KeAi
CHINESE ROOTS
GLOBAL IMPACT

International Journal of Innovation Studies

A Goal–Question–Metric (GQM)-Based Framework for Measuring the Quality of Legacy Software Systems in Organizational Environments

Sufian Khamis, Wajdi Aljedaibi, and Abdullah Algarni
Department of Computer Science, Faculty of Computing & Information Technology
King Abdulaziz University, Jeddah, Saudi Arabia

Abstract

Legacy software systems remain central to many organizations, yet their quality is often assessed through general descriptions rather than systematic measurement. This paper proposes a structured Goal–Question–Metric (GQM)-based framework for measuring the quality of legacy software systems within organizational technical environments. The framework was developed through repeated GQM sessions, workshops, and extended discussions with technical leaders and specialists, with emphasis on objective and quantifiable metrics. It examines legacy software system quality through indicators related to vendor license and support, internal support, programming-language condition, technical-update documentation, business-process alignment, user satisfaction, user complaints, security breaches, documentation types, and fundamental problems. By translating quality assessment into an explicit goal, focused questions, measurable indicators, and interpretable numerical formulations, the proposed framework supports clearer evaluation of whether a legacy software system remains manageable, requires closer monitoring, or represents a quality-related risk. The proposed formulation is validated through case studies using real organizational data from public, private, semi-government, and international contexts. The validation results demonstrate how the quality measure can distinguish acceptable, pre-risk, and risk conditions and support more informed decisions regarding maintenance, modernization, reengineering, or replacement.

Keywords: software engineering, legacy systems, software quality, quality measurement, Goal–Question–Metric (GQM), software metrics.

1. Introduction

Software systems have become essential to the operation of modern organizations and now support critical activities across sectors such as banking, healthcare, transportation, manufacturing, education, and public services (Sommerville; Bennett). As organizations expand and their digital environments become more complex, software systems increasingly shape operational continuity, service quality, and strategic decision-making. Over time, however, many of these

systems do not remain modern. Because of technological evolution, limited vendor support, architectural rigidity, accumulated changes, and outdated implementation environments, software systems gradually move into the legacy domain (Sommerville; Bennett; Ransom, Sommerville, and Warren).

Legacy software systems remain highly relevant because they often continue to support mission-critical organizational functions despite their technical limitations. Yet their continued presence is associated with significant challenges, including security exposure, integration difficulty, high maintenance effort, weak scalability, documentation problems, and dependency on aging technologies and scarce expertise (Crotty and Horrocks; Irani et al.; Hasan et al.). In many organizations, these systems are not peripheral artifacts but deeply embedded operational components whose condition directly influences business continuity, modernization efforts, and digital transformation outcomes.

Recognizing the existence of legacy software systems is not sufficient by itself. Organizations also need to understand whether these systems remain supported, documented, secure, maintainable, aligned with current business processes, and acceptable to users. Without measurement, decision-makers may know that a legacy software system exists but still lack a clear quantitative understanding of its quality condition. As a result, important decisions about maintenance, modernization, reengineering, or replacement may depend more on intuition than on evidence (Dedeke; Ramos, Oliveira, and Anquetil). This creates the main research problem addressed in this paper: the lack of a structured and quantitative framework for measuring the quality of legacy software systems within organizational environments.

For this reason, the present study focuses on measuring the quality of the legacy software system as a foundational part of legacy software system assessment. The paper applies the Goal–Question–Metric (GQM) paradigm as the methodological basis for constructing a structured quality measurement framework. GQM is suitable in this context because it begins with a clearly formulated goal, derives focused questions from that goal, and translates those questions into measurable indicators (Basili).

The quality measurement framework proposed in this paper was developed through repeated GQM sessions, workshops, and extended discussions with technical leaders and experienced specialists from public, private, and semi-government organizations. These sessions supported the formulation of the measurement goal, the generation of corresponding questions, and the identification of metrics grounded in practical organizational realities.

A notable characteristic of the proposed quality measurement framework is that most of its metrics are objective rather than subjective. Objective metrics rely on

facts, counts, states, and numerically observable conditions, and are therefore less dependent on personal interpretation (Basili). This is especially important in the context of legacy software system quality, where subjective impressions alone may obscure the true level of technical weakness, operational misalignment, documentation problems, security exposure, or support limitations. Although one subjective indicator, user satisfaction, remains useful in this study, the proposed framework is intentionally centered on objective and quantifiable metrics in order to support more reliable measurement and clearer decision-making. This orientation also responds to a major limitation identified in the previous literature, where many existing approaches were found to be fragmented, partially qualitative, or overly dependent on subjective indicators (Ramos, Oliveira, and Anquetil).

In addition to defining quality-related metrics, this paper formulates the proposed measures into interpretable numerical expressions and validates them using real organizational data. The validation is applied across public, private, semi-government, and international organizational contexts in order to examine whether the proposed quality measure produces meaningful and sensitive results. This allows the study not only to identify relevant quality indicators, but also to support practical comparison among legacy software systems and to distinguish acceptable, pre-risk, and risk conditions from a quality perspective.

The remainder of this paper is organized as follows. Section 2 presents the background. Section 3 discusses legacy software system quality within the broader measurement framework. Section 4 presents the GQM structure for measuring the quality of the legacy software system, explains the proposed quality metrics, formulates the Legacy Software System Quality measure, and validates the proposed formulation using real organizational data. Section 5 discusses the main findings and implications of the proposed quality measurement framework. Section 6 presents the conclusion and future work.

2. Background

2.1 Legacy Software Systems and Their Organizational Challenges

In modern organizations, software systems form a core part of the technical and operational environment and are used to support a wide range of everyday and strategic activities across different sectors (Sommerville; Bennett). In this context, software systems are no longer merely technical tools; rather, they are deeply embedded in workflows, business processes, communication structures, and decision-making activities (Sommerville). As organizations grow, their software environments also expand through the addition of applications, integrations, patches, and supporting technologies, which makes these environments increasingly complex and tightly coupled (Crotty and Horrocks; Irani et al.).

Over time, however, software systems do not remain modern indefinitely (Sommerville; Ransom, Sommerville, and Warren). As technologies evolve, vendor support weakens, architectures become rigid, and maintenance changes accumulate, many systems gradually become legacy software systems (Sommerville; Dedeke). Prior work has emphasized that legacy software systems are typically systems that continue to support important business functions while relying on outdated technologies, obsolete programming environments, limited support, or historically inherited structures (Sommerville; Ransom, Sommerville, and Warren; Dedeke; Ali et al.).

The literature further shows that legacy software systems remain central in many organizations because they store accumulated business logic, support mission-critical operations, and are deeply integrated with surrounding applications and processes (Dedeke; Sommerville; Ransom, Sommerville, and Warren). Their continued use is reinforced by several factors, including replacement risk, high migration cost, dependence on older technologies, resistance to change, limited documentation, and shortages of specialists who can maintain aging environments (Bennett; Dedeke; Khadka et al.; Ramos, Oliveira, and Anquetil). At the same time, legacy software systems are associated with important quality-related challenges, including security exposure, weak scalability, poor interoperability, architectural inflexibility, performance degradation, limited documentation, and increased maintenance burden (Crotty and Horrocks; Hasan et al.; Ali et al.; Sommerville). These challenges show that legacy software system quality is not only a technical issue, but also an organizational concern that may affect continuity, modernization, and digital transformation.

2.2 Measuring the Quality of Legacy Software Systems

Given this persistent and sometimes hidden influence, organizations need more than a general awareness that legacy software systems exist (Dedeke; Ramos, Oliveira, and Anquetil). They need to understand the actual condition of such systems, including whether they remain supported, maintainable, documented, secure, aligned with current business processes, and acceptable to users. Without measurement, organizations may recognize the existence of legacy software systems while still lacking a clear and quantitative understanding of their quality condition and practical effect (Ramos, Oliveira, and Anquetil).

Measurement is therefore a necessary foundation for sound decision-making in legacy software system management (DeMarco; Lindstrom). In software engineering more generally, measurement provides a way to quantify relevant attributes of software products or processes and to interpret them in relation to organizational goals (Lindstrom). In the context of legacy software system quality, measurement helps organizations identify support limitations, documentation weaknesses, business-process misalignment, security exposure, user

dissatisfaction, and structural problems that may require maintenance, reengineering, modernization, or replacement.

2.3 Goal–Question–Metric (GQM) Paradigm and Related Measurement Limitations

Within this broader measurement context, the Goal–Question–Metric (GQM) paradigm provides a suitable methodological basis for structuring measurement and evaluation (Basili; Caldiera and Rombach). GQM structures measurement around three hierarchical levels: a conceptual level in which a goal is defined, an operational level in which questions are derived from that goal, and a quantitative level in which metrics are identified to answer those questions (Basili; Caldiera and Rombach). This hierarchical relationship among goals, questions, and metrics is illustrated in Figure 2.1. In the present study, the GQM paradigm is applied to measuring the quality of legacy software systems.

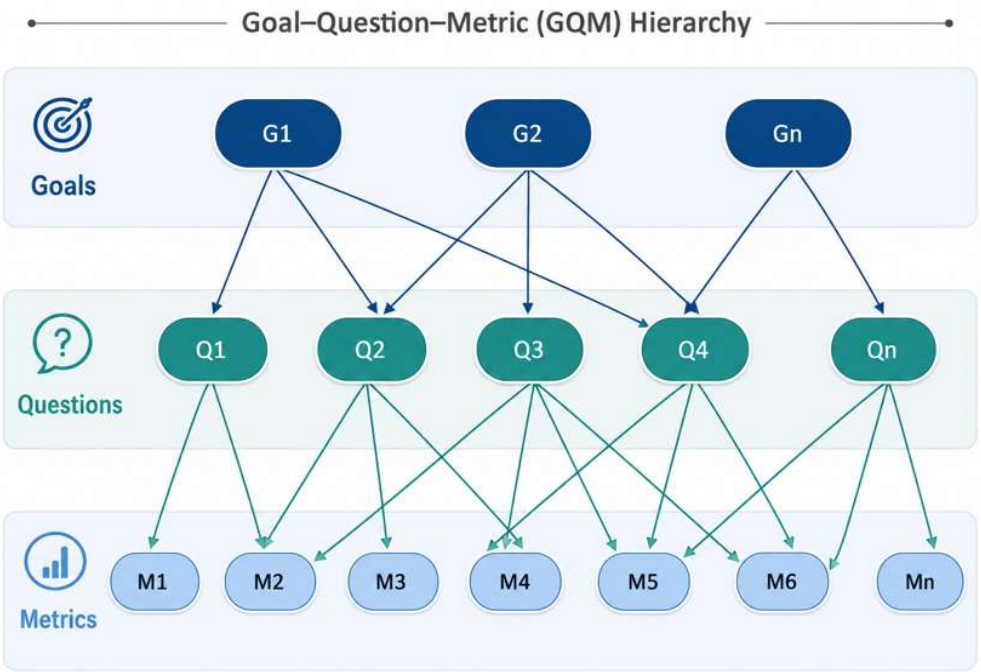

Figure 2.1. Hierarchical structure of the Goal–Question–Metric (GQM) paradigm.

Previous studies also show that attempts to measure legacy software systems remain limited (Ramos, Oliveira, and Anquetil; M'baya, Laval, and Moalla). Existing work has contributed useful ideas, but many approaches address only narrow aspects of legacy software systems, such as maintainability, code analysis, documentation, or outsourcing support conditions (Ramos, Oliveira, and Anquetil). Other studies remain fragmented, depend heavily on qualitative or subjective judgments, or stop short of providing quantitative formulations that can support consistent interpretation across technical environments (Ramos, Oliveira, and Anquetil; M'baya, Laval, and Moalla). These limitations create a clear need

for a more systematic measurement framework that can evaluate the quality of legacy software systems using interpretable and comparable indicators.

For example, De Lucia, Fasolino, and Pompella proposed a decisional framework for legacy system management that considered both business and technical perspectives. Their work aimed to support decisions such as ordinary maintenance, reverse engineering, migration, and system retirement. Although the study referred to the use of the GQM paradigm for predicting the quality profile of a legacy system, the operational details of this use were not fully elaborated. In addition, the framework was mainly decision-oriented and did not provide a final numerical formulation that could be used to calculate and interpret the overall quality condition of a legacy software system in a consistent way.

Similarly, Sneed and Foshag focused on measuring the quality of legacy database structures. Their study contributed objective metrics and mathematical formulations for assessing database-related quality aspects. However, its scope was limited to legacy database structures rather than the broader quality condition of legacy software systems as organizational systems. Therefore, while these studies provide useful foundations, they do not fully address the need for a broader GQM-based quality measurement framework that combines multiple quality-related indicators into interpretable formulations and validates them using organizational case data.

Against this background, the present paper applies the GQM paradigm to organize a set of measurable indicators for evaluating the quality condition of legacy software systems in organizational technical environments. These indicators are then transformed into interpretable formulations and validated using real organizational data.

3. Legacy Software System Quality in the Measurement Framework

Legacy software systems should not be viewed as isolated technical artifacts or as temporary organizational issues. In practice, they are part of a long and continuous technological journey. A software system may begin as a modern system, operate effectively for several years, and then gradually move toward the legacy domain as updates become infrequent, dependencies increase, architectural flexibility declines, and alignment with modern requirements weakens. Prior work has already shown that legacy software systems remain deeply embedded in organizations, continue to support mission-critical operations, and often influence business continuity, modernization efforts, and digital transformation outcomes (Aljedaibi and Khamis, “Assessing Legacy Software Systems”).

For this reason, legacy software systems must first be explored and measured before any sound action can be taken. Before an organization decides whether a

system should be maintained, reengineered, modernized, or replaced, it must first understand the current reality of that system quantitatively. Such a measurement-oriented view is consistent with earlier work, which emphasized the need to measure the depth and breadth of legacy software systems in organizations in order to support informed decisions regarding modernization, reengineering, or replacement (Aljedaibi and Khamis, “Assessing Legacy Software Systems”).

Within this perspective, the present paper focuses specifically on the quality of the legacy software system as one focused measurement area within the broader measurement of legacy software systems. It does not begin with solution selection; rather, it begins with structured assessment. The objective is to establish a measurable view of legacy software system quality using the Goal–Question–Metric (GQM) paradigm. In this paper, the quality of the legacy software system is examined through a dedicated goal, focused questions, measurable metrics, interpretable formulations, and validation using real organizational data.

In the broader measurement structure, legacy software systems can be examined through several major measurement directions, including breadth, depth, risk of slippage toward legacy software systems, software aging coefficient, human technical capability immersion, seasonality, and substitutability. From a measurement perspective, the breadth of legacy software systems can be understood through three main aspects: existence, quality, and user functionality. Existence reflects the degree to which the legacy software system is present and relied upon across the software environment. Quality reflects the condition of the legacy software system. User functionality reflects the extent to which users continue to depend on the legacy software system to perform required tasks and business functions.

Accordingly, quality is positioned in this paper as a focused component of breadth measurement. This positioning is important because the spread of a legacy software system cannot be interpreted correctly without understanding its quality condition. A broadly used legacy software system that remains supported, documented, secure, maintainable, and aligned with current business processes has a different organizational meaning from a broadly used system that lacks support, depends on obsolete technology, suffers from weak documentation, generates user complaints, or has unresolved fundamental problems. Therefore, measuring quality helps reveal whether the presence of the legacy software system represents a manageable condition or a stronger source of technical and operational risk.

The purpose of the following section is therefore to move from this broader measurement context to the specific measurement of legacy software system quality. It presents the GQM structure, metrics, partial indicators, formulation, and

validation cases used to evaluate the quality of a specific legacy software system within its organizational environment.

4. Measuring the Quality of the Legacy Software System

In this study, the quality of the legacy software system is treated as a central measurement dimension within the proposed framework. It focuses on the condition of a specific legacy software system from a quality perspective. The purpose is to assess whether the selected legacy software system remains technically sound, operationally supportable, sufficiently documented, secure, and aligned with current organizational requirements.

More specifically, the quality of the legacy software system refers to the extent to which a specific legacy software system remains supported, technically maintainable, appropriately documented, operationally aligned with current business processes, secure, and acceptable to its users. This definition is consistent with the selected metrics and reflects both technical and operational aspects of software quality. In this sense, quality is not treated as a single property, but as a composite condition shaped by support status, implementation technology, documentation maturity, business-process alignment, user perception, security condition, and the presence or absence of fundamental problems.

Measuring the quality of the legacy software system is important because legacy software systems do not all deteriorate in the same way or at the same rate. Some legacy software systems may still operate with acceptable quality because they remain supported, documented, secure, and functionally aligned with organizational needs. Others may continue to operate despite having accumulated serious quality deficiencies, such as lack of vendor support, obsolete programming languages, undocumented technical updates, user dissatisfaction, repeated security incidents, or unresolved structural problems. For this reason, quality measurement helps distinguish between legacy software systems that remain relatively stable and those that are approaching technically unsafe or operationally unsuitable conditions.

Within the proposed Goal–Question–Metric (GQM)-based framework, the quality of the legacy software system is represented by a dedicated sub-goal, a structured set of questions, and a corresponding collection of measurable indicators. In this regard, Table 4.1 presents the GQM structure adopted for measuring the quality of the legacy software system.

The GQM structure presented in Table 4.1 was derived through iterative GQM sessions, workshops, and extended discussions with IT heads, project managers, technical teams, and experienced specialists from participating organizational contexts. During these sessions, the quality measurement goal was first

formulated, and then a set of operational questions was generated to clarify what should be known about the selected legacy software system. The corresponding metrics were then identified to answer these questions in a measurable way. Emphasis was placed on selecting objective and quantifiable metrics wherever possible.

A notable characteristic of the proposed metric set is that most of its metrics are objective, while one metric—user satisfaction level—is subjective. Nevertheless, this subjective metric remains necessary because quality cannot be fully understood from technical indicators alone. Therefore, the quality assessment in this study intentionally combines technical, operational, and human-centered indicators to produce a more balanced view of system quality.

Sub-goal (Quality)
To analyze the legacy software system for the purpose of measurement with respect to its quality from the perspective of IT heads, project managers, and technical teams in the context of IT environments.
Questions
.1 ?Does the legacy software system have an active external vendor license
.2 Does the legacy software system have active external vendor support?
.3 Does the in-house legacy software system have active internal support?
.4 Is the in-house legacy software system written in an outdated programming language?
.5 Does the in-house legacy software system have documented technical updates?
.6 ?How many business processes are defined for the legacy software system
.7 How many business processes have been executed by the legacy software ?system
.8 ?What is the user satisfaction level with the legacy software system
.9 How many user complaints are related to the functionality of all software ?systems that belong to the same domain
.10 How many user complaints are related to the functionality of the legacy ?software system
.11 How many security breaches are there in all software systems that belong to the same domain?
.12 ?How many security breaches are there in the legacy software system
.13 ?How many documentation types are there for the legacy software system
.14 How many fundamental problems are there within the legacy software system?
Metrics

1.	Whether the legacy software system has an active external vendor license.
2.	Whether the legacy software system has active external vendor support.
3.	Whether the in-house legacy software system has active internal support.
4.	Whether the in-house legacy software system is written in an outdated programming language.
5.	Whether the in-house legacy software system has documented technical updates.
6.	Total number of business processes defined for the legacy software system (TProcDef).
7.	Number of business processes executed by the legacy software system (NLProcExec).
8.	User satisfaction level with the legacy software system (LUsrSat).
9.	Total number of user complaints related to the functionality of all software systems that belong to the same domain (TUsrFuncComp).
10.	Number of user complaints related to the functionality of the legacy software system (NLUsrFuncComp).
11.	Total number of security breaches in all software systems that belong to the same domain (TSecBreach).
12.	Number of security breaches in the legacy software system (NLSecBreach).
13.	Number of documentation types for the legacy software system (NLDocTypes).
14.	Number of fundamental problems within the legacy software system (NLFundProb).

Table 4.1. GQM structure for measuring the quality of the legacy software system.

The proposed metrics provide a multidimensional view of the quality of the legacy software system. Together, they capture legal and vendor-related status, internal technical support, implementation maturity, documentation condition, business-process alignment, user perception, security condition, and deeper structural problems. The following subsections briefly introduce these metrics, explain their importance, and provide the basis for formulating and validating the overall quality measure.

4.1 External Vendor License Status

This metric indicates whether the legacy software system has an active external vendor license at the time of assessment. It is treated as a binary objective indicator and reflects the legal and contractual status of the system from the perspective of vendor authorization. This metric is applicable primarily to legacy software systems acquired from external vendors.

The importance of this metric lies in the fact that licensing status is often one of the first signals of software quality and operational legitimacy. A legacy software system with an active external vendor license is generally in a more stable legal

and technical position than a system operating under an expired or inactive license.

In practical assessment, this metric helps distinguish between systems that remain legally maintained and those that are operating under weakened contractual conditions. A positive value suggests a safer and more manageable quality state, while a negative value indicates increased legal, operational, and technical risk.

This metric is strongly related to external vendor support. When active licensing is combined with active vendor support, the system is typically in a safer and higher-quality condition.

4.2 External Vendor Support Status

This metric indicates whether the legacy software system currently receives active support from its external vendor. It is also treated as a binary objective indicator and reflects whether the system can still benefit from formal external maintenance, issue resolution, and vendor-provided assistance. This metric is applicable primarily to legacy software systems acquired from external vendors.

The importance of this metric is substantial because external vendor support often determines whether the system can be updated, stabilized, corrected, or securely maintained over time. A legacy software system with active vendor support is generally easier to manage and less exposed to long-term technical stagnation than a system for which support has ended.

This metric contributes to the overall measure by capturing how the absence of vendor support increases the likelihood of unresolved failures, outdated components, and higher maintenance difficulty.

This metric is closely related to active external licensing. If a legacy software system has both an active license and active vendor support, it is in a relatively safe and stable condition. If only one of the two is present, the system may be in a partially stable but incomplete state. If both are absent, the system is more likely to indicate poor quality and elevated organizational risk.

For legacy software systems acquired from external vendors, the relationship between external vendor license status and external vendor support status can be expressed through the external vendor-based quality state. This indicator reflects whether the system is still legally and technically covered by the vendor. It is defined as follows:

$$LStExtVend = \begin{cases} 2, & \text{if both the license is active and vendor support is available} \\ 1, & \text{if either the license is active or vendor support is available} \\ 0, & \text{if neither an active license nor vendor support is available} \end{cases}$$

where **LStExtVend** represents the external vendor-based quality state of the legacy software system.
The value of **LStExtVend** belongs to the following set:

$$LStExtVend \in \{0, 1, 2\}$$

A value of **2** represents the best case, where the system has both an active license and active vendor support. A value of **1** represents an intermediate case, where only one of the two elements is available. A value of **0** represents the worst case, where the system has neither an active license nor vendor support.

The rationale behind this scoring is based on three practical quality states: best case, intermediate case, and worst case. This makes **LStExtVend** a useful partial indicator for assessing the quality of externally acquired legacy software systems.

4.3 Internal Support Status of the In-House Legacy Software System

This metric indicates whether an internally developed legacy software system still receives active internal support from developers, technical teams, or maintenance staff within the organization. It is treated as a binary objective indicator. This metric is applicable primarily to in-house legacy software systems developed within the organization.

Its importance lies in the fact that internal support is the main quality safeguard for in-house legacy software systems. If the system is still supported internally, its maintenance condition remains more controlled, more manageable, and relatively safer. In contrast, an internally developed legacy software system that remains operational but no longer has technical ownership or maintenance support becomes more vulnerable to defects, stagnation, and deterioration.

For quality interpretation, this metric helps distinguish between a safer internal quality condition and a weaker internal quality condition. An internally developed legacy software system may still be acceptable if active internal support is available. However, if the same system is no longer supported internally, its continued operation becomes more problematic and indicates lower quality.

This metric is related to programming-language obsolescence and documentation status. If the system has active internal support, uses a manageable technology base, and includes documented technical updates, it is more likely to be in a stable and higher-quality state. If internal support is missing, the negative effect of outdated technology and weak documentation becomes more severe.

4.4 Programming Language Obsolescence in the In-House Legacy Software System

This metric indicates whether the in-house legacy software system is written in an outdated programming language. It is treated as a binary objective indicator and focuses on the technological age and maintainability of the implementation language. This metric is applicable primarily to in-house legacy software systems developed within the organization.

This metric is important because the programming language of a legacy software system influences maintainability, developer availability, tool compatibility, long-term support, and modernization feasibility. Systems written in outdated languages—such as COBOL or older mainframe-related technologies—often require rare expertise and may be more difficult to evolve, integrate, and secure.

Within the proposed formulation, this metric contributes to quality assessment by identifying one of the structural limitations of the system. If the system is written in an outdated programming language, this suggests a lower maintainability position and often a greater long-term dependency risk. If it is not written in such a language, the system may still be legacy for other reasons, but its technical quality from the implementation perspective is relatively better.

4.5 Technical Update Documentation in the In-House Legacy Software System

This metric indicates whether the technical updates applied to the in-house legacy software system are properly documented. It is treated as a binary objective indicator and focuses on whether changes made to the system are accompanied by sufficient technical documentation. This metric is applicable primarily to in-house legacy software systems developed within the organization.

The relevance of this metric lies in the fact that undocumented technical updates reduce software quality by weakening traceability, increasing maintenance difficulty, and creating uncertainty about the actual state of the system. In many legacy environments, updates accumulate over time without proper documentation. As this gap grows, the system becomes harder to understand and more difficult to maintain correctly.

In practical assessment, this metric helps evaluate whether the technical evolution of the system has been managed in a controlled and transparent way. If technical updates are documented, system quality is generally stronger. If updates are undocumented, quality decreases because hidden changes increase the risk of inconsistent design, duplicated structures, weak integration, and future technical errors.

This metric is closely related to internal support and programming-language obsolescence. An in-house legacy software system that has active internal support, is not written in an outdated language, and has documented technical updates is generally in a safer and higher-quality condition.

For in-house legacy software systems, the relationship among internal support, programming language status, and technical update documentation can be expressed through the in-house development-based quality state. This indicator reflects the internal maintainability condition of the legacy software system. It is defined as follows:

$$LStInHouseDev = LIntSupp + LModLang + LDocTecUpd$$

where **LStInHouseDev** represents the in-house development-based quality state of the legacy software system. **LIntSupp** represents internal support status, **LModLang** represents whether the system is written in a modern programming language, and **LDocTecUpd** represents whether technical updates are documented.
The three components are interpreted as binary values:

$$LIntSupp = \begin{cases} 1, & \text{if the system has active internal support} \\ 0, & \text{if the system has no active internal support} \end{cases}$$

$$LModLang = \begin{cases} 1, & \text{if the system is not written in an outdated programming language} \\ 0, & \text{if the system is written in an outdated programming language} \end{cases}$$

$$LDocTecUpd = \begin{cases} 1, & \text{if technical updates are documented} \\ 0, & \text{if technical updates are undocumented or missing} \end{cases}$$

Therefore, the value of **LStInHouseDev** ranges from 0 to 3:

$$LStInHouseDev \in \{0, 1, 2, 3\}$$

A value of **3** represents the best case, where the in-house legacy software system has active internal support, is written in a modern programming language, and has documented technical updates. A value of **0** represents the worst case, where none of these quality elements is available. Values of **1** or **2** represent intermediate conditions.

The rationale behind this scoring is that in-house legacy software quality depends strongly on internal maintainability. A system is in a stronger quality position when it is supported, technically maintainable, and documented. If one or more of these elements are missing, the quality state becomes weaker and may require corrective action.

4.6 Total Number of Business Processes Defined for the Legacy Software System (TProcDef)

TProcDef represents the total number of business processes formally defined for the legacy software system. In this study, a business process refers to a structured set of activities or steps performed by the organization in order to deliver a service, support an operation, or produce a required output.

The importance of this metric lies in its role as a process-reference metric. Software quality cannot be judged only by technical characteristics; it must also be evaluated in relation to the operational processes the system is expected to support. TProcDef therefore provides the baseline process scope against which the actual process execution capability of the system can be interpreted.

For quality interpretation, TProcDef does not by itself indicate whether the system is of high or low quality. Rather, it defines the expected process space of the system. Its significance becomes clearer when interpreted together with NLProcExec, since quality depends in part on how well the system still supports the business processes that it is expected to handle.

4.7 Number of Business Processes Executed by the Legacy Software System (NLProcExec)

NLProcExec represents the number of business processes that are actually executed by the legacy software system in practice. It reflects the operational process capability of the system and its current alignment with business-process requirements.

This metric is useful for identifying whether a legacy software system still supports the current business processes of the organization. If the system continues to support current process requirements and still contributes to operational goals, this indicates stronger quality. Conversely, if it no longer supports the business processes required by the organization, this suggests functional obsolescence and lower quality.

From a measurement perspective, NLProcExec is one of the clearest indicators of whether the system remains relevant to current operations. A higher level of process execution relative to the defined process scope suggests stronger alignment with current business needs. A lower level suggests that the system is losing relevance or failing to support current operational requirements.

This metric is directly related to TProcDef. Together, the two metrics help show whether the system still complies with current business-process requirements. Organizational changes in business processes can reduce the effective quality of

legacy software systems, even when the systems themselves have not changed technically. This makes process alignment an essential part of quality measurement.

The relationship between **TProcDef** and **NLProcExec** can be expressed through the business process execution ratio of the legacy software system. This ratio measures the proportion of defined business processes that are actually executed by the legacy software system. It is defined as follows:

$$LProcExec$$

$$NLProcExec$$

where **LProcExec** represents the business process execution ratio of the legacy software system, **NLProcExec** represents the number of business processes executed by the legacy software system, and **TProcDef** represents the total number of business processes defined for the legacy software system.

The value of **LProcExec** ranges from 0 to 1:

$$0 \leq LProcExec \leq 1$$

A value close to **1** indicates that the legacy software system still executes most or all of its defined business processes. This reflects stronger alignment with current business-process requirements. A value close to **0** indicates that the system executes few or none of its defined business processes, which may suggest functional obsolescence or weak alignment with current organizational needs.

This ratio is important because the quality of a legacy software system is not determined only by technical support or documentation. It also depends on whether the system still supports the business processes it was intended to serve. Therefore, **LProcExec** provides a partial quality indicator from the business-process alignment perspective.

4.8 User Satisfaction Level with the Legacy Software System (LUsrSat)

LUsrSat measures the level of user satisfaction with the legacy software system. This metric is typically collected through surveys, interviews, or related feedback mechanisms, and it may be quantified using a Likert scale or an averaged satisfaction score. A Likert scale is a rating scale commonly used in surveys to capture the degree of agreement, satisfaction, or perception reported by respondents. In this study, a five-point Likert scale is assumed, where 1 indicates very low satisfaction and 5 indicates very high satisfaction.

The inclusion of this metric is justified because it reflects the human and functional dimension of software quality from the viewpoint of the end user. Technical stability alone does not guarantee acceptable quality if users remain dissatisfied with system performance, usability, reliability, or service capability.

For this reason, user satisfaction is an essential complementary indicator in quality assessment.

For quality interpretation, this metric helps capture the extent to which users perceive the system as acceptable, useful, and supportive of their work. High user satisfaction generally indicates stronger perceived quality, while lower satisfaction suggests reservations, persistent problems, or functional limitations. In this sense, the metric captures a user-centered dimension that is not always visible through purely technical indicators.

The user satisfaction level can be transformed into a normalized satisfaction ratio to make the survey results comparable with the other quality-related indicators. Based on this five-point scale, the normalized user satisfaction ratio is defined as follows:

$$LUsrSat = \frac{\sum_{i=1}^{n}(SatScore_i - 1)}{n \times 4}$$

where **LUsrSat** represents the normalized user satisfaction ratio, **SatScore**$_i$ represents the satisfaction score given by respondent *i*, and **n** represents the total number of respondents. The value **4** represents the difference between the maximum and minimum values of the five-point scale.
The value of **LUsrSat** ranges from 0 to 1:

$$0 \leq LUsrSat \leq 1$$

A value close to **1** indicates high user satisfaction with the legacy software system, which reflects a stronger quality condition from the user perspective. A value close to **0** indicates low user satisfaction, which may suggest usability issues, functional limitations, performance concerns, or dissatisfaction with the system's ability to support user needs. Therefore, **LUsrSat** provides a user-centered partial indicator of legacy software system quality.

4.9 Total Number of User Complaints Related to the Functionality of All Software Systems That Belong to the Same Domain (TUsrFuncComp)

TUsrFuncComp represents the total number of user complaints related to system functionality across all software systems that belong to the same domain as the legacy software system. It serves as a domain-level complaint reference metric.

The importance of this metric lies in its contextual role. Complaint counts associated with a single legacy software system become more meaningful when compared with the broader complaint volume across similar systems in the same domain. This provides a more balanced interpretation of whether the complaint

level associated with the legacy software system is relatively low, moderate, or high.

From a measurement perspective, this metric does not directly measure the quality of the legacy software system alone. Rather, it provides the domain context within which system-specific complaints can be interpreted. Its main role is therefore comparative.

4.10 Number of User Complaints Related to the Functionality of the Legacy Software System (NLUsrFuncComp)

NLUsrFuncComp represents the number of user complaints specifically related to the functionality of the legacy software system. These complaints reflect user-reported problems associated with how the system performs or supports required tasks.

This metric is important because one of the practical indicators of software quality is the extent to which users experience functional problems. When no functionality-related complaints are associated with the legacy software system, this supports a more positive quality interpretation. As complaint volume increases, quality generally decreases because the system is failing to satisfy users' functional expectations.

From a measurement perspective, NLUsrFuncComp directly indicates the extent of functional dissatisfaction associated with the system. A low value suggests fewer visible functional problems, whereas a high value indicates recurring or unresolved weaknesses in the system's practical behavior.

The relationship between **TUsrFuncComp** and **NLUsrFuncComp** can be expressed through the functionality-related user complaint ratio of the legacy software system. This ratio measures the proportion of functionality-related complaints associated with the legacy software system compared with the total number of functionality-related complaints across all software systems that belong to the same domain. It is defined as follows:

$$LUsrFuncComp = \frac{NLUsrFuncComp}{TUsrFuncComp}$$

where **LUsrFuncComp** represents the functionality-related user complaint ratio of the legacy software system, **NLUsrFuncComp** represents the number of user complaints related to the functionality of the legacy software system, and **TUsrFuncComp** represents the total number of user complaints related to functionality across all software systems in the same domain.
The value of **LUsrFuncComp** ranges from 0 to 1:

$$0 \leq LUsrFuncComp \leq 1$$

A value close to **0** indicates that few or no functionality-related complaints are associated with the legacy software system, which reflects a better quality condition. A value close to **1** indicates that most or all functionality-related complaints in the domain are associated with the legacy software system, which reflects a weaker quality condition and may indicate functional problems that require corrective action. Therefore, **LUsrFuncComp** provides a complaint-based partial indicator of system quality.

4.11 Total Number of Security Breaches in All Software Systems That Belong to the Same Domain (TSecBreach)

TSecBreach represents the total number of recorded security breaches across all software systems that belong to the same domain as the legacy software system. It provides the domain-level security context for interpreting the security condition of the selected system. This metric captures the need to interpret the security quality of a single legacy software system in relation to the broader security profile of comparable systems in the same domain.

From a measurement perspective, TSecBreach does not directly evaluate the selected legacy software system alone. Instead, it establishes the comparative security baseline against which NLSecBreach can be interpreted. Without such a baseline, the meaning of system-specific breach counts may be incomplete.

4.12 Number of Security Breaches in the Legacy Software System (NLSecBreach)

NLSecBreach represents the number of recorded security breaches associated with the legacy software system. These breaches reflect observable security failures related to the system.

This indicator helps reveal the central role of security in software quality, especially in legacy environments where aging technologies, unsupported components, or weak protections may increase exposure. If no security breaches are recorded for the legacy software system, this suggests stronger quality from a security standpoint. As the number of recorded breaches increases, quality decreases because the system becomes less trustworthy and more vulnerable.

From a measurement perspective, NLSecBreach directly captures the observable security condition of the system. It provides empirical evidence of whether the system remains secure in practice.

This metric is related to TSecBreach because its meaning becomes more informative when interpreted against the broader security-breach context of the

same domain. The relationship between **TSecBreach** and **NLSecBreach** can be expressed through the security breach ratio of the legacy software system. This ratio measures the proportion of recorded security breaches associated with the legacy software system compared with the total number of recorded security breaches across all software systems that belong to the same domain. It is defined as follows:

$$LSecBreach = \frac{NLSecBreach}{TSecBreach}$$

where **LSecBreach** represents the security breach ratio of the legacy software system, **NLSecBreach** represents the number of recorded security breaches in the legacy software system, and **TSecBreach** represents the total number of recorded security breaches across all software systems in the same domain.
The value of **LSecBreach** ranges from 0 to 1:

$$0 \leq LSecBreach \leq 1$$

A value close to **0** indicates that the legacy software system has few or no recorded security breaches, which reflects a safer quality condition from the security perspective. A value close to **1** indicates that most or all security breaches in the domain are associated with the legacy software system, which reflects a higher security concern. Therefore, **LSecBreach** provides a security-based partial indicator of legacy software system quality.

4.13 Number of Documentation Types for the Legacy Software System (NLDocTypes)

NLDocTypes represents the number of documentation types available for the legacy software system. Documentation types may include requirements and analysis documentation, design documentation, system architecture documentation, source code, testing documentation, installation and deployment documentation, user and operational manuals, change logs, API documentation, security documentation, and maintenance documentation.

The relevance of this metric lies in the fact that documentation availability is one of the clearest indicators of maintainability and technical clarity. A legacy software system with richer documentation is generally easier to understand, support, modify, and control than a system with weak or fragmented documentation. Therefore, a higher number of available documentation types usually indicates better quality.

From a measurement perspective, NLDocTypes helps quantify documentation maturity. It does not only reflect whether documentation exists, but also how

broad that documentation is across different technical and operational dimensions of the system.

This metric is related to documented technical updates. The presence of broader documentation types strengthens the interpretability and maintainability of the system, while low documentation availability often coincides with greater technical ambiguity and lower quality.

The documentation coverage ratio measures the extent to which the legacy software system has the expected documentation types. In this study, the maximum expected number of documentation types is **11**. Therefore, the ratio is defined as follows:

$$LDocTypes = \frac{NLDocTypes}{11}$$

where **LDocTypes** represents the documentation coverage ratio of the legacy software system, **NLDocTypes** represents the number of documentation types available for the legacy software system, and **11** represents the maximum expected number of documentation types.
The value of **LDocTypes** ranges from 0 to 1:

$$0 \leq LDocTypes \leq 1$$

A value close to **1** indicates that most or all expected documentation types are available, which reflects stronger documentation maturity and better maintainability. A value close to **0** indicates that documentation is weak or missing, which may increase maintenance difficulty and reduce technical clarity. Therefore, **LDocTypes** provides a documentation-based partial indicator of legacy software system quality.

4.14 Number of Fundamental Problems within the Legacy Software System (NLFundProb)

NLFundProb represents the number of major structural and operational problems present within the legacy software system. In this study, such problems may include incompatibility with other software systems, maintenance challenges, scalability challenges, performance bottlenecks and latency issues, high downtime and system failures, high operational costs, vendor lock-in and unsupported technologies, and data management complexity.

This metric is useful for identifying deeper quality weaknesses that are not always visible through isolated indicators alone. While support, documentation, security, and complaint metrics reveal specific quality dimensions, NLFundProb

summarizes whether the system suffers from broader and more persistent technical and organizational difficulties.

From a measurement perspective, a higher number of fundamental problems indicates lower software quality because it suggests the presence of multiple structural weaknesses that reduce system reliability, maintainability, and long-term sustainability. A lower number suggests a more stable and manageable system condition.

This metric may be related to several other metrics in this section, such as support, technology, documentation, complaints, and security. For this reason, NLFundProb can be viewed as one of the strongest summary indicators of deeper quality deterioration within the legacy software system.
The fundamental problem ratio measures the extent to which the legacy software system suffers from major structural or operational problems. In this study, the maximum expected number of fundamental problem types is **8**. Therefore, the ratio is defined as follows:

$$\boldsymbol{LFundProb = \frac{NLFundProb}{8}}$$

where **LFundProb** represents the fundamental problem ratio of the legacy software system, **NLFundProb** represents the number of fundamental problems identified within the legacy software system, and **8** represents the maximum expected number of fundamental problem types.
The value of **LFundProb** ranges from 0 to 1:

$$0 \leq LFundProb \leq 1$$

A value close to **0** indicates that the legacy software system has few or no fundamental problems, which reflects a stronger quality condition. A value close to **1** indicates that the system suffers from many fundamental problems, which reflects deeper quality deterioration and may require modernization, reengineering, or replacement planning. Therefore, **LFundProb** provides a problem-based partial indicator of legacy software system quality.

Taken together, the proposed metrics provide a structured and multidimensional view of the quality of the legacy software system. This broad measurement perspective is important because the quality of a legacy software system cannot be captured by a single indicator, but should be understood through multiple interacting technical, organizational, and user-centered conditions.

4.15 Formulation of the Quality of the Legacy Software System

The formulation of Breadth (Quality) aims to measure the quality condition of a specific legacy software system within its operational environment. Unlike the existence-related measures, this formulation does not measure how widely or

strongly the system exists, but rather how acceptable its quality condition is from technical, operational, security, documentation, and user perspectives. Therefore, this formulation is applied to one legacy software system at a time.

Because legacy software systems may have different origins, two formulations are used. The first formulation is applied when the legacy software system is acquired from an external vendor. The second formulation is applied when the legacy software system is developed in-house. This distinction is important because externally acquired systems depend mainly on vendor license and vendor support, while in-house systems depend mainly on internal support, modern programming language, and technical update documentation.

For an externally acquired legacy software system, **Breadth (Quality)** is expressed as follows:

$$Breadth\ (Quality)_V = (LStExtVend + LProcExec + LUsrSat + LDocTypes) - (LUsrFuncComp + LSecBreach + LFundProb)$$

where **Breadth (Quality)$_V$** represents the quality score of a legacy software system acquired from an external vendor. **LStExtVend** represents the external vendor-based quality state. **LProcExec** represents the business process execution ratio. **LUsrSat** represents the user satisfaction ratio. **LDocTypes** represents the documentation coverage ratio. **LUsrFuncComp** represents the functionality-related user complaint ratio. **LSecBreach** represents the security breach ratio. **LFundProb** represents the fundamental problem ratio.

For an in-house developed legacy software system, **Breadth (Quality)** is expressed as follows:

$$Breadth\ (Quality)_I = (LStInHouseDev + LProcExec + LUsrSat + LDocTypes) - (LUsrFuncComp + LSecBreach + LFundProb)$$

where **Breadth (Quality)$_I$** represents the quality score of an in-house developed legacy software system. **LStInHouseDev** represents the in-house development-based quality state, while the remaining components have the same meaning as in the vendor-based formulation.

The logic of the formulation is based on the distinction between positive and negative quality indicators. Positive indicators are added because higher values indicate better quality. These include vendor or internal development state, business-process execution, user satisfaction, and documentation coverage. In contrast, negative indicators are subtracted because higher values indicate weaker quality. These include user complaints, security breaches, and fundamental problems. In this way, the final score increases when the system has stronger quality characteristics and decreases when quality problems become more visible. The possible range differs according to the source of the system. For externally acquired legacy software systems, the maximum value is obtained when all positive quality indicators are at their highest values and all negative quality indicators are zero:

$$-3 \leq Breadth(Quality)_V \leq 5$$

For in-house developed legacy software systems, the maximum value is higher because the in-house development-based quality state has three possible positive components:

$$-3 \leq Breadth(Quality)_I \leq 6$$

A higher value indicates stronger quality, better support, stronger business-process alignment, higher user satisfaction, better documentation, fewer complaints, fewer security breaches, and fewer fundamental problems. A lower value indicates weaker quality and a greater need for modernization, reengineering, or replacement planning.

Since the two formulations have different original ranges, normalization is needed in the validation stage to map the results into a common scale. After normalization, values closer to **10** indicate stronger quality, while values closer to **0** indicate weaker quality.

For practical interpretation before normalization, the quality value can be understood according to the general classification shown in Table 4.15.1.

Breadth (Quality) Value	**Interpretation**
Low value	Poor quality condition. The system lacks several important quality indicators and may require modernization, reengineering, or replacement planning.
Medium value	Moderate quality condition. The system satisfies some quality indicators but still shows weaknesses that require monitoring and improvement.
High value	Strong quality condition. The system satisfies most or all quality indicators and remains relatively aligned with technical, operational, security, documentation, and user needs.

Table 4.15.1. Interpretation of Breadth (Quality) values.

Thus, **Breadth (Quality)** provides a system-level indicator of the quality condition of a legacy software system. It transforms multiple quality-related

indicators into a single measurable expression that can support prioritization and decision-making.

4.16 Validation and Case Study of the Quality of the Legacy Software System

The purpose of this validation is to increase confidence in the proposed formulation of the quality of the legacy software system. The validation aims to examine whether the formulation produces meaningful and sensitive results when applied to real organizational data. In this context, sensitivity means that the result should increase when positive quality indicators improve and decrease when negative quality indicators become stronger.

To support this validation, the formulation was applied to legacy software systems drawn from the same organizational contexts used in the previous validation stages. The same symbolic codes were used to preserve anonymity and maintain consistency. **PUB-D1** and **PUB-D2** refer to two departments in a large public educational organization. **INT-ORG** refers to an international organization. **SEM-GOV** refers to a semi-government organization. **PRI-TEL** refers to a private telecommunications organization.

Since the original range of **Breadth (Quality)** differs according to the source of the legacy software system, normalization was applied to convert all results into a common scale from 0 to 10. For externally acquired legacy software systems, the original range is from **-3 to 5**. For in-house developed legacy software systems, the original range is from **-3 to 6**. The normalized value is calculated as follows:

$$Z_i = \left(\frac{MAXNUM - MINNUM}{max(x) - min(x)} \right) \times (X_i - min(x)) + MINNUM$$

where X_i is the original value of Breadth (Quality), *min(x)* and *max(x)* represent the minimum and maximum possible values of the original scale, and *MINNUM* and *MAXNUM* represent the lower and upper bounds of the normalized scale. In this study, the normalized scale ranges from 0 to 10.

A higher normalized value in **Breadth (Quality)** indicates a better condition. Therefore, values close to 10 indicate stronger quality, while values close to 0 indicate weaker quality. This difference is important because the quality formulation is designed to reward positive indicators, such as support, documentation, process execution, and user satisfaction, and penalize negative indicators, such as complaints, security breaches, and fundamental problems.

For interpretation, three threshold zones were used: red, yellow, and green. The red zone represents a risk condition, the yellow zone represents a pre-risk

condition, and the green zone represents an acceptable condition. Table 4.16.1 presents the normalized interpretation thresholds used to classify Breadth (Quality) values into red, yellow, and green decision zones.

The threshold values in Table 4.16.1 were defined to support practical interpretation of the normalized quality score and should be understood as practical decision-support thresholds for this study. Since the normalized scale ranges from 0 to 10, the values were divided into three decision-oriented zones that reflect weak, moderate, and strong quality conditions. This distribution was guided by expert judgment obtained during the GQM sessions and workshops, together with the expected interpretation of the quality formulation. Because a higher Breadth (Quality) value indicates a stronger quality condition, values from 0 to 4 were classified as the red zone to represent weak quality and the need for immediate attention. Values from 4.01 to 7 were classified as the yellow zone to represent a transitional pre-risk condition that still requires monitoring and improvement. Values from 7.01 to 10 were classified as the green zone to represent a stronger and more acceptable quality condition. This interpretation is therefore intended to provide a practical decision-support scale for comparing systems and identifying those that require quality improvement, modernization planning, or replacement consideration.

Normalized Breadth (Quality) Range	**Zone**	**Interpretation**	**Required Attention**
0–4	Red zone	Risk condition. The legacy software system has weak quality.	Immediate action, major updates, or replacement planning
4.01–7	Yellow zone	Pre-risk condition. The legacy software system has moderate quality and requires improvement.	Careful monitoring and quality improvement planning

7.01–10	Green zone	Acceptable condition. The legacy software system has strong quality.	Normal monitoring and continued quality control

Table 4.16.1. Normalized interpretation thresholds for Breadth (Quality).

The formulation was applied to thirteen individual legacy software systems drawn from the five organizational cases. In **PUB-D1**, six systems were measured. In **PUB-D2**, four systems were measured. In addition, one main legacy software system was measured in each of **INT-ORG**, **SEM-GOV**, and **PRI-TEL**. Table 4.16.2 presents the validation results of **Breadth (Quality)** across the examined legacy software systems. The table reports the positive and negative quality ratios, the calculated **Breadth (Quality)** value, the normalized value on a 0–10 scale, and the corresponding interpretation zone. Figure 4.16.1 further illustrates the normalized **Breadth (Quality)** results and highlights the classification of each system into the red, yellow, or green zone.

Case Code	Software System	Source	LStExtVend / LStInHouseDev	LProcExec	LUsrSat	LDocTypes	LUsrFuncComp	LSecBreach	LFundProb	Breadth (Quality)	Normalized Value 0–10	Zone
PUB-D1	System A	In-house	2	0.76	1.00	0.27	0.07	0.75	0.00	3.22	6.91	Yellow
	System B	In-house	2	0.88	0.50	0.18	0.03	0.00	0.13	3.40	7.11	Green
	System C	In-house	2	0.73	0.75	0.09	0.06	0.25	0.00	3.27	6.96	Yellow
	System D	External vendor	0	0.82	0.75	0.82	0.22	0.00	0.50	1.66	5.83	Yellow
	System E	External vendor	0	1.00	0.75	0.18	0.11	0.00	0.13	1.69	5.87	Yellow

	System F	In-house	2	0.87	1.00	0.45	0.51	0.00	0.25	3.57	7.30	Green
PUB-D2	System G	In-house	1	0.75	0.75	0.09	0.89	0.00	0.75	0.95	4.39	Yellow
	System H	In-house	2	0.80	0.50	0.18	0.04	1.00	0.13	2.32	5.91	Yellow
	System I	In-house	1	0.53	0.50	0.09	0.08	0.00	0.50	1.55	5.05	Yellow
	System J	In-house	1	0.80	0.75	0.18	0.00	0.00	0.00	2.73	6.37	Yellow
INT-ORG	Software System	External vendor	1	0.50	0.00	0.18	0.30	0.00	0.88	0.51	4.38	Yellow
SEM-GOV	Software System	External vendor	2	0.50	0.00	0.00	0.25	0.00	0.75	1.50	5.63	Yellow
PRI-TEL	Software System	External vendor	1	0.75	0.75	0.55	0.71	0.00	0.38	1.96	6.20	Yellow

Table 4.16.2. Validation results of Breadth (Quality) across legacy software systems.

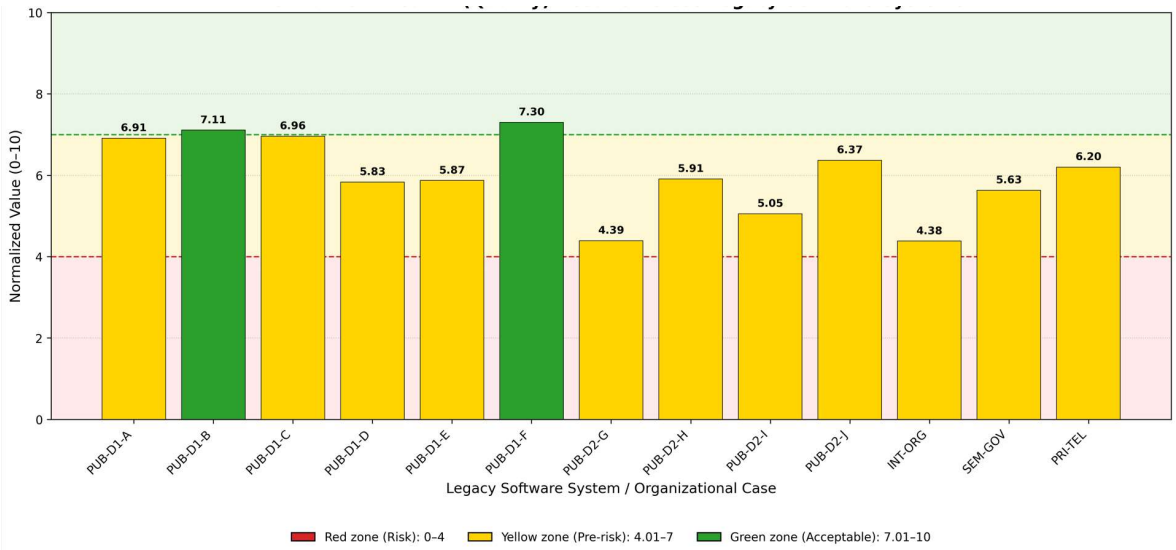

Figure 4.16.1. Normalized Breadth (Quality) results across legacy software systems.

The results show that most examined systems fall within the yellow zone, which indicates moderate quality and a need for careful monitoring and improvement. In **PUB-D1**, most systems are in the yellow zone, except **System B** and **System F**, which fall within the green zone. **System B** shows a stronger quality condition because it has internal support, uses a modern programming language, executes a high proportion of its business processes, has low complaint levels, has no recorded security breaches, and has few fundamental problems. **System F** also shows strong quality because it combines internal support, modern implementation, high business-process execution, high user satisfaction, no recorded security breaches, and relatively limited fundamental problems.

Systems **D** and **E** in **PUB-D1** are externally acquired systems and fall within the yellow zone. Although they show acceptable values in some indicators, their external vendor-based quality state is weak because they lack active vendor license and support. This reduces their overall quality score and shows the importance of vendor coverage for externally acquired legacy software systems.

In **PUB-D2**, all systems fall within the yellow zone. However, the results still require attention. **System G** is affected by a high functionality-related complaint ratio and a high fundamental problem ratio. **System H** has a strong internal state and acceptable process execution, but its security breach ratio weakens its quality score. **System I** is affected by lower process execution, moderate user satisfaction, limited documentation, and fundamental problems. **System J** is the strongest among the PUB-D2 systems, mainly because it has low complaints, no security breaches, and no fundamental problems, but it remains in the yellow zone because its internal quality state and documentation coverage are not high enough to move it into the green zone.

The three systems from **INT-ORG**, **SEM-GOV**, and **PRI-TEL** also fall within the yellow zone. The **INT-ORG** system has a low score because of missing vendor support, low business-process execution, very low user satisfaction, weak documentation coverage, and a high fundamental problem ratio. The **SEM-GOV** system benefits from stronger vendor-based coverage, but its score is reduced by low user satisfaction, no available documentation types, and several fundamental problems. The **PRI-TEL** system has the highest quality score among these three external-vendor systems, but it remains in the yellow zone because of missing vendor support, a high complaint ratio, and the presence of fundamental problems.

The concentration of most systems in the yellow zone suggests that the quality issue is not always represented by immediate failure or complete deterioration. Rather, it often appears as a gradual quality degradation risk. This means that several legacy software systems may still be operationally usable, but they already show weaknesses that require early attention. If these weaknesses are not addressed through quality improvement, documentation enhancement, support renewal, security control, or modernization planning, these systems may gradually move toward the red zone.

Overall, the validation results show that the proposed formulation is meaningful and sensitive to differences in quality-related indicators. Systems with stronger support, better business-process execution, higher satisfaction, stronger documentation, fewer complaints, fewer security breaches, and fewer fundamental problems receive higher scores. In contrast, systems with missing support, weak documentation, high complaint levels, security issues, or fundamental problems receive lower scores. Therefore, **Breadth (Quality)** can be used as a practical system-level indicator for identifying legacy software systems that require quality improvement, modernization planning, or replacement consideration.

5. Discussion

The proposed quality measurement framework shows that the quality of a legacy software system should not be assessed through a single isolated indicator. Rather, it should be examined through a set of complementary indicators that together reveal the technical, operational, organizational, and user-related condition of the system. In this study, the application of the Goal–Question–Metric (GQM) paradigm led to a structured quality measurement framework that covers several important aspects, including vendor license status, vendor support, internal support, programming-language condition, technical-update documentation, business-process alignment, user satisfaction, user complaints, security breaches, documentation types, and fundamental problems.

One important outcome of this study is that it makes the quality condition of legacy software systems more measurable in organizational environments. Instead of relying only on general impressions such as whether a system is "old," "difficult," or "poorly maintained," the proposed framework translates quality assessment into an explicit goal, focused questions, measurable indicators, and interpretable formulations. This is important because many organizations already know that legacy software systems exist, yet still lack a structured basis for understanding whether these systems remain supported, documented, secure, aligned with current business processes, and acceptable to users. The framework therefore supports movement from informal judgment to organized quality measurement.

The discussion also shows that legacy software system quality is a composite condition rather than a single technical property. A legacy software system may be old but still remain in an acceptable quality condition if it is supported, documented, secure, technically maintainable, and aligned with current business processes. Conversely, a system may continue to operate while suffering from serious quality weaknesses such as lack of vendor support, missing internal support, obsolete implementation technology, undocumented technical updates, user dissatisfaction, repeated complaints, security incidents, weak documentation, or unresolved fundamental problems. For this reason, quality should be interpreted through the combined effect of multiple indicators rather than through one metric alone.

The validation cases further strengthen the practical value of the proposed framework. By applying the quality formulation to real organizational data from public, private, semi-government, and international contexts, the study shows that the proposed quality measure can produce meaningful distinctions among legacy software systems. The use of normalized scales and threshold zones also supports clearer interpretation by distinguishing acceptable, pre-risk, and risk conditions from a quality perspective. This does not mean that the numerical value should replace expert judgment; rather, it provides structured evidence that can guide expert interpretation and support more transparent decisions about maintenance, modernization, reengineering, or replacement.

Overall, the findings of this study indicate that the quality of legacy software systems can be examined more effectively when measurement is structured, multidimensional, and explicitly tied to organizational goals. The proposed framework contributes by moving legacy software system quality assessment away from vague description and toward a clearer measurement-oriented perspective. It offers a practical basis for identifying whether a legacy software system remains manageable, requires closer monitoring, or needs stronger modernization, reengineering, or replacement consideration. In this sense, the value of the proposed framework lies not only in the number of metrics it

introduces, but in the way those metrics are organized, formulated, validated, and interpreted to support clearer decision-making.

6. Conclusion and Future Work

This paper proposes a structured Goal–Question–Metric (GQM)-based framework for measuring the quality of legacy software systems within organizational technical environments. The framework was developed in response to the need for a more systematic, objective, and interpretable way to assess the quality condition of legacy software systems. Rather than treating quality as a general impression or evaluating it through isolated indicators, the proposed framework organizes quality measurement through an explicit goal, focused questions, measurable metrics, interpretable formulations, and validation cases.

The proposed framework examines legacy software system quality through several indicators, including vendor license status, vendor support, internal support, programming-language condition, technical-update documentation, business-process alignment, user satisfaction, user complaints, security breaches, documentation types, and fundamental problems. Together, these indicators provide a multidimensional view of whether a legacy software system remains supported, maintainable, documented, secure, aligned with current business processes, and acceptable to users.

One of the main contributions of this work is that it moves the assessment of legacy software system quality from general technical impressions toward a clearer measurement-oriented perspective. The proposed metrics were translated into numerical formulations that support consistent interpretation. The study also validated the proposed formulation using real organizational data from public, private, semi-government, and international contexts. This validation showed that the quality measure can distinguish among acceptable, pre-risk, and risk conditions and support comparison across individual legacy software systems.

Overall, the findings of this study indicate that the quality of legacy software systems can be examined more effectively when measurement is structured, multidimensional, validated, and explicitly tied to organizational goals. The proposed framework offers a practical basis for identifying whether a legacy software system remains manageable, requires closer monitoring, or needs stronger consideration for maintenance, modernization, reengineering, or replacement.

Future work may focus on implementing the proposed quality measurement framework as an interactive dashboard, developing recommendation and action-planning mechanisms based on the measurement results, and extending the

broader measurement of legacy software systems to other dimensions such as existence, user functionality, depth, slippage risk, software aging, human technical capability immersion, seasonality, and substitutability.

Acknowledgements

The project was funded by KAU Endowment (WAQF) at King Abdulaziz University, Jeddah, Saudi Arabia. The authors, therefore, acknowledge with thanks WAQF and the Deanship of Scientific Research (DSR) for technical and financial support.

References

- Ali, Muhammad, et al. “Addressing Software Related Issues on Legacy Systems – A Review.” *International Journal of Scientific and Technology Research*, vol. 9, no. 3, 2020, pp. 3738–3742.
- Aljedaibi, Wajdi, and Sufian Khamis. "Assessing Legacy Software Systems: Existence, Challenges, Impact, and Migration Strategies in Organizational Contexts." *Communications in Mathematics and Applications*, Vol. 17, No. 1, 2026.
- Aljedaibi, Wajdi, and Sufian Khamis. "Large-scale Software Critical Success Factors: Watch Them Live!." (2021).
- Aljedaibi, Wajdi, and Sufian Khamis. “Top Management Support during Large-scale Software Systems Implementations: Important Issues in Management Process.” *Indian Journal of Science and Technology*, vol. 12, no. 25, July 2019, pp. 1–11. DOI: 10.17485/ijst/2019/v12i25/145744.
- Aljedaibi, Wajdi, and Sufian Khamis. "Towards Measuring the Project Management Process During Large Scale Software System Implementation Phase." *ISeCure* 11.3 (2019): 161.
- Alkazemi, Basem Y., Mohammed K. Nour, and Abdulqader Qada Meelud. "Towards a framework to assess legacy systems." *2013 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, 2013.
- Bakar, Humairath K. M. Abu, Rozilawati Razali, and Dian Indrayani Jambari. “Implementation Phases in Modernisation of Legacy Systems.” 2019 6th International Conference on Research and Innovation in Information Systems (ICRIIS), IEEE, 2019.
- Basili, Victor R. “Software Modeling and Measurement: The Goal/Question/Metric Paradigm.” *Technical Report*, University of Maryland, 1992.
- Bennett, Keith. “Legacy Systems: Coping with Success.” *IEEE Software*, vol. 12, no. 1, 1995, pp. 19–23.
- Caldiera, Victor R. B. G., and H. Dieter Rombach. “The Goal Question Metric Paradigm.” *Encyclopedia of Software Engineering*, Wiley, 1994, pp. 528–532.

- Crotty, J., and I. Horrocks. "Managing Legacy System Costs: A Case Study of a Meta-Assessment Model to Identify Solutions in a Large Financial Services Company." *Applied Computing and Informatics*, vol. 13, no. 2, 2017, pp. 175–183.
- Dedeke, A. "Improving Legacy-System Sustainability: A Systematic Approach." *IT Professional*, vol. 14, no. 1, 2012, pp. 38–43.
- De Lucia, Andrea, A. R. Fasolino, and E. Pompella. "A Decisional Framework for Legacy System Management." *Proceedings of the IEEE International Conference on Software Maintenance (ICSM 2001)*, IEEE, 2001, pp. 642–651.
- DeMarco, Tom. *Controlling Software Projects*. Yourdon Press, 1986.
- Hasan, M. H., et al. "Legacy Systems to Cloud Migration: A Review from the Architectural Perspective." *Journal of Systems and Software*, vol. 202, 2023, p. 111702.
- Irani, Zahir, et al. "The Impact of Legacy Systems on Digital Transformation in European Public Administration: Lessons Learned from a Multi-Case Analysis." *Government Information Quarterly*, vol. 39, no. 2, 2022, article 101613.
- Khadka, Ravi, et al. "How Do Professionals Perceive Legacy Systems and Software Modernization?" *Proceedings of the 36th International Conference on Software Engineering (ICSE '14)*, ACM, 2014, pp. 1–10.
- Khamis, Sufian, and Wajdi Aljedaibi. "A Framework for Measuring Critical Success Factors of Large-Scale Software Systems." *International Journal of Computer Engineering and Technology*, vol. 7, no. 6, 2016, pp. 71–82.
- Khamis, Sufian, and Wajdi Aljedaibi. "On the Measurement of Data Accuracy During Large-Scale Software Systems Implementations." *Bioscience Biotechnology Research Communications*, vol. 12, no. 1 (Special Issue), Jan. 2019, pp. 143–156, doi:10.21786/bbrc/12.1/17.
- Lindstrom, B. *A Software Measurement Case Study Using GQM*. Lund University, 2004.
- M'baya, A., J. Laval, and N. Moalla. "An Assessment Conceptual Framework for the Modernization of Legacy Systems." *Proceedings of the 11th International Conference on Software, Knowledge, Information Management and Applications (SKIMA 2017)*, IEEE, 2017, pp. 1–11.
- Ramos, Carlos S., Káthia M. Oliveira, and Nicolas Anquetil. "Legacy Software Evaluation Model for Outsourced Maintainers." *Proceedings of the Eighth European Conference on Software Maintenance and Reengineering (CSMR 2004)*, IEEE, 2004, pp. 48–57.
- Ransom, John, Ian Sommerville, and Ian Warren. "A Method for Assessing Legacy Systems for Evolution." *Proceedings of the Second Euromicro Conference on Software Maintenance and Reengineering*, IEEE, 1998, pp. 128–134.

- Sommerville, Ian. *Software Engineering*. 10th ed., Pearson, 2015.
- Sneed, Harry M., and Oliver Foshag. "Measuring Legacy Database Structures." Software Measurement: Current Trends in Research and Practice, Deutscher Universitätsverlag, 1999, pp. 143–158.