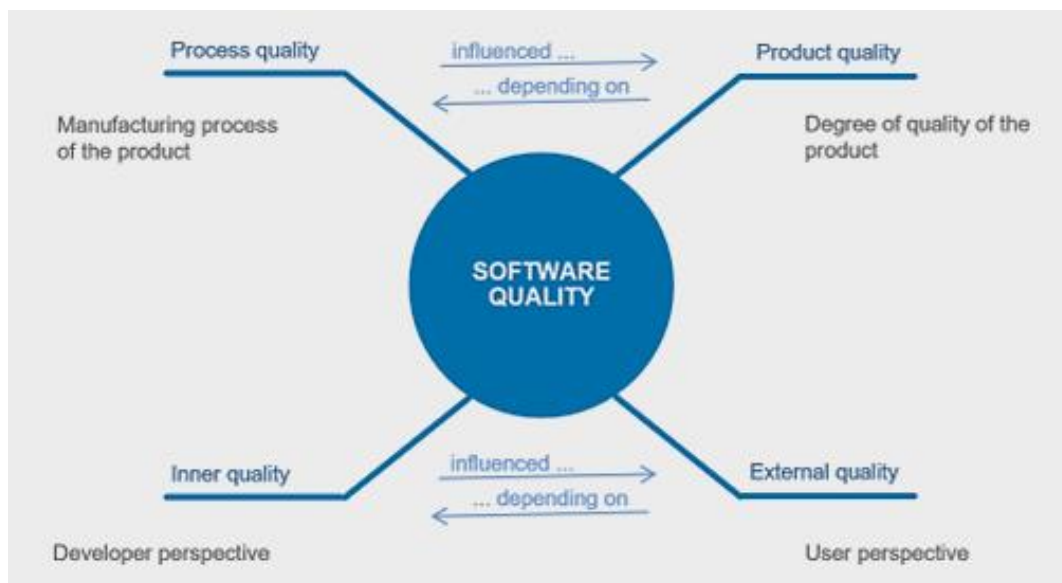


Figure 1. Viewing software quality from different perspectives (Source: Own Representation).

3. Process-based models are necessary in addition to product control because only a high-quality process can produce a high-quality product. This should be considered in future research facilities. As Hassan and colleagues [13] observe, the quality of software is contingent upon the implementation of efficient and effective processes to ensure the integrity of the software development process. Hassan and colleagues

[13] conducted an investigation into the efficacy of various strategies for software development in achieving this objective. Nevertheless, process quality is not an end in itself; rather, it serves to ensure product quality. The AZ model, as proposed by Akbar et al. [14], allows for a process-oriented view of the entire software development cycle.

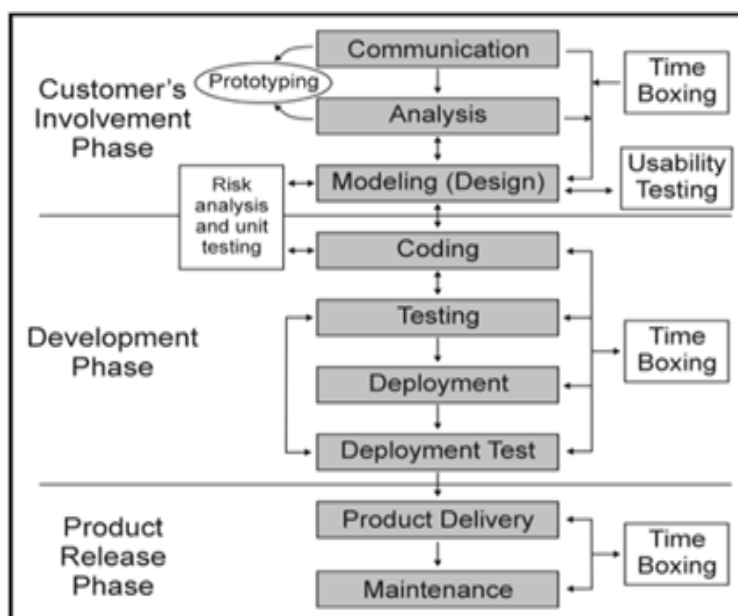


Figure 2. AZ Model of Software Development (Source: Akbar, et al. [14]).

The incorporation of novel activities throughout the software development lifecycle enables the AZ model to transcend the constraints of preceding quality frameworks,

markedly influencing the quality of software products [2, 15]. Furthermore, this model does not adopt a value-based perspective.

4. Value-based models: Granja-Alvarez and Barranco-García [16] developed a software quality model based on the COCOMO model (Constructive Cost Model) that offers a value-based view. The maintenance phase represents the greatest expenditure of resources for a software project; thus, maintainability constitutes an indispensable quality factor for enhancing productivity in the maintenance phase [16]. They developed a model for estimating maintenance costs based on this quality factor and demonstrated its application [16]. For this purpose, the authors categorized the maintenance effort into three categories [16]. These categories are as follows:

- 1) Effort to understand the program code
- 2) Effort to change the program code
- 3) Effort for testing the program code

The AZ model emphasizes maintainability as a critical factor for productivity in the maintenance phase, categorizing maintenance efforts into understanding, changing, and testing the program code [15, 16].

The following table provides a comparative overview of the the software quality models discussed - ISO/IEC 25010, the AZ Model, and Value-Based Models – evaluated across five key dimensions: scope, adaptability, stakeholder orientation, strengths, and limitations.

Table 1. Comparative overview of the software quality models.

Model	Scope	Adaptability	Stakeholder Orientation	Strengths	Limitations
ISO/IEC 25010	Comprehensive standard covering product and system quality characteristics	Moderate – adaptable through tailoring of characteristics and sub-characteristics	Broad – includes developers, testers, users, and maintainers	Well-established, standardized, supports traceability and comparability	Can be too generic; lacks contextual flexibility for specific domains
AZ Model	Process-oriented model covering the entire software development lifecycle	High – integrates new activities and supports iterative improvements	Primarily internal – focuses on development and QA teams	Emphasizes maintainability and process quality; supports continuous improvement	Lacks value-based perspective; may not fully address external stakeholder needs
Value-Based Models	Focus on economic and business value of software quality	High – tailored to organizational goals and cost-benefit considerations	Strong external focus – prioritizes customer and business stakeholder perspectives	Aligns quality with business outcomes; supports ROI-driven decision-making	May underemphasize technical quality aspects; requires mature metrics and stakeholder alignment

3. Software Quality Model in the Software Company Operating in the Insurance Industry

3.1. Research Approach

Expert interviews were conducted to enhance the model. The experts were selected according to the objective of the research. The interviews were conducted over a four-month period by the author and two other colleagues from the quality management department of the company. The interviews were recorded on audio tape and subsequently transcribed using the f4@ transcription software [17]. Prior to the inaugural interview with the selected expert, three mock interviews were conducted between the three interviewers assuming various roles (interviewer, interviewee, observer) to pilot the developed interview guide. The objective of the pilot

interviews was to facilitate rehearsal of the interviews in advance of their implementation under actual conditions [18, 19]. The guideline was thus tested regarding consistency and clarity, and the procedure, temporal scope, formulation, and comprehensibility of individual questions were examined. Any requisite modifications were implemented [18-20]. Each expert interview was scheduled to last for 90 minutes. In some cases, the interviews extended beyond the allotted time, particularly when the interviewees deemed a comprehensive examination of specific topics both pertinent and indispensable [21, 22]. The researcher's role was not merely observational; he also served as a moderator during the interview process. Various measures were taken to ensure the integrity and objectivity of the interviewer team with regard to potential bias. First, triangulation was conducted by involving the interviewer team in the coding process and comparing interpretations. Subsequently, reflexive memos were maintained to document assumptions and decisions throughout the analysis. Finally, a consistency check of expert statements was carried out in relation to their respective organizational affiliations.

iations to validate interpretations and ensure alignment with their perspectives. Theme extraction was guided by constant comparison and iterative refinement of categories. A coding schema was developed to ensure consistency across coders and to facilitate traceability of interpretations. Data saturation was reached after 58 interviews, as no new themes emerged and existing categories were sufficiently populated with representative data. Selective coding involved identifying a core category – “value-chain-integrated quality assurance” - that linked all other categories and explained the central phenomenon observed in the data. In the axial coding phase, these initial codes were grouped into categories based on conceptual similarities. For instance, codes like “cross-team coordination”, “shared responsibility”, and “gatekeeper accountability” were clustered under the category “collaborative quality assurance”. To enhance methodological transparency, the analytical process followed a structured Grounded Theory approach. The coding strategy was implemented in three stages: open coding, axial coding, and selective coding. During open coding, interview transcripts were segmented into discrete meaning units and labeled with initial codes. For example, statements such as “quality gates help us coordinate across teams” were coded as “cross-team coordination”. The sample size of 58 experts was considered appropriate due to the relevance of the various roles and stakeholders along the value chain, as well as the inherent complexity of the software itself and the processes involved. This aligns with Mason’s observation that many research studies inadequately address the concept of data saturation, which was carefully considered in this study. Saturation is achieved when the addition of new data is unlikely to yield significant insights.

The following table provides a detailed overview of the expert sample involved in the study, categorized by their professional roles, hierarchical levels, and organizational affiliations. This distribution reflects the deliberate selection of participants to ensure a comprehensive representation of perspectives across the software development value chain. The diversity of roles - from developers and analysts to senior management - was essential for capturing the multifaceted nature of software quality practices within the organization.

Table 2. Distribution of Experts by Role, Hierarchical Level, and Organization (Source: Own Representation).

Role	58
Application Developer	14
Business Analyst	4
Test Designer	3
Test Manager	9
Software Architect	2
Subject Matter Expert	4
Quality Manager	2

Process Manager	1
Product Owner	3
Agile Master/Coach	5
Release Manager	1
Build Manager	2
Requirements Manager	1
Group Leader	4
Head of Department	1
Program Manager	1
Board of Directors	1
Hierarchical level	58
Senior Management	3
Middle Management	16
Operational Level	39
Organization	58
Development Organization	36
Test Organization	13
Subject Departments	5
Program Management	4

The experts were selected based on their competence, experience, company affiliation, task or role, as well as their special knowledge in their respective fields within the value chain [23]. In selecting the experts, consideration was given to their area of responsibility and the specific expertise they bring to the field of software quality [24]. Additionally, the diversity of perspectives and experiences across different hierarchical levels was considered, as well as the depth and longevity of their professional engagement in the domain of software development [19]. The objective was to examine several groups at various stages of the sub-process [18]. Moreover, the experts were not randomly selected; rather, they were deliberately chosen and required to meet the specified criteria.

3.2. Research Context

The research context is that of a software company, where the classic waterfall model was previously used for software development. To achieve greater flexibility and adaptability, the decision was taken to switch to an agile software development methodology [4]. Given the absence of prior experience within the organization, the various departments initiated the transition concurrently. Based on the experience gained, modifications and updates were implemented on an ongoing basis. This led to a lack of a unified solution, as each department followed its own distinct agile implementation. To

address potential software quality issues, the author developed a software quality model to be implemented across all departments. This model is based on agile methodology and incorporates the value chain [11].

The software quality model consists of phases, each culminating in a quality gate. Quality gates are defined as decision points in a development project where the release of the next project step is determined based on predefined quality criteria [4, 25]. The primary objectives are to ensure product quality at these quality-dependent synchronization points,

enhance process quality, and manage software development costs. Five quality gates are employed to maintain the integrity of the five phases in the software development process. The phases are as follows:

- 1) Requirement Collection Phase
- 2) Software Development Phase
- 3) System Test Phase
- 4) Overall Integration Test Phase (Overall System Test Phase)
- 5) Go-Live Phase

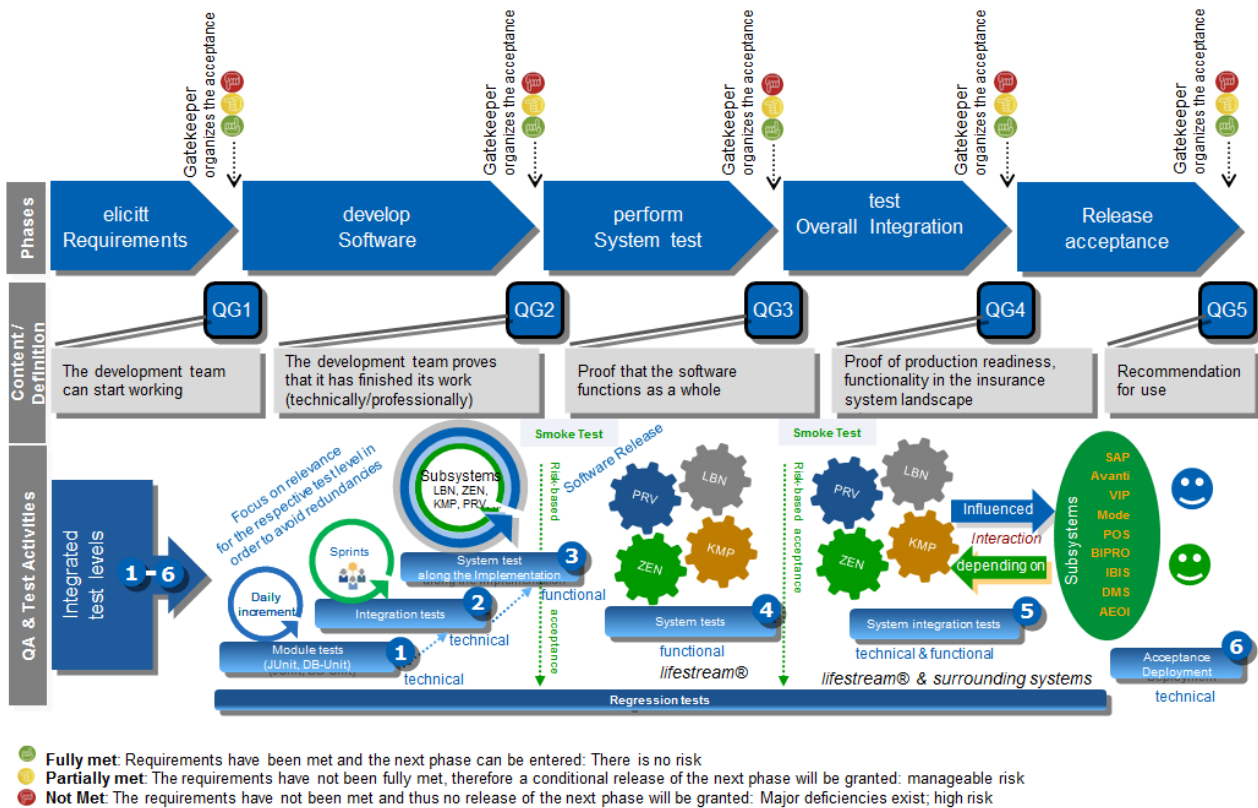


Figure 3. Value Chain of Existing Software Product Development (Source: Own Representation).

It should be noted that each quality gate is the responsibility of a gatekeeper. The gatekeeper is responsible for organizing the acceptance process within the context of quality gate meetings [26]. It is essential to ensure that all relevant stakeholders are included in the composition of the acceptance team to facilitate comprehensive communication and alignment of existing results. The composition of the acceptance team is based on the existing topics and potential stakeholders involved. The objects of testing are software components, content, or services that are evaluated at a designated juncture, or “quality gate,” in accordance with the extent to which they align with the agreed-upon degree of fulfillment (or maturity) of the software [11, 27]. The subsequent phase of the software development process is initiated only when the quality of the available results at the respective quality gate has been determined to be sufficient. The deci-

sion-making process for releasing a quality gate thus depends on the degree of fulfillment, that is, on the achieved maturity of the software regarding the existing results. This is implemented by a traffic light circuit as follows:

- 1) Status Green: A status of “Green” indicates that all requirements have been fulfilled. Accordingly, there is no risk, and the subsequent phase may be initiated.
- 2) Status Yellow: The requisite criteria were not fully met. The risk is deemed to be manageable, and thus a conditional release for the subsequent phase is granted.
- 3) Status Red: The requisite criteria have not been satisfied. Given the elevated level of risk, a conditional release for the subsequent phase is not granted [9].

Considering the findings of the empirical study, the acceptance procedure for the quality gates along the value chain was delineated as follows:

- 1) For Quality Gate 1 (QG1), the development team and the requester must confirm that the requirements have been understood to the extent that implementation is possible. In this context, the test object is constituted by individual topics in the form of requirements specifications and user stories, for example. The acceptance criteria encompass the following characteristics: completeness, consistency, feasibility, testability, clarity/readability, usability, and document properties [4]. A distinctive challenge in ensuring software quality at this quality gate is to prioritize the assessment of individual requirements, the customer benefit, direct communication, and team composition [1]. The collective responsibility for the success of Quality Gate 1 is borne by the product owner, development teams, test organization, and customer (departments, requesters, IT) [26]. The gatekeeper, in collaboration with the relevant parties, issues the release, indicating that the quality gate has been successfully completed and is ready for the next stage. Subsequently, the released requirements can be scheduled and implemented [4].
- 2) For Quality Gate 2 (QG2), both new and existing functionalities are validated using a combination of module and integration tests, as well as system tests conducted during the development phase. The test objects in question are functional requirements (theme, user story, etc.), non-functional requirements (performance test, load test, etc.), maintenance data (configuration data), errors (tickets), test preparation for system tests, and the setup of test environments [4]. For the quality gate to be accepted successfully, it is necessary that both functional and non-functional requirements have been fully implemented and tested, that those responsible for maintenance data (configuration data) have given their approval, that there are no errors (tickets) classified as A-critical or B-high, and that those responsible for system testing have released the test preparation [1]. The challenge in ensuring software quality at this quality gate is to prioritize the assessment of individual functionalities (from a technical and a team-supporting perspective), team outcomes, potential interfaces, and the stability of the subsystem. The collective responsibility for the success of Quality Gate 2 rests with the product owner, development teams, test organization, and customer (departments, requesters, IT). The gatekeeper, in collaboration with the relevant parties, issues the release for the achievement of the quality gate. The designation "release - Ready for QG3" is applicable to functionalities (user stories) that may exert an influence on other subsystems (modules) of the entire software. Consequently, these must be considered in the context of the system test (QG3). A release designated as "Ready for QG5" is applicable to isolated functionalities that are not expected to exert any influence on subsequent subsystems, whether in QG3 or QG4. Consequently, Quality Gates 3 and 4 may be circumvented. This necessitates that software features or modules are developed as autonomously as feasible and are only integrated into software releases at a subsequent stage if necessary.
- 3) For Quality Gate 3 (QG3), the entire software is to be validated as a new software release through the implementation of system tests. The test object is comprised of functional requirements (theme, user story, etc.), non-functional requirements (performance test, load test, etc.), maintenance data (configuration data), errors (tickets), and test preparation for the overall integration test and interfaces [1, 11]. For the quality gate to be successfully accepted, it is necessary that both functional and non-functional requirements have been fully implemented and tested [4]. Furthermore, no errors (tickets) with the classification A-Critical, B-High, or C-Medium must be present, and no errors (tickets) with the priority "Blocker" or "Critical" must be present [9]. It is the responsibility of the individuals overseeing OIT (overall integration tests) and acceptance tests to have completed the preparation for the tests. The specific challenge in ensuring software quality at this quality gate is to prioritize the assessment of individual subsystems (those oriented towards business objectives and team support), the evaluation of overarching team outcomes, the examination of the interactions between subsystems, and the verification and validation of the software. The product owner, test organization, and project manager are collectively accountable for the successful completion of Quality Gate 3. The gatekeeper, in collaboration with the relevant parties, authorizes the release of the product in question upon the successful completion of the quality gate. The designation "release - Ready for QG4" is applicable to software releases that may have a potential impact on subsystems of the entire system landscape of the insurance company. Consequently, these releases must be considered in QG4 as part of the OIT (overall integration tests). A release designated as "Ready for QG5" pertains to discrete software releases that are not anticipated to exert any influence on the broader system landscape encompassed by QG4. In this instance, therefore, QG4 is bypassed [2].
- 4) For Quality Gate 4 (QG4), it is necessary to ensure that the entire software system landscape of the insurance company is secure regarding production readiness. This is to be achieved using the OIT (overall integration test). The successful acceptance test represents the final quality assurance activity. The test objects are comprised of functional requirements (theme, user story, etc.), non-functional requirements (performance test, load test, etc.), maintenance data (configuration data), errors (tickets), and interfaces [1]. For the quality gate to be accepted successfully, it is necessary that both functional and non-functional requirements have been fully implemented and tested, that interfaces to peripheral

systems have been tested, that there are no errors (tickets) classified as A-Critical, B-High, or C-Medium, and that there are no errors (tickets) classified as “Blocker” or “Critical” with the highest level of priority [4]. The success of Quality Gate 4 is a collective responsibility shared by the test organization, departments, and subject matter experts. The designation ‘Ready for QG5’ applies to the respective software release and its interaction with the participating systems within the insurance company’s entire software system landscape.

- 5) For Quality Gate 5 (QG5), it is necessary to examine and assess the outcomes of the preceding quality gates (QG1–QG4) to ascertain the suitability of the software products for acceptance. This acceptance serves as the foundation for the recommendation for the live deployment (commissioning) of the software products. The focus of the inspection is on the test and quality reports [4]. The success of Quality Gate 5 is contingent upon the collective responsibility of program management and system managers [1]. The term “release-going live” is used to describe the release of functionalities or software that have undergone a successful risk assessment for productive use [26]. It should be noted that the attainment of QG5 does not necessarily guarantee the commencement of the software product’s operational phase.

The results of all quality gates are archived in a publicly accessible, centralized repository on the company collaboration platform, making them available to all employees [11]. The test levels in the value chain are sequentially structured, with each level building on the previous one. They are described as follows:

- 1) Test Level 1 - Module Test: This test level focuses on evaluating discrete functional units, or modules, within the larger software system. It is technically oriented and examines the detailed functionality of an application at the module level during the software development phase. The respective development team is responsible for planning, developing, and automating these tests, as well as ensuring their satisfactory outcomes [4]. They must implement all test activities for the developed features in a timely manner, allowing for prompt quality assessment, reduced turnaround time, and calculation of the first pass yield rate [1, 9].
- 2) Test Level 2 - Integration Test: This test level is technically oriented and focuses on testing multiple modules in interaction with one another. The respective development team is responsible for agreeing on the relevant test cases and interfaces, procedures (tooling), test data, required test environment, and the timing of the test execution [26]. They are also accountable for the results. The findings should be presented to the team for discussion, and a collective decision should be made on the way forward. Both functional and non-functional requirements must be considered [28].

- 3) Test Level 3 - System Test: This test level focuses on a specific domain and involves testing predefined use cases (business cases) of the subsystem. The test designer or test manager of the respective development team is responsible for performing the functional tests, considering any affected interfaces and existing dependencies from the user’s perspective. Both functional and non-functional requirements must be considered. During this phase, it is crucial to intensify direct and regular communication, as well as continuous alignment with the business departments. The results should be presented for discussion with the gatekeeper and other relevant parties, including team members and representatives from the business departments. After this discussion, a joint decision will be made regarding the next steps [4].

- 4) Test Level 4 - System Test: This level of testing involves a comprehensive examination of the entire system. It is business-oriented and focuses on testing all subsystems that form the complete release. The objective is to ensure that these subsystems interact correctly and without errors. The test manager is responsible for conducting these tests, which aim to evaluate the functionality of the interfaces. Both functional and non-functional requirements must be considered. Test managers communicate directly with colleagues from relevant business departments. The selected test environment should closely mimic the production environment. The results are then discussed with the gatekeeper and other relevant stakeholders, including those from development, the test organization, and business departments. A joint decision on how to proceed is made based on these discussions [28].

- 5) Test Level 5 - System Integration Test: This test level is technically oriented and involves a comprehensive integration test of the company’s entire system landscape. It evaluates the software’s behavior in interaction with other existing applications. A successful system integration test should demonstrate that the software has reached sufficient maturity for productive use. Both functional and non-functional requirements must be considered. The results are presented to the gatekeeper and other relevant stakeholders, including program management, development, the test organization, and business units. A joint decision on the optimal course of action is made based on the discussion [29].

- 6) Test Level 6 – Acceptance Test: This test level focuses on the final acceptance of the software. It is technically oriented, ensuring that the already accepted software can be flawlessly installed and operated productively. The system administrator is responsible for successfully completing the commissioning process. This stage concludes with a random sample acceptance test conducted by the designated individual [2].

Additionally, regression tests and smoke tests should be

employed to ensure the integrity of ongoing software development:

- 1) Regression tests should be conducted continuously by the development teams (at the subsystem level) and by the test organization (at the overall system level). The objective is to ensure the continued functionality of existing systems and to identify any new errors introduced by changes to the software [28].
- 2) Smoke tests should be conducted by the development teams (at the subsystem level) and across the test organization (at the overall system level). These tests aim to verify the stability and functionality of the software version in question, whether at the subsystem or overall system level [29].

3.3. Acceptance Procedures Along the Value Chain

The An acceptance procedure along the value chain, coordinated with the current needs of the company and the parties involved, is intended to provide a solution. The following figure illustrates the proposed approach to depicting the acceptance procedure. The results of the empirical study on the acceptance procedure are then discussed. The acceptance procedure establishes a framework that offers participants security, a clear self-image of themselves and their respective teams, an understanding of their tasks, and a clear delineation of existing expectations. Each individual sub-process (quality gate) is considered a measuring point, providing information regarding the respective benefit, the test object, the acceptance criteria, the gatekeeper (owner), the composition of planned meetings, and details about documentation and reports [2].

Measuring points	QG1 Acceptance requirements	QG2 Acceptance of software development	QG3 Acceptance system test	QG4 Acceptance GIT	QG5 Acceptance Release
Benefit	The development team and the Client confirm that the requirements have been understood to the extent that implementation is possible.	Secure new and existing functionalities by means of module and integration tests	Secure the entire lifestream software by means of system tests	Using the GIT (overall integration test), secure the entire system landscape with regard to production readiness. The successful acceptance test is the final activity.	Review and evaluation of the results from the previous Quality Gate (QG4) with regard to the recommendation to go live.
Test item	Individual topics Individual Epics Specifications	Functional requirements (theme, user story, etc...) Non-functional requirements (performance, load test, etc...) Maintenance data Error/Bugs Test preparation for system test incl. test environments	Functional requirements (theme, user story, etc...) Non-functional requirements (performance test, load test, etc...) Maintenance data Error/Bugs Test preparation for the overall integration test Interfaces	Functional requirements (theme, user story, etc...) Non-functional requirements (performance, load test, etc...) Maintenance data Error/Bugs Interfaces	Documents (overall recommendation from QG1 - QG4) Reports
Go/No-Go criterion	Completeness Consistency Feasibility Testability Clarity/ legibility Usability Document properties QG1 must be achieved before the release planning workshop.	Fully implemented and tested (planned tests are 100% completed) functional and non-functional requirements Persons responsible for maintenance data have given their release No Bugs with classification A-Critical and B-High The persons responsible for the system test release the test preparation	Fully implemented and tested (planned tests are 100% completed) functional and non-functional requirements No Bugs with classification A-Critical, B-High and C-Medium No Bugs with the priority "Blocker", or "Critical". The persons responsible for GIT and acceptance test release the test preparation	Fully implemented and tested (planned tests are 100% completed) functional and non-functional requirements as well as interfaces to the surrounding systems No Bugs with classification A-Critical, B-High and C-Medium No Bugs with the priority "Blocker", or "Critical".	Risk assessment through program management
QG- Keeper	QG-Keeper 1: Release Ready for QG2	QG-Keeper 2: Release Ready for QG3 or Ready for QG5	QG-Keeper 3: Release Ready for QG4 or Ready for QG5	QG-Keeper 4: Release Ready for QG5	QG-Keeper 5: Release Go live
QG meeting	Product Owner (PO) Development Teams Representation test organization Client/Subject Matter Expert	Product Owner (PO) Development Teams Representation test organization Department / Customer	Test organization Product Owner (PO) Project Manager	Test organization Departments Topic responsible	Quality management lifestream
Reports/ Docs	Reports are created according to needs and continuously adapted to the purpose	Reports are created according to needs and continuously adapted to the purpose	Reports are created according to needs and continuously adapted to the purpose	Reports are created according to needs and continuously adapted to the purpose	Q-Report
Focus of software quality	Individual requirements Benefits for the customer Direct communication Team composition	Individual functionalities (technically oriented, team supporting) Team results Possible interfaces Stability of the subsystem	Individual subsystems (technically oriented, team-supporting) Overarching team results All subsystems in interaction Verification and validation	All existing systems within the system landscape in the interaction Subject-oriented and product-interrogating (functional)	Stability in production (operation) Technically oriented and product questioning (non-functional)

Figure 4. Acceptance Procedure for the Quality Gates (Source: Own Representation).

The efficacy of each sub-process (quality gate) depends on the outcome of the preceding sub-process. Therefore, it is crucial to ensure enhanced coordination between the individuals responsible for each sub-process (quality gate). In this context, all standard processes must be conducted uniformly to ensure the highest possible result quality. The information provided regarding the focus of each quality gate helps teams achieve the highest possible result quality by reducing or avoiding redundant activities.

Moreover, a concentrated perspective on the quality of outcomes for each sub-process can facilitate expedient, transparent, and well-informed decision-making, as all parties are aware of their responsibilities and the rationale behind them [4].

After presenting the results of the implemented software quality model within the company, this report will discuss and interpret the benefits and limitations of using the software quality model.

4. The Benefits and Limitations of Using the Software Quality Model

Some experts view the software quality model as an effective and sustainable quality process that is practical rather than an end [9]. The process is sufficiently abstract to allow flexibility in its handling without impacting the process or its quality [30]. This flexibility permits necessary changes to be made. It is evident that requirements or value creation must pass through distinct stages and gates [30]. The software quality model offers a high degree of clarity and delineates the specific responsibilities associated with each quality stage [30]. Moreover, it demonstrates the quality attained at each stage of software development, ensuring a certain degree of maturity before proceeding to the next phase. The software quality model employed by the company is regarded as a mature and beneficial approach [9]. It is straightforward and provides a development framework [30]. The model is defined, periodically scrutinized, and revised as necessary, facilitating transparency in software quality development [30].

Some experts indicated that while the absence of a software quality model might expedite task completion, it would inevitably lead to a significant decline in software quality [9]. Without this model, another framework would be necessary to provide an appropriate structure for action [30]. The software quality model offers a comprehensive framework for the entire software development system, detailing the various stages of development [30].

The software quality model enhances coordination across the software development process [9]. Its time-based view facilitates the establishment of coordinated deadlines, enabling all stakeholders to align with them and improving task coordination [30]. The model affords individual employees greater flexibility without imposing inflexible deadlines [30]. It serves as a central system that can be evaluated, traded, and

documented, providing a communication platform [9].

4.1. The Utilization of the Software Quality Model

The experts were queried about how the software quality model is employed within their respective domains and the prerequisites for its implementation.

Requirements for the Utilization of a Software Quality Model: A software quality model must provide a framework for action. Its utilization depends on adherence to a code of conduct, compliance with pertinent regulations, and the enhancement of these regulations [9]. It is essential that all parties involved in the development process recognize the software quality model as a valuable tool, rather than viewing it as a restrictive obligation [30]. Employees must have quality consciousness and a sense of responsibility [30]. There is a need for greater openness to change, especially among new or younger employees who are more likely to be motivated to use the software quality model [9].

Reinforcing and clarifying the framework, as well as reiterating the advantages of the software quality model, would be beneficial [9]. This approach will help avoid future errors [30]. It is imperative that the software quality model be well known to all and that communication regarding it be enhanced [9]. Employees should be aware of the existence of quality gates and their definitions, criteria, and importance [30]. The ramifications of failing to comply with quality gates must be clearly explained [9]. Quality gates should assess the maturity of artifacts rather than impose deadlines, although reaching a quality gate by a designated time is necessary to avoid impacting subsequent gates [30].

Experts view scheduling as beneficial for providing insights and enabling more effective planning [30]. However, adherence to inflexible deadlines can necessitate compromises [9]. It would be prudent to consider redesigning processes for optimal outcomes [9].

A practical software quality model must be straightforward to navigate and offer an optimal user experience [30]. It should be comprehensible to all parties involved, including developers and non-testing personnel, to gain user acceptance [30]. Despite the implementation of a software quality model, procedures across departments remain disparate, often due to prior operational procedures [9]. Creating user acceptance is a significant challenge, especially in well-established areas [9]. The current model is unwieldy due to its production-section orientation and is particularly challenging to use in agile methodologies [30]. Quality criteria are often too vague and subjective [30].

Establishing a culture of shared responsibility for quality at all stages is imperative [9]. Gatekeepers should be accountable for subsequent quality gates [9]. All employees must be included in the process, with an appropriate level of awareness regarding quality and associated processes [30]. Team-

work and communication must be firmly established to foster collaboration across the value chain, preventing fragmentation and ensuring a unified approach to problem-solving [30]. Implementing key performance indicators would be beneficial, as they facilitate drawing pertinent conclusions [9]. The quality of the product can be influenced by various factors, including production methods [9].

Utilization of the Software Quality Model in Everyday Life: Experts were queried about their experiences using the software quality model in everyday life, resulting in a plethora of statements [30].

Applying the software quality model in practice is challenging due to its vague formulations [30]. Its utilization depends on the specific task at hand, but its existence is widely acknowledged and deliberately observed [9]. One interviewee indicated that the model is used primarily for orientation, while for others, it has become an integrated component [9]. Quality gates are queried in a standardized format, and a process is in place for each gate, although this has attracted criticism [30]. Teams should be accountable for their topics and projects [30]. The model is employed to guarantee the functional correctness of the software [9].

The quality model facilitates the pursuit of visionary objectives and maintains a consistent emphasis on software quality [9]. It allows employees to assess their performance against established standards and identify the root causes of issues and their impact on development [30]. The model ensures quality by recording and querying specific software states, preventing the next development phase until criteria are met [30]. This leads to more efficient work, better-equipped teams, and enhanced infrastructure [9]. However, only select phases of the model are employed by some experts [9]. The advantages of the software quality model are not universally acknowledged [30]. It has yet to be sufficiently publicized within the company and is not fully operational, with only select elements implemented [30]. Quality gates are clearly defined, but many employees are unaware of their precise meaning [9]. The software quality model guarantees comprehensive testing, with significant benefits from early testing in the development process, which is more cost-effective than testing only at the end [9].

4.2. Quality Gates

Recent The experts were queried about the challenges they perceive regarding the utility of the existing software quality gates.

The purpose of the quality gates was not evident to the respondents, and there were numerous uncertainties regarding them. A significant challenge lies in effectively communicating the intended meaning and purpose of the quality gates [30]. The software quality model developed by the author employs five quality gates to facilitate the various stages of the software development process. This model enables the identification of the current state of the software or development [30]. Prior to

the implementation of quality gates, the process was notably disorganized, as no definitive assessment could be made regarding the individual stages, which are now systematically incorporated into the quality gates. The quality gates provide a solution to the problems of maturity, their criteria, and the consideration of measuring maturity [30].

The utilization of the software quality model and the quality gates is subject to disparate evaluative criteria in everyday practice. In some cases, the software quality model is important, whereas in others, its necessity arises only when quality gates are not adhered to or when a release is required [9]. Other interviewees indicate that the model is not utilized, or is utilized only minimally, within their respective organizational units. Most cases concern the resolution of production tickets. The focus of the quality gates varies according to the objective to be achieved. The interviews revealed a distinct emphasis on quality gates, accompanied by robust coordination procedures. The overall management of the processes is regarded as positive [9].

Another interviewee posited that the quality gate is often merely a buzzword, with the team, and even the developers themselves, determining whether the quality gate has been reached [30]. In other areas of the company, the team engages in discourse regarding the fulfillment of all requisite criteria for the quality gate, which is to be attained as a primary objective [30]. The decision of whether the quality gate has been reached is made by the team, rather than by individual members [9]. The implementation of quality gates may inadvertently result in a lack of collective responsibility for the overall process. This could lead to an environment where each employee focuses solely on their respective quality gate, working towards its achievement without considering the broader implications of their actions [30]. The consequences of these actions on the remainder of the software are not considered. Even though errors are identified at the quality gates, there is a lack of consideration for the broader process [9]. The software quality model does not account for the following processes [9]. There is a dearth of feedback between the pertinent departments, as well as within the context of software implementations and between the quality gates [30]. This aspect was not considered during the process introduction. A quality gate for communication and interaction is absent [9].

Each gatekeeper is solely accountable for the quality of their respective gate and the associated quality standards, without any inter-gate coordination [9]. These individuals are solely accountable for their respective quality gates, without considering the broader impact of quality on the entire process. In general, there is a lack of shared accountability for all quality gates [30]. The prevailing tendency towards silo thinking and compartmentalization of processes within the divisions may be reinforced by this approach [9]. The quality gates are not observed with consistent regularity in practice [9]. In particular, the tendency to prioritize the issuance of green lights within the individual gates is identified as a po-

tential area for improvement, as this formal signal may not always align with the actual quality of the software [30]. At the quality gates, the requirements set by the preceding developer levels are considered exclusively, without consideration of the requirements set by the developers of subsequent levels [30]. There is a lack of consideration given to the interdependencies of the various functionalities [9]. One of the shortcomings is the rigid delineation between a feature and the comprehensive system [9]. The benefits derived from the implementation of a quality gate model do not appear to outweigh the costs associated with its implementation [30]. The interviewee expressed a preference for more streamlined processes [30]. It is imperative that the content of the delivery be made more accessible to employees in a timely manner, particularly as the handover date approaches [9].

One interviewee observed that the overall testing quality gates are relaxed to accommodate subsequent additions. Another interviewee posited that considering the novel challenges that have emerged, the quality model may require the introduction of new levels or stages. For example, the five quality gates could be further subdivided. The quality gate model would require greater flexibility, such that the removal of a single quality gate from the entire process might be feasible in certain instances. Nevertheless, the cycles would have to be somewhat briefer.

4.3. Challenges in Working with the Software Quality Model

The experts were queried about the challenges they perceive in working with the software quality model. Some experts indicated that the software quality model does not bring as much change as employees commonly assume. In the event of altered technical circumstances, it may be possible to omit certain steps that are not required for the overall integration process. However, many interview participants identified the following challenges:

The experts highlighted several challenges associated with the software quality model, including time pressure and speed, existing complexity, lack of understanding of the purpose of the Quality Gate model, problems with software quality assessment, existing silo thinking, and internal company issues [31]. Additionally, they pointed out agility, communication, flexibility, and modernization as potential challenges from the system architecture [9]. Many of the challenges identified by the experts originate from within the company itself, such as a lack of effective communication [3]. This is evidenced by numerous expert statements describing these challenges.

Ultimately, a significant challenge pertains to the prevailing corporate culture. From the perspective of management, there is a call for enhanced communication regarding the importance of quality. In conclusion, it is imperative that a designated individual or entity assumes responsibility for addressing these challenges. Responsibilities must be subjected to more rigorous regulation.

4.4. The Challenge of Time Pressure and Speed

A significant number of respondents identified time constraints as a major challenge. The interval between software completion and deployment is shrinking, necessitating more efficient management of this period [32]. It is crucial that even minor requirements be implemented promptly, rather than postponed until the next release [33]. The software must be made available to departments more quickly, and the provision of rapid feedback is lacking. Adherence to deadlines by employees is also an issue. The current capacity is insufficient to guarantee the requisite software quality.

This highlights the significant influence of domain expertise on software quality. It is essential that the process of ensuring software quality does not exceed the implementation effort. One potential avenue for enhancing speed is through the modularization of the software. However, modularization also introduces numerous additional interfaces. It is imperative to reduce test cycles and the interval between quality gates while ensuring the software remains of the highest quality despite the need for speed.

4.5. The Challenge of Complexity

The increasing complexity of modern systems presents significant challenges to ensuring software quality. For example, the interaction of functionality with other systems must be considered [34]. To some extent, this complexity can be reduced by automating tests. Additionally, the advent of new technologies has contributed to increased complexity. Consequently, implementing robust internal software quality standards can help mitigate system complexity, as is the case in the company under consideration.

Moreover, the experts highlighted forthcoming changes. In the future, modern products must be able to be mapped, so the architecture and corresponding tools must be kept up to date. Another challenge is the necessity for the company to evolve its technological capabilities. Following a period of relative stagnation and minimal disruption, the company is currently facing several challenges, primarily at the technological level and in relation to finding suitable quality assurance mechanisms. The modernization of the architecture and tools does not impact software quality. An appropriate information technology (IT) architecture is conducive to software quality. These challenges primarily pertain to the time constraints currently prevalent within the organization.

4.6. Existing Silo Thinking

The prevailing approach to software development, characterized by siloed thinking, was identified as a significant obstacle to achieving quality assurance [35]. Prioritizing quality from the outset, rather than focusing solely on feature completion, presents a significant challenge. The entrenched “silo thinking,” reinforced by self-sufficient teams and diverse personnel with disparate roles, must be dismantled [36].

Currently, there is considerable specialization and compartmentalization, with multiple discrete subsystems operating in parallel to ensure overall system functionality. The preexisting separation of locations adds another layer of complexity. It is necessary to integrate various sites and define workflows and responsibilities.

During expert interviews, one expert noted that feedback between responsible areas and within the context of software implementations, as well as between the “quality gates,” is lacking. This is also reflected in the fact that the test organization and the development organization do not form a cohesive unit, lacking joint responsibility for a product or solution. Furthermore, the absence of established or operational standards leads to subjective interpretations, impeding effective collaboration. If the software quality model were accepted, it would result in markedly superior software quality.

The company’s implementation of an agile approach to software development is specifically designed to dismantle siloed thinking and enhance collaboration across business units and locations. In transitioning to agile software development, it is imperative that the organizational structure undergoes transformation, accompanied by the establishment of a new corporate culture. Replacing silo thinking with agility leads to the implementation of new processes. The objective of the software quality model is to ensure that agile methodologies positively impact software quality and eliminate silo thinking.

4.7. Challenge of Quality Assessment and Process Improvement

Another challenge associated with the software quality model pertains to the assessment of quality. It is essential that customers or recipients can discern quality, but according to experts, this is not always the case [2]. There is a lack of insight into how quality is evaluated from an external perspective. Although the company has demonstrated the ability to deliver high-quality software, one interview participant offered a distinct perspective. It is possible for a software product with suboptimal quality to possess the requisite functionality and still meet customer expectations. The company’s software quality is not yet optimal, as evidenced by the prevalence of production errors [9]. Currently, the organization focuses more on identifying errors rather than ensuring quality assurance from the outset. It is necessary to integrate the corresponding processes.

4.8. Transferability and Integration of the Quality Model in Modern Development Environments

To ensure broader applicability of the developed software quality model, its transferability to organizational contexts beyond the insurance industry should be evaluated. Its modular structure and focus on integrating the entire value chain

make it adaptable to various sectors, particularly those undergoing agile transformation. The model is also compatible with modern software development practices such as DevOps, by supporting close collaboration between development (Dev) and IT operations (Ops), as well as promoting rapid feedback cycles, continuous improvement, and automation. Furthermore, the model encourages the integration of CI/CD deployment processes (Continuous Integration (CI), Continuous Delivery (CD)) and automated testing environments. These integrations can enhance the model’s effectiveness by enabling faster releases with consistently high quality, real-time quality feedback, automated validation of artifacts, and seamless deployment pipelines [38]. However, successful implementation of the model in large or highly fragmented organizations requires addressing cultural and organizational challenges. These include fostering transparency, promoting cross-functional collaboration, breaking down silos, and cultivating a shared quality mindset across all levels of hierarchy. Leadership commitment, open communication, and targeted change management strategies are essential to sustainably embed the model in diverse organizational contexts.

5. Key Success Factors for a Software Quality Model

The key success factors for a software quality model are crucial for enabling the successful completion of software development projects in a cost-effective manner. The interview results indicated that software quality is a significant aspect of the software development process within the company and is given high importance [9]. It has become evident that consistently considering software quality throughout the entire value chain of software products is a proven means of ensuring the strategic and economic success of the company [30]. The primary objective of all stakeholders along the value chain is to ensure adequate software quality [30]. However, there are instances when the pursuit of expediency in implementation relegates software quality to a secondary position [9]. It is essential that software quality remains stable and consistently high [9]. Furthermore, the customer’s perspective must be given greater consideration [30]. The individual quality gates are regarded as sub-processes of the value chain [30]. Experts perceive the procedures for fulfilling these sub-processes as lacking flexibility, being excessively time-consuming, and overly rigid [9].

In general, the experts interviewed along the value chain perceive significant potential for cost savings, given that the expenses associated with ensuring software quality are perceived as excessively high [9]. The implementation of software quality assurance measures is often delayed, leading to repeated examinations of test objects across different test stages [30]. Moreover, the current degree of test automation is deemed insufficient, with considerable manual effort required [30].

The measurement and communication of performance, as well as the dissemination of results or partial results, are neither structured nor systematic [9]. There is a paucity of key figures, with some existing ones being unknown [9]. The success of the endeavor is determined exclusively by the subjective assessment of the individuals involved [30]. There is no discourse regarding the quality gates, either individually or comprehensively [30]. The existing individual quality gates are perceived as independent sub-processes, and communication among those involved is oriented toward completing their respective sub-processes [9]. There is a lack of comprehensive joint accountability and planning security regarding software quality and its status [9]. Moreover, from the experts' perspective, the existence of established coordination processes for determining software quality is not always apparent, both within and across quality gates [30]. The lack of defined entry and exit criteria for individual quality gates results in informal agreements between participants [30]. The individual quality gates are designed to ensure adherence to the prescribed steps in the software development process [9]. However, this structure is perceived by experts as a potential limitation to accountability [9]. The defined entry and exit criteria are frequently not met [30]. During the implementation phase, responsibility for software quality is assigned to the software quality model [30]. As a result, the software quality model is not perceived or operationalized as a unified entity [9].

The experts characterize the existing software quality model as an effective, fixed guiding framework for software development, integrating software quality assurance measures in all quality gates [9]. However, the stipulated entry and exit criteria are not adhered to [30]. In practice, the software quality model is typically utilized with a primary focus on fulfilling individual quality gates, with less emphasis on communication between them [30]. The attainment of a quality gate is contingent upon delivery deadlines, not software maturity [9].

The empirical research identified the following central success factors for a software quality model: efficiency, effectiveness, and agility (flexibility) must be ensured [9]; transparency must be maintained throughout the process [30]; key figures that measure the success of the introduction or development process must be objective and comprehensible [30]; and corporate communication must report transparently on quality issues [9]. It is also crucial to consider the involvement of both internal and external stakeholders [9]. The model should be applicable to a range of scenarios, including new launches and the maintenance of existing products [30]. It should also ensure syntactic and semantic consistency [30]. A holistic approach to the model is essential to guarantee the company's long-term success [9]. Furthermore, the company must accept the model [9]. It is also important to establish a culture of error and ensure that the necessary resources (time, money, and expertise) are allocated by top management [30]. If milestones or deadlines are not met, sanctions should be

imposed [30] and measures should be derived that ideally lead to process optimization, using the overall model as an early warning system [9].

6. Criticism of Software Quality Models

Ensuring the production of high-quality software products presents significant challenges. These factors must also be examined in terms of the potential for long-term success for software product manufacturers. A review of the literature revealed a lack of sources providing data on software quality across the entire value chain [9]. Consequently, an additional search was conducted to identify the practices of well-known software houses. Many software companies make information available on their websites or via online search engines. However, this data is largely limited to information about their products, profitability, key figures, organizational structure, and corporate culture [37]. While this information can be collected, processed, and analyzed online, there is no publicly available information regarding the organization of workflows and sub-processes, nor how they are utilized and documented by relevant parties [12]. There is a paucity of information regarding the specific software quality models employed. Therefore, no conclusions can be drawn about the influence of software quality on the entire value chain of software products, regardless of whether the data is publicly available.

The software quality model was the subject of considerable commentary from experts, who identified several shortcomings. For the software quality model to be effective in the context of accelerating change, it must be measurable. It is imperative that the model supports this process rather than impedes it. Additional quality criteria or aspects must be considered to ensure the software continues to function as intended in the future. This may entail partial go-live for component replacement while the software is in operation. It is evident that the existing quality model will be unable to satisfy all future requirements, necessitating the introduction of supplementary elements. The formation of teams within the company because of agile working will have ramifications for software quality and must be considered in the quality model. The current software quality model is overly compartmentalized, making it challenging to achieve collective action. It is no longer aligned with contemporary work practices due to a lack of comprehensive oversight and accountability at the individual level. It is also imperative that the software quality model be compatible with an agile environment.

Moreover, it was observed that the software quality model and the scope of the quality gates lack transparency and exhibit excessive rigidity. This is partly attributable to the way the model was introduced. This issue should have been addressed more systematically from the outset. Improvements must be made in communication. Additionally, there is a notable absence of resolve within the organization to utilize the software quality model uniformly.

Various experts have indicated that the software quality model is unduly restrictive, imposing excessive constraints. Competence is, to some extent, denied to the teams, as a certain order has consistently been observed during software development. The software quality model constrains the flexibility of developers. It is essential to exercise caution and avoid excessive regulation, as this could impede the company's or employees' ability to act freely and take initiative. The perception of reduced responsibility among employees may result from the assumption that another party is assuming responsibility for the task in question.

The software quality model requires a significant investment of time and resources, particularly in terms of coordination efforts. The introduction of quality gates has resulted in increased bureaucratic processes, particularly in terms of formalism and documentation. The software quality model is overly complex, and the development process is unnecessarily lengthy. The potential for optimization is absent. Moreover, the model should have a more comprehensive effect and provide more than quantitative information. Currently, there is a dearth of high-quality statements about software quality.

7. Conclusion

The interconnection between quality models and the definition of software quality has been preliminarily addressed. These models facilitate the concretization of the concept of software quality through structural decomposition. As previously stated, there are several distinct perspectives on software quality: value-based, view-based, product-based, and process-based. However, no quality models have been identified that integrate these disparate perspectives across the entire value chain. Excluding the value chain from consideration prevents the identification of significant weaknesses and problems that impact the company. Considering activities across the value chain enables a detailed examination of the complex concept of software quality, which was absent from the approaches and extensions considered in the literature review. When the entire value chain is considered, activities can be viewed in connection with each other, allowing for the achievement of strategic value based on the conclusions drawn.

Nevertheless, there are numerous challenges associated with implementing software quality models in real-world settings. The utilization of software quality models is prevalent across various industries, where the requisite standards for software products vary considerably. It is thus evident that software quality models must be adapted to align with the specific requirements of each project. The software quality model is adopted in varying ways within the company. There is a lack of consensus regarding the software quality model, often due to a lack of established or operational standards. This makes collaboration more challenging and results in a lack of unified coordination, which is essential for the deployment of software or specific features. Such an outcome would enhance software quality. Additionally, experts have

identified disparate expectations regarding the software quality model. The model's central strengths, as identified by experts, are its comprehensive documentation, standardization, and established presence. These attributes necessitate strict adherence to its specifications by all involved parties. The experts identified rigidity, low flexibility, and a lack of awareness as the primary weaknesses. In general, employees are often un-aware that ensuring software quality also entails planning security measures, which ultimately reduces overall costs within the company. It is imperative that the software quality model be communicated more effectively. There is a lack of an organizational culture that is receptive to change and capable of implementing it.

The prerequisites for utilizing a software quality model, as outlined in the literature, are substantiated by empirical evidence. The company must possess the requisite knowledge and expertise to implement the process. A culture of quality management, accompanied by a willingness and openness from management to secure framework conditions and provide necessary resources, is essential. All participants must be acquainted with and accept the objectives. Such a model must therefore be transparent and comprehensible. This finding was also identified in the literature review. In this context, the literature also identifies the reduction of information overload as a further consideration, though this issue was not identified by the experts as a specific concern.

Both experts and literature agree on the necessity of integrating quality assurance with standard development activities. This implies that errors are identified and re-solved during the development process, rather than only at the customer site. In some cases, this may also result in a reduction in speed or agility, as the necessity for rework on previous releases can cause delays in the release of new versions. Accordingly, it is essential to comprehend the processes that precede and succeed them and to maintain a comprehensive perspective of the process. This is the only way to enable a software quality model to fully realize its advantages, satisfy the demands of agility, and simultaneously enhance quality awareness.

From the author's perspective, there are still numerous issues with the practical application of software quality models and the requisite adaptations (e.g., due to variability across domains and contexts) that have not yet been sufficiently investigated from a scientific standpoint. Many models lack detailed instructions on how to implement practical adaptations. This issue has been addressed by the software quality model in question. It can be further developed, is clearly described, and is easily scalable. It provides a coherent framework for defining and measuring software quality. The model can be implemented in a company-specific manner with minimal effort. The most crucial nodes of the value chain are identified and positioned in relation to one another, allowing for a comprehensive understanding of their respective benefits. Potential success factors and metrics for their measurement are identified. The responsibilities and accountabilities of the relevant parties are made clear. The

communication and decision-making channels are clearly delineated. It is imperative that all parties involved are aware of their shared responsibility for success. A monitoring mechanism and a continuous improvement process are established. Moreover, the software quality model can be utilized as a control instrument and an economic asset within the company. Additionally, it can be integrated into the existing tool landscape of the respective company.

Further research activities should focus on deriving appropriate metrics to achieve desired goals while considering existing stakeholders. The effort to determine the different quantities according to the complexity of the environment can be considerable. Therefore, the use of digitalization and new technological possibilities could help develop new smart solutions. Further research should be carried out in this regard.

Abbreviations

CIO	Chief Information Officer
COCOMO	Constructive Cost Model
IIT	Informatics and Information Technologies
IT	Information Technology
QG	Quality Gate

Acknowledgments

I would like to express my deepest gratitude to Professor D. Atanasova for her support throughout this research.

Author Contributions

Félix Tánolé Conceptualization, Data curation, Formal Analysis, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing - original draft, Writing - review & editing

Desislava Atanasova: Supervision

Funding

The authors declare that they have no funding Support.

Data Availability Statement

Data supporting the findings of this study are available in:
Location: MagentaCloud Share

Link: <https://magentacloud.de/s/PaX87xb8NzEJ4J2>

Password: Publication

Content:

1) Codings

2) Transcriptions

Conflicts of Interest

The authors declare no conflicts of interest.

References

- [1] B. Gezici and A. K. Tarhan, "Systematic literature review on software quality for AI-based software," *Empirical Software Engineering*, vol. 27, article number 66, 2022.
- [2] T. Galli, F. Chiclana and F. Siewe, "Software Product Quality Models, Developments, Trends, and Evaluation," *SN Computer Science*, vol. 1, article number 154, 2020.
- [3] A. Deisenboeck et al., "Software quality models: Exploratory review," *Academia.edu* 2020.
- [4] K. Champion, S. Khatri and B. M. Hill, "Qualities of Quality: A Tertiary Review of Software Quality Measurement Research," *arXiv preprint arXiv: 2107.13687*, 2021.
- [5] V. Lenarduzzi, D. Taibi and C. A. Visaggio, "Software Product Quality Models, Developments, Trends, and Evaluation," *SN Computer Science*, vol. 1, no. 2, 2020.
- [6] C. Izurieta, J. Bieman and A. M. Andrews, "A Review of Software Quality Models for the Evaluation of Software Products," *arXiv preprint arXiv: 1412.2977*, 2020.
- [7] M. Foidl and M. Felderer, "Evolution of Software Quality Models in Context of the Standard ISO," *Software Quality Assurance*, Springer, pp. 123-135, 2014.
- [8] S. Yadav and R. Kishan, "Software Product Quality Models," *Software Quality Assurance*, Springer, pp. 45-67, 2014.
- [9] M. Nistala, S. H. Hemayati and R. Rashidi, "Developers talking about code quality," *Empirical Software Engineering*, vol. 28, no. 5, pp. 10381-10400, 2023.
- [10] S. Yadav and R. Kishan, "Models of Sustainable Software: A Scoping Review," *MDPI*, vol. 14, no. 1, article number 551, 2022.
- [11] J. Elmidaoui, et al., "Systematic literature review on software quality for AI-based software," *Empirical Software Engineering*, vol. 26, no. 2, pp. 10105-10125, 2022.
- [12] ISO/IEC 25010: 2023, "Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model," International Organization for Standardization, 2023.
- [13] S. Hassan, A. Sharma and S. K. Dubey, "Qualities of Quality: A Tertiary Review of Software Quality Measurement Research," *arXiv preprint arXiv: 2107.13687*, 2021.
- [14] M. Akbar, J. Sang, A. Khan, F. Amin and N. Shafiq, "Improving the Quality of Software Development Process by Introducing a New Methodology AZ-Model," *IEEE Access*, 6, pp. 4811-4823, 2018.
- [15] J. Estdale and E. Georgiadou, "Applying the ISO/IEC 25010 Quality Models to Software Product," *Systems, Software and Services Process Improvement*, pp 492-503, 2018.

- [16] J. Granja-Alvarez and M. Barranco-García, "A Method for Estimating Maintenance Cost in a Software Project: A Case Study," *Journal of Software Maintenance: Research and Practice*, 2020.
- [17] S. Döringer, "The problem-centred expert interview: Combining qualitative interviewing approaches for investigating implicit expert knowledge.," *International Journal of Social Research Methodology*, 2021.
- [18] A. Bogner, B. Littig and W. Menz, "Introduction: Expert Interviews — An Introduction to a New Methodological Debate.," *Interviewing Experts*, Palgrave Macmillan, 2021.
- [19] J. Gläser and G. Laudel, "Expert Interviews and Qualitative Content Analysis as Instruments of Research.," Springer, 2020.
- [20] J. Mason, "Qualitative Researching.," Sage Publications, 2021.
- [21] Cambridge University Press, "Why Do We Speak to Experts? Reviving the Strength of the Expert Interview Method.," *Perspectives on Politics*, 2022.
- [22] U. Flick, "An Introduction to Qualitative Research.," Sage Publications, 2022.
- [23] S. Bygren, G. Carrier, T. Maher, P. Maurer, D. Smiley, R. Spiewak and C. Sweed, "Applying the Fundamentals of Quality to Software Acquisition.," MITRE Corporation, 2020.
- [24] M. Jackson, S. Crouch and R. Baxter, "Software Evaluation: Criteria-based Assessment.," *Software Sustainability Institute*, 2021.
- [25] M. A. Filz, S. Gellrich, A. Turetsk, J. Wessel, C. Herrmann and S. Thiede, "Virtual Quality Gates in Manufacturing Systems: Framework, Implementation and Potential," *MDPI: Journal of Manufacturing and Materials Processing*, 4(4), 106, 2020.
- [26] A. Behrens, M. Ofori, C. Noteboom and D. Bishop, "A systematic literature review: how agile is agile project management?," *Issues in Information System*, Vol. 22, Issue 3, pp. 278-295, 2021.
- [27] D. Itzik and G. Roy, "Does agile methodology fit all characteristics of software projects? Review and analysis," *Empirical Software Engineering*, Vol. 28, article number 105, 2023.
- [28] H. Agh, A. Azamnouri and S. Wagner, "Software Product Line Testing: A Systematic Literature Review," *Empirical Software Engineering*, vol. 29, article 146, 2024.
- [29] Bluemke and A. Malanowska, "Software Testing Effort Estimation and Related Problems: A Systematic Literature Review," *ACM Comput. Surv.*, vol. 54, no. 3, article 53, 2021.
- [30] *Software Quality Journal*, "Articles | Software Quality Journal," Springer, <https://link.springer.com/journal/11219/articles> 2024.
- [31] Z. Li, J. Niu and X.-Y. Jing, "Software defect prediction: future directions and challenges," *Automated Software Engineering*, vol. 31, article number 19, 2024.
- [32] M. Kuuttila, M. Mäntylä U. Farooq and M. Claes, "Time Pressure in Software Engineering: A Systematic Literature Review," *arXiv preprint arXiv: 1901.05771*, 2020.
- [33] B. Blagoev, T. Hernes, S. Kunisch and M. Schultz, "Time as a Research Lens: A Conceptual Review and Research Agenda," *Journal of Management*, vol. 50, no. 6, pp. 2152-2196, 2024.
- [34] G. De Angelis, H. Do and B. N. Nguyen, "Introduction to the special issue: Software Quality for Modern Systems," *Journal of Software: Evolution and Process*, vol. 36, no. 2, 2024.
- [35] D. Squirrel and J. Fredrick, "Solving the Problem of Siloed IT in Organizations," *MIT Sloan Management Review*, 2020.
- [36] A. de Waal, M. Weaver, T. Day and B. van der Heij, "Silo-Busting: Overcoming the Greatest Threat to Organizational Performance," *Sustainability*, vol. 11, no. 23, 2019.
- [37] ISO/IEC 25002: 2024, "Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE)," *Quality model overview and usage*, 2024.
- [38] Jun Cui, "The Role of DevOps in Enhancing Enterprise Software Delivery Success through R&D Efficiency and Source Code Management," *arXiv*, 2024. [Online]. Available: <https://arxiv.org/pdf/2411.02209>