Middle East Technical University
Informatics Institute

Cost of Quality Analysis for an Embedded Communication Project

Advisor Name: Assist. Prof. Dr. Özden ÖZCAN TOP
(METU)

Student Name: Engin Kılınç
(Information Systems)

January 2026

TECHNICAL REPORT

METU/II-TR-2026-

Orta Doğu Teknik Üniversitesi
Enformatik Enstitüsü

Gömülü Bir Haberleşme Projesi için Kalite Maliyeti Analizi

Danışman Adı: Dr. Öğr. Üyesi Özden ÖZCAN TOP
(ODTÜ)

Öğrenci Adı: Engin Kılınç
(Bilişim Sistemleri)

Ocak 2026

TEKNİK RAPOR

ODTÜ/II-TR-2026-

REPORT DOCUMENTATION PAGE

1. AGENCY USE ONLY (Internal Use)	2. REPORT DATE 16.01.2026

3. TITLE AND SUBTITLE

COST OF QUALITY ANALYSIS FOR AN EMBEDDED COMMUNICATION PROJECT

4. AUTHOR (S) Engin Kılınç	5. REPORT NUMBER (Internal Use) METU/II-TR-2026-

6. SPONSORING/ MONITORING AGENCY NAME(S) AND SIGNATURE(S)
Information Systems Master's Programme, Department of Information Systems, Informatics Institute, METU
Advisor: Assist. Prof. Dr. Özden ÖZCAN TOP Signature:

7. SUPPLEMENTARY NOTES

8. ABSTRACT
This report examines the use of Cost of Quality (CoQ) analysis in hardware–software integrated embedded systems, focusing on projects where detailed quantitative effort data is not available. The study is based on a single industrial case involving an FPGA-based real-time embedded communication system. A systematic literature review is conducted to understand existing CoQ approaches and to identify their limitations in embedded system contexts. Based on these findings, CoQ activities in the case project are analyzed by considering software, hardware, and system or management activities separately. Quality related effort is estimated using a combination of project artifacts, expert judgement, and an exploratory Large Language Model (LLM)-based estimation approach. The results show that useful CoQ-related insights can be obtained without precise time-tracking data. In the studied project, software effort is mainly influenced by integration related rework, hardware effort is concentrated on early preventive activities, and system-level effort is largely driven by coordination and integration tasks. Overall, the study shows that CoQ analysis can be used as a qualitative, decision support tool to understand quality-- related effort in embedded system development under realistic industrial constraints.

9. SUBJECT TERMS Cost of Quality, Embedded Systems, Hardware–Software Integration, Case Study, Effort Estimation	10. NUMBER OF PAGES **90**

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Definition
CoQ	Cost of Quality
SLR	Systematic Literature Review
RTOS	Real-Time Operating System
UART	Universal Asynchronous Receiver-Transmitter
TCP/IP	Transmission Control Protocol / Internet Protocol
FPGA	Field-Programmable Gate Array
RQ	Research Question
IEEE	Institute of Electrical and Electronics Engineers
ACM	Association for Computing Machinery
LOC	Line of Code
KSLOC	Thousands of Source Lines of Code
SOI	System-of-Interest
LLM	Large Language Model
IbD	Interface-Based Design
PAF	Prevention–Appraisal–Failure

COCOMO	Constructive Cost Model
ROI	Return of Investment
PQMM	Process Quality Measurement Model
DMA	Direct Memory Access
BRAM	Block RAM
FIFO	First In First Out
I/O	Input/Output
PM	Person-Month
TL	Turkish Lira
IC	Inclusion Criteria
EC	Exclusion Criteria
SPL	Software Product Line
FSM	Functional Size Measurement
ASIC	Application-Specific Integrated Circuit
CE	Consumer Electronics
IT	Information Technology

1. INTRODUCTION

Software quality plays a key role in the success and sustainability of engineering projects. Software quality is an important aspect of engineering projects because it directly affects development time, rework, and system reliability. When quality issues occur, additional effort is usually required for activities such as repeated testing, fixing defects, or revising earlier work. Although these activities are common in most projects, their cost is rarely tracked in a clear and separate way.

The Cost of Quality (CoQ) concept provides a structured approach to better understand this situation. CoQ focuses on the effort spent on preventing defects, evaluating the system through testing and reviews, and correcting problems when they are discovered. However, in many projects, these quality related efforts are included within overall development costs, which makes it difficult to clearly identify how much time and effort are actually spent on quality.

The foundation of CoQ analysis in software engineering is largely based on the work of Crosby (1979), who defined quality as conformance to requirements and emphasized that the cost of fixing defects increases when they are detected at later stages of development. This idea later evolved into the Prevention-Appraisal-Failure (PAF) model, which classifies quality-related activities into prevention, appraisal, internal failure, and external failure costs.

Based on this model, several studies have examined how CoQ can be applied in practice. Slaughter et al. (1998) showed how quality-related effort can be categorized in large software projects, while Ahmad and Samat (2016) proposed an effort based approach to separate testing and quality assurance activities. Yuan and Gu (2006) introduced a cost-oriented testing model to support decision making in testing processes. In addition, Hollocker (1986) discussed the relationship between productivity and conformance costs, and Hong and Fong (2003) examined the use of Six Sigma to identify major cost drivers in consumer electronics projects. Other studies, such as those by Demirors et al. (2000) and Dikici et al. (2012), explored CoQ within the context of process improvement and maturity assessment.

Most studies in the literature apply these CoQ frameworks to conventional software development projects (Slaughter, Harter, & Krishnan, 1998). However, relatively few studies examine CoQ in embedded or hardware–software integrated systems. This gap motivates the present study because embedded systems differ significantly from conventional software in terms of quality management and cost distribution. In real-time embedded environments, evaluation (appraisal) costs tend to be considerably higher, reaching 21% of total development costs in industrial cases, due to the safety-critical nature of the software and the rigorous testing required for integrated subsystems (Laporte et al., 2012). In the consumer electronics sector, the cost of fixing requirement errors during testing can be 10 to 20 times higher than in early stages because software is now deeply embedded in physical hardware (Hong & Fong, 2003). In defense-related embedded systems, system complexity is often handled using specific design approaches such as Interface-Based Design. However, published CoQ data for these hardware-dependent systems is still limited (Eren, Özkan, & Demirörs, 2015).

Embedded projects face several challenges that make quality analysis more difficult. It is often hard to trace defects between hardware and software layers, since problems may appear in one part but be caused by the other. In addition, development progress depends on the availability of physical components, because software cannot be fully tested or executed without the required hardware.

Measuring non-functional quality aspects, such as timing or reliability, is also challenging. These issues are usually less visible and harder to detect, especially in real-time systems. Another difficulty is identifying quality related effort, as this effort is often mixed with general project work and not recorded separately. As a result, available data is usually incomplete or scattered.

In this study, the CoQ framework is applied to a real-time Field-Programmable Gate Array (FPGA) based embedded communication project. The aim is to examine how CoQ can be used in a mixed hardware–software environment and how it can support decision making when detailed quantitative data is not available.

Considering the motivation discussed above, this study focuses on the following Research Questions (RQ):

- **RQ1:** How can the activities in an embedded system development process be categorized under the CoQ framework?
- **RQ2:** What challenges arise when applying CoQ analysis to a real-world embedded hardware–software project?

- **RQ3:** How can a qualitative CoQ assessment support decisions when detailed quantitative data (e.g., man-hours, salaries) are not available?

The structure of this report is as follows. Section 2 reviews the literature on CoQ and its use in software and embedded system development, and highlights existing approaches and research gaps. Section 3 describes the research methodology and introduces Project X, including its scope, development domains, and organizational structure. Section 4 describes the project activities and presents the CoQ analysis by assigning these activities to CoQ categories in software, hardware, and system/management domains. Section 5 summarizes the key findings of the study and discusses its limitations and possible directions for future work.

2. BACKGROUND AND RELATED WORK

This section presents the background of the CoQ concept and details the Systematic Literature Review (SLR) conducted as part of this study, to identify research gaps in embedded systems.

2.1 How to Measure Cost of Quality

The CoQ framework (Slaughter et al., 1998) describes how effort is spent on activities such as defect prevention, defect detection, and failure correction. This perspective is closely related to Philip B. Crosby's definition of quality as "conformance to requirements" and his observation that the cost of fixing defects increases when they are found at later stages (Crosby, 1979; Slaughter et al., 1998).

Based on Crosby's ideas, the PAF model was introduced and is now widely used in quality cost analysis (Crosby, 1979). In this model, quality related activities are grouped into four main categories:

- **Prevention Costs:**
  Effort spent to avoid defects early, such as planning, documentation, requirement clarification, design reviews, or training.

- **Appraisal Costs:**
  Activities that check the system for defects, including inspections, testing, simulations, and performance validation.

- **Internal Failure Costs:**
  Issues detected before the product is delivered, such as debugging, redesign, hardware fixes, and repeating tests.

- **External Failure Costs:**
  Issues detected after delivery or during customer-facing activities, including demo failures, integration problems, or defects found by external teams.

While this model provides a robust structure for conventional software, applying it to embedded systems presents unique challenges. The close relationship between software and hardware makes it difficult to trace defects across the system. In addition, detailed accounting data is often not available, which limits the accuracy of quantitative cost calculations.

2.2 Systematic Literature Review

A **SLR** was carried out to examine how CoQ models have been used in hardware–software integrated environments. The review process was based on the guidelines proposed by Kitchenham and Charters (2007) and aimed to keep the search process systematic and repeatable.

2.2.1 Review Methodology

The review was conducted in three main phases:

- Planning the Review: Defining research questions to identify existing CoQ methods and their limitations in embedded domains.

- Conducting the Review: The literature search was carried out using primary databases, (IEEE (Institute of Electrical and Electronics Engineers) Xplore, Google Scholar) by using search terms such as *("Cost of Quality" AND Embedded)* and *("Cost of Quality" AND FPGA)*. A total of 251 papers were retrieved, and 14 studies (P1–P14) were selected for full-text analysis.

- Reporting the Review: Summarizing the findings to highlight the limitations of current approaches.

2.2.2 Planning the Review

This review was motivated by an initial examination of existing CoQ studies. The review showed that while CoQ is widely discussed in the context of conventional software development, its use in hardware-dependent and real-time embedded systems is limited.

Based on the guidelines proposed by Kitchenham and Charters (2007), a research protocol was defined at this stage to determine the scope of the review and to support a consistent search process. To structure the review, the following research questions were defined:

SLR Research Questions

- **RQ1:** What methods have been proposed in the literature to measure the CoQ in software or embedded system projects?

- **RQ2:** What are the main challenges and limitations reported for applying these CoQ measurement methods?

- **RQ3:** What benefits or advantages do these CoQ measurement methods provide according to existing studies?

These questions were used to guide the search strategy, inclusion and exclusion criteria, and the data extraction steps described in the following sections.

2.2.3 Conducting the Review

This stage of the systematic review focused on finding relevant studies and gradually narrowing the results based on predefined criteria. A search strategy was defined in advance, and selection rules were applied to decide which studies should be included in the review. These steps were followed to keep the review process clear and consistent.

Search Strategy and Selection

To address the research questions, the literature search was mainly carried out using IEEE Xplore, since it contains a large number of publications related to software engineering and embedded systems. In order to broaden the search and include studies published by other sources, Google Scholar was also used.

A set of Boolean search strings was developed by combining keywords related to the CoQ with domain-specific terms. The following search strings were applied:

- (“Cost of Quality” AND Software)
- (“Cost of Quality” AND Embedded)
- (“Cost of Quality” AND Hardware)
- (“Cost of Quality” AND “Open Source”)
- (“Cost of Quality” AND FPGA)
- (“Cost of Quality” AND Verification)
- (“Cost of Quality” AND Validation)
- (“Onur Demirörs” AND (“quality” OR “quality cost” OR “process improvement” OR “cost”))

- ("Cost of Quality" AND "real-Time systems")
- ("Cost of Quality" AND (embedded software))

Inclusion and Exclusion Criteria

To ensure that only high-quality and relevant studies were included in the final analysis, a set of formal criteria was defined. These criteria focus on approaches used to measure quality-related effort in software, hardware, and system-level domains. Particular attention was given to studies conducted in environments where direct quantitative data, such as man-hours, is limited or not available.

Inclusion Criteria (IC):

- **IC1:** Studies that propose, apply, or adapt a CoQ model (for example, the PAF model) in software development, hardware design, or integrated system environments.
- **IC2:** Research focusing on embedded systems, real-time systems, FPGA-based designs, software projects or hardware–software co-design projects.
- **IC3:** Studies that present methods for effort estimation, functional size measurement , or process quality evaluation, which can be used as indirect indicators when direct time-based data is missing.
- **IC4:** Peer-reviewed journal articles, conference papers, or detailed technical case studies that provide practical insights into the challenges, limitations, or benefits of tracking quality-related activities.

Exclusion Criteria (EC):

- **EC1:** Studies that refer to "Quality" or "Cost" only in a general manner without offering a structured framework, specific metrics, or calculation models.
- **EC2:** Sources with limited technical or academic depth, such as short summaries, presentation slides, or non-peer-reviewed blog posts.

Screening and Selection Results

The study selection followed a multi-stage screening process to reduce the initial search results to a relevant set of primary studies

Initial Search and Automated Filtering: The initial search, using the predefined search strings across IEEE Xplore and Google Scholar, returned a total of 251 papers..

Title and Abstract Screening: In this stage, the titles and abstracts of the identified papers were reviewed using the inclusion and exclusion criteria defined in Section

2.2.3.2. Papers that did not propose a specific CoQ model or were not related to software–hardware integrated systems were removed. After this step, 38 papers remained for further review.

Full-Text Evaluation and Quality Assessment: The remaining 38 papers were then examined in full coverage. Their methodologies and relevance to the research questions were evaluated to decide whether they should be included in the study. As a result of this process, 14 primary studies (P1–P14) were selected for the final analysis.

The selected 14 primary studies were published between 1986 and 2022. They include both early CoQ models and more recent studies that focus on embedded or hardware-dependent systems. While the literature reveals an extensive body of research on CoQ for conventional software projects, the limited number of studies identified in this SLR reflects a significant research gap regarding its application in hardware–software integrated systems. This deficiency confirms the necessity of the qualitative case study presented in the subsequent sec

2.2.4 Results of the SLR

This section synthesizes the findings from the 14 papers of this report.

The complete list of these 14 primary studies, including their bibliographic details and assigned codes (P1–P14), is provided in Appendix A.

Primary studies (P1–P14) and is structured to directly address the three research questions defined in the SLR protocol.

Existing CoQ Measurement Methods (Addressing RQ1)

The literature indicates that the majority of CoQ methodologies in software and embedded engineering are built upon the PAF model. These methods are categorized as follows:

- **Financial and Effort-Based Models**: Early foundational research by Slaughter et al. (1998) and Hollocker (1986) established that quality costs should be measured in financial terms (e.g., Return of Investment (ROI) or cost per line of code) to justify quality investments. Ahmad and Samat (2016) expanded this by proposing an effort-based approach to separate testing and quality assurance costs, particularly for government-outsourced projects.
- **Process-Oriented Quality Models:** Guceglioglu and Demirors (2011) introduced the Process Quality Measurement Model (PQMM), which shifts the focus from product

defects to measuring the quality of the process definition itself (analyzability, reliability, etc.) before the process is executed. Dikici et al. (2012) later showed that static process measurements can be useful for identifying problematic parts of a process at an early stage.

- **Optimization and Modeling Approaches:** Yuan and Gu (2006), as well as Demiray and Arslan (2022), presented models that consider the trade-off between quality control costs and potential failure costs in order to estimate an appropriate quality level. In addition, Hong and Fong (2003) applied Six Sigma techniques to analyze cost drivers in consumer electronics development.

Challenges and Limitations in Integrated Systems (Addressing RQ2)

Although the CoQ concept is well established in conventional software development, several challenges are reported when it is applied to hardware–software integrated systems. The literature highlights a number of practical issues that make CoQ analysis more difficult in such environments:

- **Hardware Dependencies and Complexity:** Studies by Laporte et al. (2012) and Jiang et al. (2006) point out that hardware and software are often tightly connected in embedded systems. Together with real-time timing constraints, this makes it difficult to separate and measure individual quality-related activities.
- **The Traceability Problem:** In embedded systems, determining the source of a defect is often not straightforward. It may be unclear whether a failure originates from hardware, software, or their interaction, which complicates defect classification..
- **Software-Centric Bias:** Most CoQ frameworks were originally developed for software projects. As a result, they usually do not include activities such as FPGA development, hardware bring-up, or system-level integration, even though these activities are critical in domains such as defense industry and communication systems.
- **Data Scarcity:** Many quantitative models (such as Constructive Cost Model (COCOMO)-based effort extraction) rely on granular man-hour logs or defect databases, which are often incomplete or entirely missing in real-world embedded projects

Benefits and Advantages of CoQ Measurement (Addressing RQ3)

Despite the implementation challenges, primary studies highlight several strategic advantages:

- **Rework Reduction and Predictability:** Systematic tracking of CoQ reveals inefficient activities and reduces avoidable rework, leading to higher process predictability

- **Visibility of Hidden Costs:** Ramasubbu and Kemerer (2019) demonstrated that CoQ frameworks reveal "hidden" costs that traditional accounting systems fail to capture, such as the long-term "interest" paid on technical debt
- **Improved Decision-Making and Maturity**: Integrating CoQ with Technical Debt Management allows teams to make informed design trade-offs "in the moment," rather than after a product release
- **Process Improvement Catalyst:** Tracking prevention, appraisal, and failure costs provides the objective rationale needed to identify the "weakest links" in a process and prioritize improvement initiatives

Conclusion of the SLR: The Research Gap

The results of the systematic literature review show that, although CoQ models are well defined for conventional software projects, their application to hardware–software integrated systems is limited. In particular, there is little work that addresses environments where detailed quantitative effort data is not available.

This gap in the literature provides the motivation for the qualitative case study presented in this report.

3. RESEARCH METHODOLOGY

This study uses a qualitative research approach and is based on a single case study. A case study is an empirical research method that focuses on understanding a real world phenomenon in its actual context, especially when the boundaries between the phenomenon and its environment are not clearly separated (Yin, 2018). Case studies are widely used in software engineering research because they allow development processes to be examined in realistic settings, rather than under controlled or artificial conditions (Runeson & Höst, 2009). This makes it possible to observe how activities and decisions take place during normal project work.

A single-case design was selected because the aim of this study is to understand how CoQ appears in a real embedded system project. Many CoQ related activities, such as documentation updates, testing, and debugging, become visible during everyday development tasks. For this reason, focusing on one project in detail allows these activities to be observed as they naturally occur (Runeson & Höst, 2009; Yin, 2018).

By examining Project X, the study analyzes how software and hardware components interact and how quality related costs emerge as both evolve in parallel.

3.1 Case Study Design

This study is based on a single-case design and focuses on the development process of Project X. The design follows the general case study principles described by Yin (2018) and makes it possible to observe how requirements are handled, how software and hardware change over time, and how integration activities lead to different types of CoQ-related effort during the project. The embedded system context is particularly important, since system behavior and its environment are often closely connected.

In this study, a distinction is made between the case itself and the unit of analysis, following Yin (2018). Project X is treated as the case, including its interaction with Team B and the external sensor module. However, the main unit of analysis for CoQ estimation is limited to the activities and effort of Team A.

This decision allows the analysis to be based on first-hand expert judgment, as well as available man-hour and Line of Code (LOC) data. The work carried out by Team B is therefore considered part of the project environment rather than a direct unit of analysis.

3.1.1 Case Selection

Project X was selected as a representative case (Yin, 2018) because it shows the main characteristics addressed in the research questions:

- **Mixed hardware–software environment:** It includes both software (Real-Time Operating System (RTOS), drivers) and hardware (FPGA) development.

- **Real-time constraints:** Real-time communication must be maintained through Universal Asynchronous Receiver / Transmitter (UART) and Transmission Control Protocol/Internet Protocol (TCP/IP).

- **Cross-team dependencies:** The project involves cross-team dependencies and coordination challenges.

- **CoQ Related Activities:** Many internal processes, such as documentation, testing, debugging, and integration, directly correspond to the life-cycle processes defined in ISO/IEC/IEEE 12207 and contribute to different CoQ components.

- **Limited quantitative data:** Detailed quantitative data (e.g., man-hours) are limited, making it a suitable environment to explore qualitative CoQ assessment.

For these reasons, Project X provides a suitable context for examining how CoQ is reflected in a mixed hardware–software environment, where issues such as defect traceability and effort measurement are more challenging than in conventional software projects.

3.1.2 Case Description

Project X involves the development of a real-time embedded communication bridge used for data exchange between multiple boards. The system is designed to support reliable communication with low latency.

The system consists of three main components:

1. Communication Bridge Board (developed by Team A),

2. Main Board (developed by Team B),

3. Sensor reader card (external module).

The Communication Bridge is responsible for packet handling, routing, and data format conversion over UART and TCP/IP interfaces. The Main Board receives the incoming data, performs the required processing and algorithms, and sends the results back to the bridge when needed.

The system is designed to allow the different components to work together smoothly. It also helps to reduce communication and protocol related issues and supports a communication structure that can be extended if required.

3.1.3 Project Purpose

Project X focuses on developing a communication bridge that supports stable and real-time data exchange between multiple hardware modules. Besides the technical implementation, the project also addresses common integration issues that arise from hardware–software interaction in embedded systems.

The purpose of the project can be summarized as:

- Maintaining a clear and continuously updated interface specification,
- Minimizing protocol mismatches and integration problems between teams,
- Establishing a reliable end to end communication workflow,
- Supporting future extensions of the system architecture.

3.1.4 Project Scope

The project is carried out by two engineering teams: Team A, which is responsible for the Communication Bridge, and Team B, which focuses on the Main Board.

Both teams work on software and hardware development for their own boards. System-level coordination tasks, including interface definition, integration planning, and overall communication management, are mainly handled by Team A. The project activities follow the system and software life-cycle processes defined in ISO/IEC/IEEE 12207:2017 (ISO/IEC/IEEE, 2017).

Scope of Team A (Communication Bridge Team)

Team A is responsible for designing and implementing the communication bridge board, including both its embedded software and hardware (FPGA) components. In addition, Team A serves as the system/management authority for the entire multi-board setup.

Software Scope

- Designing the RTOS-based software architecture and determining task/thread responsibilities.
  (Architecture Definition Process, 6.4.4)

- Implementing UART/TCP-IP drivers, packetization logic, and message-routing mechanisms.
  (Implementation Process, 6.4.7)

- Developing simulation tools to enable software testing before hardware availability.
  (Architecture Definition Process, 6.4.4)

- Performing software verification using unit tests, simulation-based tests, and board-level tests.
  (Verification Process, 6.4.9)

Hardware (FPGA) Scope

- Performing initial pin assignment, timing constraint definition, and memory/bandwidth planning for the communication bridge FPGA.
  (Architecture Definition Process, 6.4.4; Stakeholder Needs and Requirements Definition, 6.4.2)

- Implementing interface logic, buffering mechanisms, and timing infrastructure needed for reliable communication.
  (Implementation Process, 6.4.7)

- Conducting FPGA bring-up, timing validation, and hardware–software integration debugging.
  (Verification Process, 6.4.9; Integration Process, 6.4.8)

System-Level & Integration Scope

Team A also serves as the system integration owner for Project X, responsible for:

- Maintaining and updating the system-wide interface specification used by both teams.
  (Systems/Software Requirements Definition, 6.4.3)

- Leading requirement clarification meetings to resolve ambiguities across teams.
  (Stakeholder Needs and Requirements Definition Process, 6.4.2)

- Planning and executing integration sessions, defining test scenarios, expected message flows, logging strategies, and validation criteria.
  (Project Planning Process, 6.3.1; Integration Process, 6.4.8)

- Defining the end-to-end system operational flow that describes how sensor data, bridge logic, and main-board algorithms interact.
  (Architecture Definition Process, 6.4.4)

- Performing cross-team appraisal activities such as timing analysis, communication performance evaluation, and end-to-end validation.
  (Verification Process, 6.4.9; Validation Process, 6.4.10)

Team A, therefore, contributes not only to its own board but also to the coordination and quality assurance of the entire system.

Scope of Team B (Main Board Team)

Team B is responsible for the development of the Main Board, which includes both embedded software and FPGA/hardware updates when needed. Team B participates in system-level integration under the coordination of Team A.

Software Scope

- Providing detailed software requirements related to message structures, timing expectations, and protocol behavior.
  (Systems/Software Requirements Definition Process, 6.4.3)

- Implementing command-processing logic and running embedded algorithms that operate on incoming data from the communication bridge.
  (Implementation Process, 6.4.7)

- Conducting verification on the main-board side using simulations and on-board debugging tools.
  (Verification Process, 6.4.9)

Hardware Scope

- Executing FPGA or hardware-related updates when interface modifications, timing adjustments, or new routing/buffering requirements emerge during integration.
  (Implementation Process, 6.4.7)

- Supporting bring-up tests and low-level debugging activities in collaboration with Team A.
  (Integration Process, 6.4.8)

System/Management Scope

- Participating in joint integration sessions led by Team A, validating system behavior, and analyzing communication logs or timing deviations.
  (Integration Process, 6.4.8; Verification Process, 6.4.9)

- Updating Main Board software or hardware configurations to match evolving interface specifications.
  (Implementation Process, 6.4.7)

3.1.5 Team Details

Project X is carried out by two closely collaborating engineering teams responsible for different parts of the embedded system. The teams work in parallel during development and converge during system integration and field testing.

Table 1.Project X Team Details

Team	Engineering Domain	Role/Seniority	Employee Count
Team A (Communication Bridge)	Software	Senior Software Engineer	2
		Junior Software Engineer	1
	Hardware / FPGA	Senior FPGA Engineer	2
	System & Management	Senior System Engineer	1
	Technical Support	Technician	1
Team B (Main Board)	Software	Senior Software Engineer	1
		Junior Software Engineer	1
	Hardware / FPGA	Senior FPGA Engineer	1
Total			10

Note: Team A is primarily responsible for system-level coordination, including maintaining shared interface definitions, aligning communication assumptions, and organizing joint integration sessions with Team B. The members of Team B are listed in Table 1 to illustrate the cross-team complexity of the System-of-Interest (SOI). However, since detailed man-hour data and expert input were only available for Team A, the quantitative CoQ estimations in Section 4 focus only on Team A's effort. The impact of Team B on quality (for example, integration-related issues) is therefore captured indirectly, through the additional rework or appraisal effort required from Team A.

3.1.6 Hourly Engineering Rates and Cost Assumptions

All CoQ calculations in this study are based on estimated labor effort and hourly engineering rates. Since Project X does not include task-level financial or time-tracking records, personnel cost is used as the primary cost component across software, hardware, and system/management domains.

The hourly rates applied in this study are derived from the organization's internal salary structure. Due to confidentiality constraints, the exact salary values are not reported. Instead, representative hourly rates are used to preserve the relative differences between engineering roles (junior, senior, technician), which is sufficient for comparative CoQ analysis.

Hourly rates were calculated by converting monthly compensation into hourly values. For this calculation, a working time of 160 hours per month was assumed, based on 20 working days with 8 hours per day. The same hourly rates were then applied throughout the analysis.

- **Junior Engineer:** 250 TL/hour
- **Senior Engineer:** 550 TL/hour
- **Technician:** 180 TL/hour

The hourly rates used in this study do not represent exact official salaries. Instead, they are simplified values intended to reflect realistic cost differences between roles. This makes it possible to compare effort distribution and cost structure without exposing sensitive or confidential information.

For each activity, total cost is calculated by multiplying the estimated effort (in hours) by the number of engineers involved and their respective hourly rates. When more than one engineer worked on the same activity, the effort was divided equally unless stated otherwise.

All cost figures reported in Sections 4.5 and 4.6 are computed using these assumptions, ensuring internal consistency across all domains and CoQ categories.

3.1.7 Project Timeline

Table 2.Project X Phases

Phase	Period	Description
Phase 1 – Initial Development	June 2023 – June 2024	The main design and implementation work was carried out during this period. Key activities included creating the first interface documents, setting up the RTOS environment, developing communication drivers, and performing system-level integration tests with the Main Board and the sensor reader card.
Pause Period	June 2024 – June 2025	The project was temporarily paused due to external factors. No major development activities were performed in this period.
Phase 2 – Resumed Development	From June 2025 onwards	Development restarted in mid-2025. The teams are currently finalizing software and mechanical work, with plans to move into the field testing phase by January 2026.

3.1.8 Development Workflow

The development workflow of Project X follows an iterative and parallel structure, where software, hardware, and system integration tasks progress in coordination. The technical activities are aligned with the Technical Processes defined in ISO/IEC/IEEE 12207:2017, Section 6.4.

The process starts with the Stakeholder Needs and Requirements Definition (6.4.2) and the System/Software Requirements Definition (6.4.3). In these stages, customer expectations and interactions between the teams are combined into a common system-level interface specification. After this, Team A and Team B carry out the Architecture Definition (6.4.4) and Design Definition (6.4.5) processes to define the software and FPGA structures for their respective boards.

The next stage is Implementation (6.4.7), where embedded drivers, protocol parsers, and FPGA logic are developed. Each component is then tested through the Verification (6.4.9) process using unit tests and simulations. After verification, the components enter the Integration (6.4.8) process, where software and hardware modules are combined to enable end to end communication. Finally, the overall system is evaluated through the Validation (6.4.11) process during field testing, to confirm that it meets its intended goals in a real-world environment.

3.2 Case Study Conduct

The case study was carried out during Phase 2 of Project X, when development activities resumed and the CoQ analysis was performed. Data collection and activity identification were conducted over a period of approximately four weeks. During this time, quality-related activities were identified and the effort required for these activities was estimated using expert judgment and basic size-based measures, such as lines of code counts for COCOMO-based estimation.

Data was collected through a combination of project artifact review and direct discussions with engineers from Team A. These discussions were held individually with senior engineers from different domains, including two senior software engineers, one senior FPGA or hardware engineer, and one senior system engineer responsible for system-level and integration activities. The discussions were informal and conducted in person, with the aim of understanding which activities were performed, how they related to quality or rework, and how much effort they typically required based on experience.

No audio or video recordings were made, and no formal interview transcripts were produced. Instead, information was documented through notes taken by the author during the meetings. These notes were later reviewed together with available project artifacts, such as interface specifications, FPGA design files, simulation configurations, integration records, and source code repositories. This comparison helped confirm that the identified activities reflected the actual project outputs.

After data collection, all identified activities were assigned to CoQ categories, including Prevention, Appraisal, Internal Failure, External Failure, and Value-Added. This classification was based on the role and purpose of each activity within the development and integration process, rather than on precise time measurements. To improve reliability, the activity list and its categorization were reviewed with senior members of Team A. This step served as a form of member checking, where the engineers verbally confirmed that the activities and their classifications were consistent with the work carried out in the project. No written approval was obtained, but verbal confirmation was provided.

Although Project X involves collaboration between multiple teams, direct access to the internal development processes of Team B was not available. As a result, no interviews or expert judgments were conducted with Team B members. Activities related to Team B are therefore described only at a high level, based on observed integration behavior and interface interactions, and are included mainly to provide system context. The CoQ calculations are intentionally limited to the activities of Team A, ensuring that all estimated values are based on accessible and verifiable information.

Overall, this approach can be described as an industrial observational case study, where data is collected through direct involvement in the project, inspection of project artifacts, and interactions with experts, rather than through formal qualitative interviews. Although detailed time-tracking records were not available, this structured and transparent process still provides a reasonable and credible basis for analyzing how quality related effort is distributed within the embedded system project.

3.3 Limitations and Future Work

The main limitation of this study is the absence of detailed quantitative effort data, such as exact engineering hours or direct financial values assigned to individual activities. In embedded system projects, work is usually shared across software, hardware, and system integration tasks, and detailed time tracking is not common in industrial practice. For this reason, the CoQ estimation in this study is based on available project artifacts and structured expert judgment instead of precise numerical records.

Although Project X involves two collaborating teams, the analysis focuses only on the activities of Team A. This choice is considered reasonable, since Team A is responsible for maintaining shared interface definitions and coordinating integration activities. As a result, many quality-related issues that originate from other parts of the system become visible through additional coordination effort, repeated integration work, or rework carried out by Team A. Even though the internal development effort of Team B is not directly included, the overall system-level quality impact is therefore reflected through Team A's role in integration and coordination.

Another limitation is related to the nature of FPGA and embedded system development. Many hardware–software interaction issues become visible only during late stage integration or field testing. This makes it difficult to clearly separate prevention, appraisal, and failure-related activities. In addition, several system-level tasks, such as coordination between teams, interface discussions, and repeated integration attempts, do not always result in clear technical artifacts, which further limits the accuracy of cost estimation.

Future work could address these limitations by using a more structured approach to effort tracking. For instance, recording integration sessions, documenting interface changes, and keeping simple time logs for FPGA bring-up activities could provide more detailed data. This would allow a more detailed CoQ analysis based on numerical information and could help clarify how early prevention activities affect later rework and corrective effort.

4. COST OF QUALITY ANALYSIS

This section analyzes the activities carried out in Project X using the classical CoQ categories: Prevention, Appraisal, Internal Failure, External Failure, and Value-Added.

Since the project involves software, hardware, and system-level activities, the CoQ assessment is performed separately for these three domains.

For each domain, activities were identified, mapped to the relevant ISO/IEC/IEEE 12207:2017 life-cycle processes, and classified under the appropriate CoQ category.

These classifications will later serve as the basis for quantitative CoQ estimation.

4.1 Software-Related CoQ Activities

This subsection focuses on software-related CoQ activities. The analysis follows the Prevention–Appraisal–Failure (PAF) structure, and each CoQ category is discussed separately in the subsections that follow.

4.1.1 Prevention Costs (Software)

Prevention costs refer to the effort spent to avoid defects before they appear in the development process. In Project X, these activities are mainly related to the System/Software Requirements Definition (6.4.3) and Architecture Definition (6.4.4) stages. This phase is considered part of prevention because, in embedded systems, having a clear architecture and well defined interfaces early on is criticali. Spending effort at this stage helps reduce the likelihood of problems such as protocol inconsistencies or timing-related issues, which would otherwise lead to additional rework in later phases of the project.

In Project X, software related prevention activities included:

- **Maintaining and updating the software-side interface specification**, including packet formats, timing diagrams, and protocol rules
  (ISO/IEC/IEEE 12207:2017 — Requirements Definition, 6.4.3)

- **Planning the RTOS-based architecture and communication driver structure,** defining thread responsibilities, priorities, and interactions
  (ISO/IEC/IEEE 12207:2017 — Architecture Definition, 6.4.4)

- **Performing early modular decomposition and code-structure planning**, ensuring clean separation between driver logic, protocol encoding/decoding, and application components
  (ISO/IEC/IEEE 12207:2017 — Design Definition, 6.4.5)

- **Designing the architecture of the software simulation environment**, enabling functional verification before real hardware became available
  (ISO/IEC/IEEE 12207:2017 — Architecture Definition, 6.4.4)

These preventive activities improved clarity, reduced ambiguity, and lowered the risk of integration-related rework.

4.1.2 Appraisal Costs (Software)

Appraisal costs cover the effort related to identifying defects and checking whether the software satisfies its requirements before it is combined with hardware. In Project X, software related appraisal activities are mainly linked to the Verification (6.4.9) and Integration (6.4.8) processes.

This stage includes activities such as unit testing and software-only simulations. Such tests are used to evaluate software behavior in isolation and to confirm that the implemented logic functions as intended. For a real-time communication system, this allows potential issues to be identified in a controlled setting before the software is executed on physical hardware.

Software appraisal activities in Project X included:

- **Unit testing of UART/TCP/IP drivers and protocol parsers**
  (ISO/IEC/IEEE 12207:2017 — Verification Process, 6.4.9)

- **Module-level functional testing** to confirm correct packet formation, parsing logic, and error handling
  (ISO/IEC/IEEE 12207:2017 — Verification Process, 6.4.9)

- **Integration testing using the Main Board software simulation**, validating that the Communication Bridge software behaved correctly even when the real Main Board

was not available
(ISO/IEC/IEEE 12207:2017 — Integration Process, 6.4.8)

- **Executing software-only simulation scenarios** to verify message flow correctness and timing assumptions before hardware integration
(ISO/IEC/IEEE 12207:2017 — Verification Process 6.4.9)

(System-level timing/throughput checks were intentionally excluded from this section and classified under System/Management activities where they belong.*)*

4.1.3 Internal Failure Costs (Software)

Internal failure costs represent the rework caused by defects that are discovered before the system is released to the field. These usually occur during the Implementation (6.4.7) and early Integration (6.4.8) stages. This category includes tasks that were not completed correctly the first time, such as debugging protocol errors or fixing timing issues found during system bring-up. These costs mainly reflect the extra effort needed to make the software compatible with the hardware constraints.

Software internal failure activities included:

- **Debugging issues identified during software–simulation or software–hardware integration sessions**
(ISO/IEC/IEEE 12207:2017 — Implementation Process, 6.4.7)

- **Fixing protocol-related software defects**, including packetization, formatting, and parsing errors
(ISO/IEC/IEEE 12207:2017 — Implementation Process, 6.4.7)

- **Correcting discrepancies between simulation behavior and real hardware timing**, including buffer tuning and timing adjustments
(ISO/IEC/IEEE 12207:2017 — Implementation Process, 6.4.7)

4.1.4 External Failure Costs (Software)

External failure costs occur when defects are not detected during laboratory testing and only appear under real-world conditions. These are related to the Validation (6.4.11) or Transition (6.4.10) stages. The reason for this classification is that some embedded system problems, such as crashes or unexpected behavior, only show up under specific environmental conditions or real sensor inputs that cannot be fully simulated in a lab. These failures are especially critical because they require fixes at the final stages of the project.

External software failures in Project X included:

- **A one-time system crash** requiring manual restart to restore system function

- **Algorithm output deviations** under real sensor timing, prompting refinements to software logic

- **Occasional packet corruption or loss** during field tests, not reproducible in simulation or laboratory environments

4.1.5 Value Added Activities (Software)

Value-added activities are not part of the CoQ but represent the productive work required to build the main functionality of the system. In Project X, these activities are linked to the Implementation (6.4.7) stage, where core components such as drivers and packet encoders are developed. This category is included to clearly separate the effort spent on building the product itself from the effort spent on ensuring its quality. This distinction clarifies the impact of quality related activities on overall development effort.

Value-added software activities included:

- **Implementing UART and TCP/IP drivers**
  (ISO/IEC/IEEE 12207:2017 — Implementation Process, 6.4.7)

- **Developing the packet encoder/decoder**, enabling protocol framing, parsing, and structured data handling
  (ISO/IEC/IEEE 12207:2017 — Implementation Process, 6.4.7)

- **Executing end-to-end message workflows** within the software domain (e.g., processing commands, constructing outbound packets)
  (ISO/IEC/IEEE 12207:2017 — Integration Process, 6.4.8)

- **Implementing the software simulation tool**, enabling testing with synthetic sensor data and multi-board emulation

(ISO/IEC/IEEE 12207:2017 — Implementation Process, 6.4.7)

4.2 Hardware-Related CoQ Activities

This subsection introduces the hardware-related CoQ activities. The analysis is then organized according to the PAF categories, which are discussed individually in the subsequent subsections.

4.2.1 Prevention Costs (Hardware)

Hardware prevention costs involve early efforts to ensure that the FPGA and physical components are designed correctly from the start. In Project X, these activities are linked to Architecture Definition (6.4.4). This stage is categorized as prevention because decisions like initial pin-assignment and timing planning are made to avoid fundamental hardware errors. In embedded systems, late changes to hardware design are extremely expensive, so investing in these early design tasks acts as a critical defense against future failures.

- **Initial pin-assignment and signal-mapping decisions for the Communication Bridge FPGA**
  (ISO/IEC/IEEE 12207:2017 — Architecture Definition, 6.4.4)
- **Preparation of essential timing and Input/Output (I/O) constraints** including clock-domain definitions and basic timing requirements, is included within the Clock & Reset Architecture activity. *(ISO/IEC/IEEE 12207:2017 — Architecture Definition, 6.4.4)*
- **Defining memory and bandwidth requirements for FPGA-side data transfer** (determining optimal Block Ram (BRAM) / First in First Out (FIFO) sizes and bandwidth needs to ensure uninterrupted packet buffering and transfer in the communication bridge)
  (ISO/IEC/IEEE 12207:2017 — Stakeholder Needs & Requirements Definition, 6.4.2)

These preventive activities reduced the likelihood of fundamental design errors and ensured the FPGA was electrically and functionally ready for integration with embedded software.

4.2.2 Appraisal Costs (Hardware)

Appraisal costs in the hardware domain include the activities used to detect defects and check whether the FPGA logic works correctly before full system integration. These activities are linked to the Verification Process (6.4.9). This stage is included here because bring-up tests and signal behavior checks help ensure that the hardware is stable and reliable. These tests are necessary to confirm that the hardware is working as intended before it is combined with the embedded software in later stages.

- **Initial bring-up testing of the FPGA prototype**, validating basic operation after flashing the first bitstream (power-up behavior, UART interface availability, peripheral readiness)
  (ISO/IEC/IEEE 12207:2017 — Verification, 6.4.9)
- **Basic timing and signal-behavior checks**, ensuring the configured clocks, resets, and data paths behaved as expected under Vivado timing analysis
  (ISO/IEC/IEEE 12207:2017 — Verification Process, 6.4.9)

These appraisal activities ensured that the FPGA logic was functional and stable before more complex multi-board interactions were attempted.

4.2.3 Internal Failure Costs (Hardware)

Internal failure costs represent the rework needed to fix hardware and FPGA issues discovered in the lab before field testing. These activities are associated with the Implementation (6.4.7) and Verification (6.4.9) processes. This stage is linked to internal failure because it captures the effort spent on fixing problems that were not "done right the first time," such as correcting data-alignment issues or re-tuning timing parameters. These costs show the time lost during the stabilization of the hardware design.

Internal hardware failures observed in Project X included:

- **Correcting a Direct Memory Access (DMA) data-alignment issue where incoming samples appeared shifted**, requiring updates to the FPGA interface logic and regeneration of the bitstream
  (ISO/IEC/IEEE 12207:2017 — Implementation Process, 6.4.7)
- **Adjusting FPGA–software timing parameters after inconsistencies were observed**, Real system timing exposed behaviors that differed from design assumptions, requiring updates to buffering or handshake logic.
  (ISO/IEC/IEEE 12207:2017 — Implementation Process, 6.4.7)
- **Investigating temperature-dependent instability during internal tests**, Higher temperature conditions revealed intermittent communication degradation, requiring analysis of timing margins and I/O stability.
  (ISO/IEC/IEEE 12207:2017 — Verification Process, 6.4.9)

These issues were resolved during integration and therefore contributed to internal failure cost rather than external failure.

4.2.4 External Failure Costs (Hardware)

External failure costs arise from hardware problems that only appear when the system is operating in real-world field conditions. These are associated with the Validation (6.4.11) or field-testing phase. The justification for this category is that certain issues, like communication degradation due to temperature or mechanical factors, are often hidden in the lab and only emerge under real operational loads. These failures are the most disruptive because they require on-site diagnosis and component replacement.

In Project X, hardware-related external failures included:

- **Failure of an onboard storage module**, requiring on-site replacement (This interruption prevented normal test progression and increased field effort.)
- **Communication degradation caused by environmental conditions**, such as elevated temperature affecting signal integrity (This behavior was not observed during laboratory tests.)

These failures emerged under real operational load and thus fall under the External Failure category.

4.2.5 Value-Added Activities (Hardware)

Value-added activities refer to the main development work that directly creates the hardware functionality of the bridge. These activities are linked to the Implementation Process (6.4.7). For example, configuring the communication interfaces and implementing the buffering logic are core tasks that actually build the product. These activities are separated from CoQ to clearly distinguish between the effort spent on *building* the system and the effort spent on *ensuring its quality*.

Value-added hardware activities included:

- **Configuring FPGA-side communication interfaces** including UART IP and clocking resources required for the bridge
  (ISO/IEC/IEEE 12207:2017 — Implementation Process, 6.4.7)
- **Providing stable timing, routing, and buffering behavior** to support reliable data exchange between embedded boards
  (ISO/IEC/IEEE 12207:2017 — Implementation Process, 6.4.7)

- **Supporting low-level integration with embedded software**, ensuring that FPGA-generated data structures were compatible with software parsers and protocols
 (ISO/IEC/IEEE 12207:2017 — Integration Process, 6.4.8)

These activities form the hardware backbone of the communication bridge and directly contribute to system functionality.

4.3 System/Management CoQ Activities

This subsection outlines the system and management–level CoQ activities. The detailed discussion follows the PAF structure, with each category addressed in a separate subsection.

4.3.1 Prevention Costs (System/Management)

System-level prevention costs include the coordination work done to ensure both teams are aligned before they start integrating their components. These activities are linked to Requirements Definition (6.4.3), Stakeholder Needs (6.4.2), and Architecture Definition (6.4.4). This stage is vital for prevention because maintaining a shared interface specification and holding clarification meetings stops "cross-team" mismatches before they happen. These efforts aim to reduce the risk of expensive re-coordination later in the project.

System-level prevention activities in Project X included:

- **Maintaining and updating the system-wide interface specification**, ensuring that both the Communication Bridge Team (Team A) and the Main Board Team (Team B) used the same message formats, timing rules, and communication assumptions
 (ISO/IEC/IEEE 12207 — Systems/Software Requirements Definition, 6.4.3)

- **Requirement clarification meetings** between teams to eliminate ambiguities in expected behavior, packet contents, and timing constraints
 (ISO/IEC/IEEE 12207 — Stakeholder Needs and Requirements Definition Process , 6.4.2)

- **Planning integration sessions,** including deciding which message flows would be tested, what logs would be collected, and how end-to-end behavior would be validated in each session
 (ISO/IEC/IEEE 12207 — Project Planning Process, 6.3.1)

- **Defining the operational system flow**, which describes how data is expected to move through the sensor module, the bridge, and the main board during real-time

operation
(ISO/IEC/IEEE 12207 — Architecture Definition Process, 6.4.4)

As a result of these preventive activities, integration issues were less frequent and system behavior was interpreted more consistently across teams.

4.3.2 Appraisal Costs (System/Management)

System-level appraisal activities focus on verifying the behavior of the entire integrated system where hardware and software interact. These efforts are mapped to the Integration (6.4.8), Verification (6.4.9), and Validation (6.4.10) processes. This stage is justified because joint integration sessions and end-to-end performance analysis are used to check if the complete system meets its standards. These activities serve as a final check to confirm that all subsystems work together correctly in real-time.

System-level appraisal activities in Project X included:

- **Joint integration testing sessions** where Team A and Team B executed predefined scenarios to verify message exchange, protocol compliance, and timing behavior across all boards
  (ISO/IEC/IEEE 12207 — Integration Process, 6.4.8)

- **End-to-end system validation**, ensuring that sensor-originated data traveled through the Communication Bridge and reached the Main Board algorithms as expected
  (ISO/IEC/IEEE 12207 — Validation Process, 6.4.10)

- **Cross-team log and performance analysis,** including inspection of UART/Ethernet timings, packet contents, and processing delays observed during integrated operation
  (ISO/IEC/IEEE 12207 — Verification Process, 6.4.9)

These appraisal activities enabled the teams to detect system-level defects early and prevented unexpected failures during field testing.

4.3.3 Internal Failure Costs (System/Management)

Internal failure costs at the system level refer to the extra effort caused by unsuccessful integration attempts, mainly due to mismatched assumptions between Team A and Team B. These activities are linked to the Requirements Definition (6.4.3) and Technical Management (6.3.1) processes. They are classified as internal failure because repeated integration sessions, additional meetings, and re-alignment of expectations are forms of rework. In other words, this effort arises when the system does not behave as expected based on the written specifications, and the problems are discovered

Internal failures observed in Project X included:

- **Repeated integration sessions caused by mismatched assumptions**, such as differences in expected packet structure or message ordering between the two teams
 (ISO/IEC/IEEE 12207 — Systems/Software Requirements Definition, 6.4.3)

- **Corrections to the system-wide interface specification**, required when integrated execution revealed inconsistencies between documented behavior and actual system behavior
 (ISO/IEC/IEEE 12207 — Systems/Software Requirements Definition , 6.4.3)

- **Ongoing re-coordination efforts when integration highlighted unexpected system responses** (e.g., message timing conflicts, packet alignment mismatches), requiring adjustment of shared understanding
 (ISO/IEC/IEEE 12207 — Technical Management, 6.3.1)

These failures resulted in additional system-level analysis, repeated test cycles, and increased coordination work before the product could operate consistently.

4.3.4 External Failure Costs (System/Management)

External system-level failures occur when the fully assembled system becomes unstable or behaves unexpectedly during field operation. These issues are classified under this category because they are only triggered by real-world multi-board dynamics that lab tests could not reproduce. Fixing these problems is critical as it requires additional diagnostic effort and manual restarts to restore system function during important field runs. These costs represent the highest level of system risk.

System-level external failures observed in Project X included:

- **Temporary full-system instability during field operation**, requiring manual restart or reinitialization to restore normal communication flow
  These failures interrupted field tests, causing repeated runs and additional effort to

restore normal behavior.

- **Unexpected behavior observed only under realistic multi-board execution**, such as packet timing deviations or message delays that did not occur under laboratory conditions
  These required additional diagnostic effort to understand how environmental or real-time dynamics influenced the system.

Because these failures emerged when the fully assembled system was exercised in real conditions, they fall into the External Failure category.

4.3.5 Value-Added Activities (System/Management)

Value-added system activities represent the productive work required to combine separate components into a single, functioning communication bridge. In Project X, these activities are mainly associated with Architecture Definition (6.4.4), Integration (6.4.8), and partly with early Verification (6.4.9). They are classified as value-added because they directly contribute to making the system operational rather than merely checking it or fixing errors. For example, setting up the multi-board configuration and defining the end to end data flow are essential steps that enable real-time communication. While verification is usually classified as an appraisal activity, the initial evaluation of communication performance is considered value-added in this context, since it establishes a basic performance reference for the system.

Value-added system activities in Project X included:

- Establishing and operating the integrated multi-board system setup, consisting of the sensor card (real and simulation-generated data), the Communication Bridge, and the Main Board, and using this setup as the primary environment for system-level development and evaluation
  (ISO/IEC/IEEE 12207 — Integration Process, 6.4.8)

- Defining and documenting the end-to-end operational system flow, including the system-level behavior diagram that describes how data moves from sensor inputs through the Bridge and into Main Board algorithms
  (ISO/IEC/IEEE 12207:2017 — Architecture Definition Process 6.4.4)

- Evaluating the real-time communication performance between the Communication Bridge and the Main Board by measuring end-to-end latency, packet integrity, transmission timing, and throughput under integrated operating conditions. This activity ensures that the communication path between system components meets performance expectations and supports reliable real-time operation.
(ISO/IEC/IEEE 12207 — Verification Process 6.4.9)

These activities provide essential system-level functionality rather than overhead or rework.

4.4 Cost of Quality Estimation Methodology

Because Project X does not include detailed time-tracking data (such as engineer hour logs or task-level effort records), the CoQ had to be estimated using a combination of measurable project artifacts and structured expert judgement. In our case, software costs rely on code-based estimation (using a simplified COCOMO model for activities that produce measurable source-code changes) together with expert judgement for non-code activities. Hardware costs are derived from a milestone-based decomposition of the FPGA block-design process, complemented again by expert assessment for bring-up, debugging, and other non-measurable tasks. System-level activities,such as cross-team coordination, integration sessions, end-to-end validation, and field-observed failures,do not generate quantifiable artifacts, and therefore all system/management CoQ values are estimated solely through structured expert judgement. Overall, the methodology ensures that each domain is evaluated using the most appropriate evidence available, while maintaining consistency across all CoQ categories.

In addition to artifact-based and expert-based estimation methods, this study also employs an Large Language Model (LLM)-based effort estimation approach as a complementary technique to cross-check the consistency of the estimated values.

4.4.1 Scope of Estimation

The estimation of the CoQ components considers only the conformance and non-conformance efforts of Team A. The internal activities of Team B are not directly included in the analysis. However, any rework or appraisal effort required from Team A due to external integration problems is recorded as part of Team A’s failure-related costs.

4.4.2 Software Cost Estimation Method

Software-related activities in Project X are estimated using three separate approaches. This separation reflects the different types of work involved in software development for embedded and real-time systems and matches the structure observed in the project.

Based on this, the engineering effort for software activities is estimated using the following methods:

(A) A size-based estimation approach (for all activities that generate or modify code), and
(B) A structured expert-judgement approach (for non-code activities such as testing, analysis, or planning).
(C) LLM-Based Effort Estimation, which is used as an additional reference point by mapping activity descriptions to generalized engineering knowledge and typical industrial patterns.

(A) Size-Based Estimation for Code-Related Activities

For tasks that involve the creation or modification of source code (e.g., driver implementation or protocol fixes), effort is calculated using the Basic COCOMO model, originally defined by Boehm (1981). This method utilizes Thousands of Source Lines of Code (KSLOC) as the primary input to provide a formulaic baseline for development effort.

The Basic COCOMO effort equation is:

$$\text{Effort} \approx a \times (\text{KSLOC})^\wedge b$$

where

- E = development effort in person-months,
- KSLOC = thousands of lines of code added or modified,
- a and b = coefficients determined by the project category.

Boehm categorizes software projects into Organic, Semi-Detached, and Embedded modes, each with different nominal parameters. Since Project X is a real-time, hardware-constrained embedded communication system, the appropriate category is Embedded Mode, whose coefficients are:

Table 3.COCOMO Coefficients for Embedded System Softwares

Mode	**A**	**b**
Embedded	3.6	1.20

Converting COCOMO Effort to Engineer-Hours

CoQ cost calculation requires effort in hours, whereas COCOMO expresses effort in person-months (PM).
Therefore, the COCOMO effort is converted to hours using the standard convention used in software cost estimation practice:

$$1 \text{ PM} = 160 \text{ hours}$$

This value corresponds to 20 working days × 8 hours per day and is consistent with COCOMO-derived models in industry and academia.

Thus:

$$\text{Effort hours} = E \times 160$$

This allows direct multiplication with the hourly engineering rate in CoQ cost calculations.

Software Activities Estimated Using Basic COCOMO

Only tasks that produce measurable code modifications fall under this estimation method. In Project X, the following CoQ activities generate source code or modify existing modules and are therefore computed using the COCOMO model:

Internal Failure (Software)

- Fixing message parsing, packetization, and formatting errors
- Fixing inconsistencies between the simulation setup and the real hardware behavior (timing mismatches, packet delays)

External Failure (Software)

- Algorithm outputs deviating from expected behavior when exposed to real sensor timing and field conditions, requiring further refinement

Value-Added (Software)

- Implementing UART and TCP/IP drivers
- Developing the packet encoder/decoder (protocol parsing and framing logic)
- Implementing the simulation tool that allows embedded boards to run in a virtual field environment

These tasks produce quantifiable artifacts (KSLOC added/modified), making them suitable for size-based estimation.

(B) Expert-Judgement Estimation for Non-Code Activities

Many software quality-related activities in Project X do not generate measurable code artifacts and therefore cannot be evaluated using size-based algorithmic models. These activities include requirements analysis, architectural planning, simulation-based validation, and various cross-team coordination tasks. For such tasks, effort estimation is performed using structured expert judgment, which remains one of the most reliable approaches for software work that lacks quantifiable size indicators, as also emphasized in studies comparing expert-based and model-based effort estimation (Jørgensen, Boehm, & Rifkin, 2009).

In this approach, engineers familiar with the project estimate the approximate effort required for each activity based on their prior experience with similar tasks. To keep the estimates consistent across activities, initial values are reviewed and adjusted when needed. When more than one engineer is involved, a single representative effort value is selected, typically based on a common agreement or the median estimate. This procedure allows non-code activities to be quantified in a practical and coherent manner despite the absence of detailed time-tracking logs.

The following activities from Section 4.1 fall into this category:

Prevention (Software)

- Maintaining and updating the interface specification
- Planning the RTOS-based architecture and communication driver structure
- Performing early modular decomposition and code-structure planning
- Designing the software simulation environment

Appraisal (Software)

- Unit testing of UART/TCP/IP drivers and message parsers
- Module-level functional testing
- Integration testing using the Main Board software simulation
- Executing software-only simulation scenarios

Internal Failure (Software)

- Debugging failed integration sessions (when no code changes occur)

- Updating or correcting the interface specification
- Investigating simulation–hardware inconsistencies when resolved through analysis rather than code modification

External Failure (Software)

- Investigating a one-time system crash observed in field testing
- Analyzing occasional packet loss or corruption during field runs

Value-Added (Software)

- Executing full end-to-end data exchange workflows between all boards

(C) LLM-Based Effort Estimation

As a third estimation approach, an LLM-based effort estimation method was applied to software-related CoQ activities.

In this method, a large language model is used to estimate the engineering effort required for each activity by interpreting detailed task descriptions and mapping them to generalized knowledge of industrial software and embedded system development practices.

The same activity set used for size-based and expert-judgement-based estimation was provided to the LLM using a structured prompt. The model was asked to produce effort estimates in engineering hours and to indicate the assumed engineer profile (senior or junior) for each activity.

The LLM-based method operates independently from COCOMO and expert judgement and does not rely on project-specific historical data or source-code size metrics. Instead, it represents a knowledge-driven estimation approach based on learned patterns from a large collection of engineering-related texts.

This approach offers a different way to estimate effort, especially for activities that cannot be easily measured using artifact-based methods.

4.4.3 Hardware Cost Estimation Method

In Project X, detailed time-tracking records are not available for FPGA development activities. For this reason, the CoQ for hardware related work is estimated using three complementary approaches: milestone-based artifact breakdown, structured expert judgment, and LLM-based industrial estimation. This combination is commonly used in embedded system CoQ studies when direct effort data cannot be obtained and engineering progress is instead reflected through development milestones and expert input (Laporte et al., 2012).

On the hardware side, the main reference milestone is the FPGA block-design stage. This stage includes tasks such as processing-system configuration, interconnect definition, clock and reset planning, memory bandwidth considerations, and I/O pin assignment. Based on discussions with the digital design engineer, this milestone required approximately one work-week (about 40 hours) during the early development phase.

While individual hardware design artifacts, such as constraint sets, block-diagram elements, and mapping documents, do not directly provide quantitative effort data, they help organize preventive hardware activities into clearly defined sub-tasks. This structure makes it possible to estimate the relative effort of these activities in a consistent manner.

Hardware CoQ costs are therefore derived using the following two mechanisms:

(A) Milestone-Based Effort Estimation for Prevention Activities

The 40-hour block-design effort is decomposed into the preventive hardware activities defined in Section 4.2.1. The distribution reflects the relative complexity and workload of each task, based on engineering judgement validated through interviews with the digital design engineer:

Table 4.Effort Share (in percentage) for Hardware Prevention Activities

Prevention Activity	**% Effort Share**	**Description**
Initial pin-assignment and signal mapping	30	Pin assignment, voltage standards, external I/O mapping
Preparation of timing and I/O constraints (clock configuration, reset/clock domain planning)	15	Clock-tree definition, reset distribution, basic timing constraints
Defining memory and bandwidth requirements (buffer sizing, throughput needs)	30	BRAM/FIFO sizing, bandwidth estimation, data-path buffering strategy
Processing-system peripheral configuration & interconnect setup	25	Interface configuration, peripheral mapping, address planning

Total preventive effort = 40 hours, yielding:

- Pin assignment & signal mapping → 0.30 × 40h = 12 hours

- Timing & I/O constraints → 0.15 × 40h = 6 hours

- Memory & bandwidth planning → 0.30 × 40h = 12 hours

- Processing-system & interconnect configuration → 0.25 × 40h = 10 hours

This milestone-based decomposition ties preventive hardware CoQ costs to a concrete design deliverable completed during the project, ensuring traceable and systematic effort estimation.

(B) Expert-Judgement Estimation for Non-Measurable Hardware Activities

Many hardware tasks do not generate measurable engineering artifacts or tool-based metrics that correlate with engineering effort. These include bring-up testing, signal-level verification, real-hardware timing observations, and debugging of intermittent failures. Because such tasks involve iterative investigation and system-level interactions, they are estimated using expert judgement informed by discussions with the digital design engineer and comparison with similar FPGA debugging cycles.

Hardware Activities Estimated Using Expert Judgement

Appraisal Activities (Hardware)

Performed before system-level integration:

- Initial bring-up testing of the FPGA prototype (power-up behavior, communication interface availability, peripheral readiness)

- Timing and signal-behavior checks using tool-based timing analysis and hardware probes

Internal Failure Activities (Hardware)

Rework performed to address defects discovered before field validation:

- Correcting the data-alignment issue requiring interface-logic updates and regeneration of the configuration bitstream

- Adjusting FPGA–software timing parameters when real hardware behavior diverged from design assumptions

- Investigating temperature-dependent instability observed during internal test cycles

External Failure Activities (Hardware)

Failures that became evident only under realistic field conditions:

- Replacing the malfunctioning storage module during field testing

- Investigating communication degradation caused by environmental or mechanical factors (e.g., temperature variations, connector sensitivity)

Value-Added Activities (Hardware)

These activities contribute directly to the functional FPGA implementation and therefore are not included in CoQ cost calculations, but listed here for completeness:

- Configuring hardware-side communication interfaces

- Implementing stable timing, routing, and buffering behavior to support reliable data exchange

- Supporting low-level integration with embedded software to ensure compatibility of data structures and communication paths

These activities represent productive engineering effort rather than quality-related cost components.

(C) LLM-Based Effort Estimation

As a third estimation approach, an LLM-based effort estimation method was applied to all hardware and FPGA-related CoQ activities. In this method, a LLM is used to provide a standalone, knowledge-driven baseline for each activity by interpreting detailed task descriptions and mapping them to generalized industrial benchmarks for FPGA design, hardware bring-up, and hardware–software integration learned during the model's training.

The same activity set used for milestone-based and expert-judgment-based estimation was provided to the LLM using a structured prompt. The model was asked to produce independent

"Most Likely" effort estimates in engineering hours and to provide a technical justification based on industrial FPGA development practices, such as typical time requirements for timing closure or integration debugging.

The LLM-based method operates independently from artifact-based milestones and internal expert judgment, as it does not rely on project-specific historical man-hour logs or specific design file counts. Instead, it represents an alternative perspective on effort estimation, particularly for activities that are difficult to quantify using traditional models, such as analyzing temperature-dependent instabilities or signal-level verification. This cross-domain validation mechanism ensures that the effort values for complex hardware tasks are consistent with broad industrial performance standards.

4.4.4 System/Management Cost Estimation Method

System-level and management-related activities do not produce measurable artifacts such as source code or detailed technical outputs. These tasks mostly involve coordination, communication, integration planning, and system-level evaluation. Because of this, size-based or artifact-based cost estimation methods cannot be applied. Similar observations are made in several CoQ studies, where managerial and system-level effort is estimated through expert judgment rather than quantitative metrics (Hong & Fong, 2003; Laporte, Berrhouma, Doucet, & Palza-Vargas, 2012; Slaughter, Harter, & Krishnan, 1998).

Accordingly, the engineering effort for System/Management activities is determined by integrating three independent estimation methods:

(A) A structured expert-judgement approach
(B) LLM-Based Effort Estimation

(A) Structured Expert Judgment: Engineers who regularly took part in system-level activities provided approximate duration estimates based on their practical industrial experience. When multiple inputs were available, a representative consensus value (typically the median or most frequently stated estimate) was used to ensure internal consistency (Jørgensen, Boehm, & Rifkin, 2009). These duration values were then multiplied by the relevant hourly engineer cost to compute the baseline CoQ.

System/Management Activities Estimated Using Expert Judgement

Prevention (System/Management)

- Maintaining the system-wide interface specification
- Requirement clarification meetings
- Planning integration sessions
- Defining the operational system flow

Appraisal (System/Management)

- Joint integration testing sessions
- End-to-end system validation
- Cross-team log and performance analysis

Internal Failure (System/Management)

- Repeated integration sessions due to mismatched assumptions
- Corrections to the system-wide interface specification after inconsistencies
- Re-coordination efforts caused by unexpected system behavior

External Failure (System/Management)

- System instability observed during field tests
- Timing or communication deviations appearing only under real conditions

(B) Standalone LLM-Based Industrial Estimation: As an independent calculation method, a LLM was used as a supplementary estimation approach for system and management-related activities. Written task descriptions, such as interface synchronization and cross-team requirements alignment, were provided to the model, and the resulting effort estimates were derived based on general systems engineering and technical management practices.

4.5 Cost of Quality Calculation Results

This section presents the CoQ calculation results for Project X. The results are organized by development domain in order to reflect the different characteristics of software, hardware, and system/management activities. Each subsection reports the calculated CoQ distributions based on the estimation methods described in Section 4.4.

4.5.1 Software CoQ Calculation Results

In this section, the CoQ for software activities is calculated using the estimation methods defined in Section 4.4. Activities that produced measurable source-code changes were evaluated using the simplified COCOMO model, while all non-code activities were estimated through structured expert judgement.

COCOMO-Based Effort and Cost Calculations

Based on the estimation framework defined in Section 4.4.1, the Basic COCOMO model was applied to software activities that produced measurable source-code changes. For each activity, the corresponding KSLOC value was derived from the actual modified or newly developed modules, and the effort was calculated using the embedded-mode COCOMO coefficients. The resulting person-month values were converted to engineering hours and then mapped to cost by distributing the effort across the engineers involved in the activity and applying their respective hourly rates.

The following subsections present the calculated effort and cost values for each COCOMO-based software activity.

Implementing UART and TCP/IP Drivers

- KSLOC: 0.250
- Effort (PM): $3{,}6 * (0{,}250)^{1.20}$ = 0,68
- Effort (Hours): $0{,}68 * 160$ = 109 hours
- Team: 2 Senior Engineers (each ≈ 55 h)
- Hourly Rate: Senior → 550 TL/hour
- Total Cost: $109 * 550$ = 59.950 TL

Developing the Packet Encoder / Decoder

- KSLOC: 0.482
- Effort (PM): $3.6 * (0{,}482)^{1.20}$ = 1,49
- Effort (Hours): $1{,}49 * 160$ = 238 hours
- Team: 1 Senior + 1 Junior Engineer (each ≈ 119 h)
- Hourly Rate: Senior → 550 TL/hour, Junior → 250 TL/hour
- Total Cost: $(119 * 550) + (119 * 250)$ = 95.200 TL

Fixing Protocol-Related Software Defects

- KSLOC: 0,116
- Effort (PM): $3.6 * (0{,}116)^{1.20}$ = 0.28
- Effort (Hours): $0{,}28 * 160$ = 45 hours
- Team: 1 Senior + 1 Junior Engineer (each ≈ 22,5 h)
- Hourly Rate: Senior → 550 TL/hour, Junior → 250 TL/hour
- Total Cost: $(22{,}5 * 550) + (22{,}5 * 250)$ = 18.000 TL

Algorithm Output Deviations

- KSLOC: 0,325
- Effort (PM): $3{,}6 \ * \ (0{,}325)^{1.20}$ = 0.97
- Effort (Hours): $0{,}97 \ * \ 160$ = 155 hours
- Team: 1 Senior + 1 Junior Engineer (each ≈ 77,5 h)
- Hourly Rate: Senior → 550TL/hour, Junior → 250 TL/hour
- Total Cost: $(77{,}5 \ * \ 550) \ + (77{,}5 \ * \ 250)$ = 62.000 TL

Correcting Simulation–Real Hardware Timing Mismatch

- KSLOC: 0,494
- Effort (PM): $3{,}6 \ * \ (0{,}494)^{1.20}$ = 1.58
- Effort (Hours): $1{,}58 \ * \ 160$ = 253 hours
- Team: 1 Senior + 1 Junior Engineer (each ≈ 126,5 h)
- Hourly Rate: Senior → 550 TL/hour, Junior → 250 TL/hour
- Total Cost: $(126{,}5 \ * \ 550) \ + (126{,}5 \ * \ 250)$ = 101.200 TL

Table 5.Calculation of Software Cost of Quality Activity Efforts

Activity	**CoQ Category**	**Estimation Method**	**Effort Breakdown**	**Total Hours**	**Total Cost**
Maintaining and updating the software-side interface specification	Prevention	Expert Judgement	1 Senior (≈20 h), specification update, packet format clarification, and alignment with implemented protocol behavior	20 hours	11.000 TL
Planning the RTOS-based architecture and communication driver structure	Prevention	Expert Judgement	1 Senior (≈25 h), task allocation, scheduling strategy, and driver architecture planning	25 hours	13.750 TL
Performing early modular decomposition and code-structure planning	Prevention	Expert Judgement	1 Senior (≈40 h), module boundary definition and dependency analysis	40 hours	22.000 TL
Designing the architecture of the software simulation environment,	Prevention	Expert Judgement	1 Senior + 1 Junior (≈15 h each), defining simulation environment architecture (scenario flow + communication interface + message structures) so verification could be done in absence of hardware	30 hours	12.000 TL
Unit testing of UART/TCP/IP drivers and protocol parsers	Appraisal	Expert Judgement	1 Senior + 1 Junior (≈30 h each), test execution and result evaluation	60 hours	24.000 TL

Table 5 cont.

Module-level functional testing	Appraisal	Expert Judgement	1 Junior (≈40 h), functional verification of individual software modules	40 hours	10.000 TL
Integration testing using the Main Board software simulation	Appraisal	Expert Judgement	1 Senior + 1 Junior (≈60 h each), integration execution and failure inspection, debugging in case of failures	120 hours	48.000 TL
Executing software-only simulation scenarios	Appraisal	Expert Judgement	1 Junior + 1 Senior (≈20 h each), message flow execution and behavior observation for both timing and integrity	40 hours	16.000 TL
Debugging issues identified during software–simulation or software–hardware integration sessions	Internal Failure	Expert Judgement	1 Senior + 1 Junior (≈35 h each), failure reproduction, log inspection, and root-cause isolation without code modification	70 hours	28.000 TL
Fixing protocol-related software defects	Internal Failure	COCOMO	KSLOC=0.116, PM=0.28, 1 Senior + 1 Junior (≈22.5 h each)	45 hours	18.000 TL
Updating the software-side interface specification	Internal Failure	Expert Judgement	1 Senior (≈20 h), correcting documented packet fields and timing assumptions based on integration findings	20 hours	11.000 TL
Correcting discrepancies between	Internal Failure	COCOMO	KSLOC=0.494, PM=1.58, 1 Senior + 1 Junior (≈126.5 h each)	253 hours	101.200 TL

Table 5 cont.

simulation behavior and real hardware timing					
A one-time system crash	External Failure	Expert Judgement	1 Senior (≈10 h), crash reproduction attempt, runtime log analysis, and system restart validation	10 hours	5.500 TL
Algorithm output deviations	External Failure	COCOMO	KSLOC=0.325, PM=0.97, 1 Senior + 1 Junior (≈77.5 h each)	155 hours	62.000 TL
Occasional packet corruption or loss	External Failure	Expert Judgement	1 Senior + 1 Junior (≈15 h each), packet trace inspection and communication integrity checks under field conditions	30 hours	12.000 TL
Implementing UART and TCP/IP drivers	Value Added	COCOMO	KSLOC=0.250, PM=0.68, 2 Senior (≈55 h each)	109 hours	59.950 TL
Developing the packet encoder/decode	Value Added	COCOMO	KSLOC=0.482, PM=1.49, 1 Senior + 1 Junior (≈119 h each)	238 hours	95.200 TL
Executing end-to-end message workflows	Value Added	Expert Judgement	1 Senior + 1 Junior (≈120 h each), coordinated end-to-end execution, cross-board validation, and runtime behavior confirmation	240 hours	96.000 TL
Implementing the software simulation tool	Value Added	Expert Judgement	1 Senior + 1 Junior (≈90 h each), simulation scenario setup, environment configuration, and validation against real system behavior	180 hours	72.000 TL

Table 6. presents the monetary and percentage distribution of software CoQ activities. The table summarizes how software development effort is distributed across prevention, appraisal, internal failure, external failure, and value-added categories, based on the total software CoQ cost.

Table 6.Distribution of Software Cost of Quality by Category

CoQ Category	**Cost (TL)**	**Percentage (%)**
Prevention	58.750	8.18
Appraisal	98.000	13.66
Internal Failure	158.200	22.05
External Failure	79.500	11.08
Value Added	323.150	45.03
Total	717.600	100

LLM Based Software CoQ Calculation Results

This subsection presents the CoQ calculation results obtained using the LLM-based effort estimation method for software-related activities.

For this approach, the same set of software activities and CoQ classifications defined in Section 4.5.1.1 were used. The number and role of engineers assigned to each activity were kept identical to the primary estimation, while the total effort values (in hours) were derived from LLM-generated estimates based on structured activity descriptions.

The structured prompt used to obtain these LLM-based effort estimates is provided in Appendix B, ensuring transparency and reproducibility of the estimation process.

Table 7.LLM Based Estimation for Efforts of Software Cost of Quality Activities

Activity	**CoQ Category**	**Number of Engineers Working**	**Total Hours (LLM Estimated)**	**Total Cost**
Maintaining and updating the software-side interface specification	Prevention	1 Senior	40 hours	22.000 TL
Planning the RTOS-based architecture and communication driver structure	Prevention	1 Senior	60 hours	33.000 TL
Performing early modular decomposition and code-structure planning	Prevention	1 Senior	48 hours	26.400 TL
Designing the architecture of the software simulation environment,	Prevention	1 Senior + 1 Junior	72 hours	28.800 TL
Unit testing of UART/TCP/IP drivers and protocol parsers	Appraisal	1 Senior + 1 Junior	80 hours	32.000 TL
Module-level	Appraisal	1 Junior	40 hours	10.000 TL

Table 7 cont.

functional testing				
Integration testing using the Main Board software simulation	Appraisal	1 Senior + 1 Junior	56 hours	22.400 TL
Executing software-only simulation scenarios	Appraisal	1 Junior + 1 Senior	48 hours	19.200 TL
Debugging issues identified during software–simulation or software–hardware integration sessions	Internal Failure	1 Senior + 1 Junior	96 hours	38.400 TL
Fixing protocol-related software defects	Internal Failure	1 Senior + 1 Junior	72 hours	28.800 TL
Updating the software-side interface specification	Internal Failure	1 Senior	32 hours	17.600 TL
Correcting discrepancies between simulation behavior and real hardware timing	Internal Failure	1 Senior + 1 Junior	80 hours	32.000 TL

Table 7 cont.

A one-time system crash	External Failure	1 Senior	24 hours	13.200 TL
Algorithm output deviations	External Failure	1 Senior + 1 Junior	48 hours	19.200 TL
Occasional packet corruption or loss	External Failure	1 Senior + 1 Junior	56 hours	22.400 TL
Implementing UART and TCP/IP drivers	Value Added	2 Senior	120 hours	66.000 TL
Developing the packet encoder/decode	Value Added	1 Senior + 1 Junior	64 hours	25.600 TL
Executing end-to-end message workflows	Value Added	1 Senior + 1 Junior	56 hours	22.400 TL
Implementing the software simulation tool	Value Added	1 Senior + 1 Junior	96 hours	38.400 TL

Table 8.Distribution of Software Cost of Quality by Category (for LLM Based Estimation)

CoQ Category	**Cost (TL)**	**Percentage (%)**
Prevention	110.200	21,28
Appraisal	83.600	16,14
Internal Failure	116.800	22,55
External Failure	54.800	10,58
Value Added	152.400	29,45
Total	517.800	100

4.5.2 Hardware CoQ Calculation Results

In this section, the CoQ for hardware-related activities is calculated based on milestone-driven effort allocation and structured expert judgement, as defined in Section 4.4. Since no detailed time-tracking data is available for FPGA development, the total block-design effort is distributed across hardware CoQ activities according to their relative contribution to the overall design, bring-up, and debugging process.

Table 9.Calculation of Hardware Cost of Quality Activities

Activity	**CoQ Category**	**Estimation Method**	**Effort Breakdown**	**Total Hours**	**Total Cost**
Initial pin-assignment and signal mapping	Prevention	Milestone-based	1 Senior Engineer (≈12 h), performing FPGA block-design level signal mapping and pin assignment as part of the initial communication subsystem configuration.	12 hours	6.600 TL
Preparation of timing and I/O constraints	Prevention	Milestone-based	1 Senior Engineer (≈6 h), defining timing and I/O constraints within the FPGA block design to ensure correct clocking and interface behavior.	6 hours	3.300 TL
Defining memory and bandwidth requirements	Prevention	Milestone-based	1 Senior Engineer (≈12 h), determining memory usage and bandwidth requirements during block design to support expected communication throughput.	12 hours	6.600 TL
Processing-system peripheral configuration & interconnect setup	Prevention	Milestone-based	1 Senior Engineer (≈10 h), configuring processing-system peripherals and interconnects within the FPGA block design to enable communication with external components.	10 hours	5.500 TL
Initial bring-up testing of the FPGA	Appraisal	Expert Judgement	1 Senior Engineer (≈8 h), loading the initial FPGA bitstream and verifying basic board operation	8 hours	4.400 TL

Table 9 cont.

prototype			and peripheral availability		
Timing and signal-behavior checks using tool-based timing analysis	Appraisal	Expert Judgement	1 Senior Engineer (≈8 h), observing signal behavior to confirm stable clocking and data paths	8 hours	4.400 TL
Correcting the DMA data-alignment issue requiring interface-logic updates and regeneration of the configuration bitstream	Internal Failure	Expert Judgement	1 Senior Engineer (≈20 h), tracing misaligned data reception, correcting interface logic, and regenerating the FPGA bitstream	20 hours	11.000 TL
Adjusting FPGA–software timing parameters when real hardware behavior diverged from design assumptions	Internal Failure	Expert Judgement	1 Senior Engineer (≈8 h), tuning handshake and buffering parameters after timing mismatches observed during integration	8 hours	4.400 TL
Investigating temperature-dependent instability observed during internal test cycles	Internal Failure	Expert Judgement	1 Senior Engineer (≈8 h), analyzing occasional communication issues under extreme temperature conditions	8 hours	4.400 TL

Table 9 cont.

Replacing the malfunctioning storage module during field testing	External Failure	Expert Judgement	1 Technician(≈3 h) + 1 Senior Engineer (≈1 h), on-site diagnosis, component replacement, and final functional verification	4 hours	1.090 TL
Investigating communication degradation caused by environmental or mechanical factors	External Failure	Expert Judgement	1 Senior Engineer (≈4 h), examining communication issues observed under field conditions and identifying environmental causes	4 hours	2.200 TL
Configuring FPGA-side communication interfaces	Value Added	Expert Judgement	1 Senior Engineer (≈16 h), configuring FPGA-side communication interfaces to support reliable data exchange	16 hours	8.800 TL
Providing stable timing, routing, and buffering behavior to support reliable data exchange	Value Added	Expert Judgement	1 Senior Engineer (≈8 h), establishing stable timing paths and buffering behavior for continuous data flow	8 hours	4.400 TL
Supporting low-level integration with embedded software to ensure compatibility of data structures	Value Added	Expert Judgement	1 Senior Engineer (≈8 h), verifying that FPGA-generated data structures, buffering behavior, and timing characteristics matched the expectations of the embedded software, and resolving minor interface mismatches observed during integration	8 hours	4.400 TL

Table 9 cont.

and communication paths					

Table 10. shows the monetary and percentage distribution of hardware CoQ activities. The table summarizes how hardware-related effort is distributed across prevention, appraisal, internal failure, external failure, and value-added categories, based on the total hardware CoQ cost.

Table 10.Distribution of Hardware Cost of Quality by Category

CoQ Category	**Cost (TL)**	**Percentage (%)**
Prevention	22.000	30.77
Appraisal	8.800	12.31
Internal Failure	19.800	27.70
External Failure	3.290	4.61
Value Added	17.600	24.61
Total	71.490	100

LLM Based Hardware CoQ Calculation Results

This subsection presents the CoQ calculation results obtained using the LLM-based effort estimation method for hardware-related activities.

For this approach, the same set of hardware activities and CoQ classifications defined in Section 4.5.2 were used. The number and role of engineers assigned to each activity were kept identical to the primary estimation, while the total effort values (in hours) were derived from LLM-generated estimates based on structured activity descriptions related to FPGA design, bring-up, and integration tasks.

The structured prompt used to obtain these LLM-based effort estimates is provided in Appendix B, ensuring transparency and reproducibility of the estimation process.

Table 11. LLM Based Estimation for Efforts of Hardware Cost of Quality Activities

Activity	**CoQ Category**	**Number of Engineers Working**	**Total Hours (LLM Estimated)**	**Total Cost**
Initial pin-assignment and signal mapping	Prevention	1 Senior Engineer	16 hours	8.800 TL
Preparation of timing and I/O constraints	Prevention	1 Senior Engineer	20 hours	11.000 TL
Defining memory and bandwidth requirements	Prevention	1 Senior Engineer	12 hours	6.600 TL
Processing-system peripheral configuration & interconnect setup	Prevention	1 Senior Engineer	16 hours	8.800 TL
Initial bring-up testing of the FPGA prototype	Appraisal	1 Senior Engineer	24 hours	13.200 TL
Timing and signal-behavior checks using tool-based timing analysis	Appraisal	1 Senior Engineer	16 hours	8.800 TL
Correcting the DMA data-alignment issue requiring interface-logic updates and regeneration of the configuration bitstream	Internal Failure	1 Senior Engineer	20 hours	11.000 TL
Adjusting FPGA–software timing parameters when real hardware behavior diverged from design assumptions	Internal Failure	1 Senior Engineer	16 hours	8.800 TL

Table 11 cont.

Investigating temperature-dependent instability observed during internal test cycles	Internal Failure	1 Senior Engineer	24 hours	13.200 TL
Replacing the malfunctioning storage module during field testing	External Failure	1 Technician + 1 Senior Engineer	8 hours	2.920 TL
Investigating communication degradation caused by environmental or mechanical factors	External Failure	1 Senior Engineer	20 hours	11.000 TL
Configuring FPGA-side communication interfaces	Value Added	1 Senior Engineer	16 hours	8.800 TL
Providing stable timing, routing, and buffering behavior to support reliable data exchange	Value Added	1 Senior Engineer	24 hours	13.200 TL
Supporting low-level integration with embedded software to ensure compatibility of data structures and communication paths	Value Added	1 Senior Engineer	20 hours	11.000 TL

Table 12.Distribution of Hardware Cost of Quality by Category (for LLM Based Estimation)

CoQ Category	Cost (TL)	Percentage (%)
Prevention	35.200	25,67
Appraisal	22.000	16,04
Internal Failure	33.000	24,07
External Failure	13.920	10,15
Value Added	33.000	24,07
Total	137.120	100

4.5.3 System/Management CoQ Calculation Results

System and management-related CoQ values are calculated exclusively using structured expert judgement, as these activities do not produce measurable technical artifacts such as source code or FPGA design outputs. The estimated effort reflects time spent on cross-team coordination, integration sessions, end-to-end validation, and issue resolution, and is expressed in working-day–level granularity to reflect realistic engineering practice.

Table 13.Calculation of System/Management Cost of Quality Activities

Activity	CoQ Category	Estimation Method	Effort Breakdown	Total Hours	Total Cost
Maintaining and updating the system-wide interface specification	Prevention	Expert Judgement	2 Senior Engineer(≈12 h each), maintaining and synchronizing the system-wide interface specification to ensure consistent message formats, timing rules, and communication assumptions across teams.	24 hours	13.200 TL
Requirement clarification meetings	Pre-	Expert	3 Senior Engineer (≈8 h each),	24	13.200 TL

Table 13 cont.

	ven-tion	Judge-ment	conducting cross-team clarification meetings to resolve ambiguities in packet contents, expected behaviors, and timing constraints before integration.	ho-urs	
Planning integration sessions	Pre-ven-tion	Expert Judge-ment	1 Senior Engineer , planning system-level integration sessions by defining test scenarios, execution order, logging strategy, and validation criteria.	12 ho-urs	6.600 TL
Defining the operational system flow	Pre-ven-tion	Expert Judge-ment	1 Senior Engineer + 1 Junior Engineer (≈10 h each), defining and documenting the expected end-to-end operational data flow across system components to guide integration and validation activities.	20 ho-urs	8.000 TL
Joint integration testing sessions	Appra-isal	Expert Judge-ment	2 Senior Engineer + 1 Junior Engineer (≈16 h each), executing system-level integration scenarios to verify cross-board message exchange, protocol compliance, and real-time timing behavior.	48 ho-urs	21.600 TL
End-to-end system validation	Appra-isal	Expert Judge-ment	1 Senior Engineer, validating complete end-to-end data transmission across the integrated system to confirm correct functional behavior.	16 ho-urs	8.800 TL
Cross-team log and performance analysis	Appra-isal	Expert Judge-ment	2 Senior Engineers (≈10 h each),analyzing cross-board logs and performance measure-	20 ho-urs	11.000 TL

Table 13 cont.

			ments to identify timing deviations, message delays, and protocol inconsistencies.		
Repeated integration sessions caused by mismatched assumptions	Internal Failure	Expert Judgement	1 Senior Engineer + 1 Junior Engineer (≈16h each), repeating system integration sessions to resolve mismatches in expected packet structures, message ordering, and timing behavior.	32 hours	12.800 TL
Corrections to the system-wide interface specification	Internal Failure	Expert Judgement	1 Senior Engineer, updating the system-wide interface specification to align documented behavior with observations from integrated execution.	16 hours	8.800 TL
Ongoing re-coordination efforts	Internal Failure	Expert Judgement	2 Senior Engineer (≈16 h each), conducting follow-up coordination and analysis sessions to address unexpected system responses and refine shared understanding between teams.	32 hours	17.600 TL
Temporary full-system instability during field operation	External Failure	Expert Judgement	1 Senior Engineer + 1 Junior Engineer (≈16 h each) ,diagnosing temporary system instability observed during field operation and restoring normal integrated behavior.	32 hours	12.800 TL
Unexpected behavior observed only under realistic multi-board execution	External Failure	Expert Judgement	2 Senior Engineer (≈12 h each), investigating communication timing deviations and message delays observed only during real-world multi-board execution.	24 hours	13.200 TL

Table 13 cont.

Establishing and operating the integrated multi-board system setup	Value Added	Expert Judgement	2 Senior Engineer + 1 Junior Engineer (≈20 h each) , establishing and operating the integrated multi-board system setup as the primary environment for system-level development and evaluation.	60 hours	27.000 TL
Defining and documenting the end-to-end operational system flow	Value Added	Expert Judgement	1 Senior Engineer, defining and documenting the system-level operational behavior, including execution states, interaction sequences, and expected system responses under different operating conditions.	16 hours	8.800 TL
Evaluating the real-time communication performance between the Communication Bridge and the Main Board	Value Added	Expert Judgement	2 Senior Engineer, (20 h each), evaluating end-to-end latency, packet integrity, transmission timing, and throughput under integrated operating conditions.integration	40 hours	22.000 TL

Table 14. summarizes how system-level effort is distributed across prevention, appraisal, internal failure, external failure, and value-added activities. Percentages are calculated relative to the total system/management CoQ cost to provide a clear view of effort allocation across quality categories.

Table 14.Distribution of System/Management Cost of Quality by Category

CoQ Category	Cost (TL)	Percentage (%)
Prevention	41.000	19.96
Appraisal	41.400	20.16
Internal Failure	39.200	19.09

Table 14 cont.

External Failure	26.000	12.65
Value Added	57.800	28.14
Total	205.400	100

LLM Based System/Management CoQ Calculation Results

This subsection presents the CoQ calculation results obtained using the LLM-based effort estimation method for system and management-related activities.

For this approach, the same set of system and management activities and CoQ classifications defined in Section 4.5.3 were used. The number and role of engineers assigned to each activity were kept identical to the primary estimation, while the total effort values (in hours) were derived from LLM-generated estimates based on structured activity descriptions focusing on coordination, integration, and end-to-end system validation tasks.

The structured prompt used to obtain these LLM-based effort estimates is provided in Appendix B, ensuring transparency and reproducibility of the estimation process.

Table 15.LLM Based Estimation for Efforts of System/Management Cost of Quality Activities

Activity	**CoQ Category**	**Number of Engineers Working**	**Total Hours (LLM Estimated)**	**Total Cost**
Maintaining and updating the system-wide interface specification	Prevention	2 Senior Engineer	80 hours	44.000 TL
Requirement clarification meetings	Prevention	3 Senior Engineer	72 hours	39.600 TL
Planning integration sessions	Prevention	1 Senior Engineer	32 hours	17.600 TL
Defining the operational system flow	Prevention	1 Senior Engineer + 1 Junior	64 hours	25.600 TL

Table 15 cont.

		Engineer		
Joint integration testing sessions	Appraisal	2 Senior Engineer + 1 Junior Engineer	96 hours	43.200 TL
End-to-end system validation	Appraisal	1 Senior Engineer	56 hours	30.800 TL
Cross-team log and performance analysis	Appraisal	2 Senior Engineers	72 hours	39.600 TL
Repeated integration sessions caused by mismatched assumptions	Internal Failure	1 Senior Engineer + 1 Junior Engineer	64 hours	25.600 TL
Corrections to the system-wide interface specification	Internal Failure	1 Senior Engineer	40 hours	22.000 TL
Ongoing re-coordination efforts	Internal Failure	2 Senior Engineer	48 hours	26.400 TL
Temporary full-system instability during field operation	External Failure	1 Senior Engineer + 1 Junior Engineer	48 hours	19.200 TL
Unexpected behavior observed only under realistic multi-board execution	External Failure	2 Senior Engineer	64 hours	35.200 TL
Establishing and operating the integrated multi-board system setup	Value Added	2 Senior Engineer + 1 Junior Engineer	80 hours	38.000 TL
Defining and documenting the end-to-end operational system flow	Value Added	1 Senior Engineer	40 hours	22.000 TL

Table 15 cont.

Evaluating the real-time communication performance between the Communication Bridge and the Main Board	Value Added	2 Senior Engineer	56 hours	30.800 TL

Table 16.Distribution of System/Management Cost of Quality by Category (for LLM Based Estimation)

CoQ Category	**Cost (TL)**	**Percentage (%)**
Prevention	126.800	27,59
Appraisal	113.600	24,71
Internal Failure	74.000	16,10
External Failure	54.400	11,84
Value Added	90.800	19,76
Total	459.600	100

4.6 Cost of Quality Analysis and Discussion

This section analyzes the CoQ results presented in Section 4.5 by examining the distribution of quality-related effort across software, hardware, and system/management domains. In addition to interpreting the primary CoQ results obtained through artifact-based estimation and structured expert judgement, this section explicitly the outcomes produced by the LLM-based effort estimation are also examined to understand their contribution to the overall analysis.

The LLM-based approach is not introduced as a replacement for traditional estimation techniques such as COCOMO or expert judgement. Instead, it is intentionally applied as an alternative method for estimating quality related effort when historical time tracking data, detailed effort logs, or measurable artifacts are For this reason, the comparison between the primary CoQ results and the LLM-based estimates focuses on how effort is distributed across CoQ categories rather than on exact numerical values.

Similarities in the resulting effort distributions are taken as an indication that the LLM-based approach can reflect realistic CoQ patterns in embedded system projects where quantitative data is scarce.

4.6.1 Software CoQ Profile

This subsection discusses the software related CoQ results presented in Section 4.5, with a particular focus on comparing the primary estimation results with the LLM-based CoQ estimates. The objective is not only to interpret how software effort is distributed across CoQ categories, but also to analyze why differences emerge between the two estimation approaches and what these differences imply in data-scarce embedded system environments.

Distribution Across CoQ Categories

According to the primary software CoQ results summarized in Table 6, Value Added activities represent the largest share of total software effort, accounting for about 45.03% of the overall software cost. This is followed by Internal Failure costs at 22.05%. Appraisal activities account for 13.66%, while External Failure and Prevention represent smaller portions at 11.08% and 8.18%, respectively.

The relatively high share of Internal Failure effort shows that a significant amount of software work is related to integration-driven rework. In Project X, these internal failures are mainly linked to protocol mismatches, timing differences between simulation results and actual hardware behavior, and corrective changes identified during combined software–hardware testing. This observation points to system integration as the main source of software quality issues, rather than isolated coding errors.

The software CoQ profile obtained from the LLM-based estimation, shown in Table 8, differs in terms of absolute distribution. In this case, Value Added activities account for 29.45% of the total effort, while Internal Failure (22.55%) and Appraisal (16.14%) together make up a larger combined share. Prevention costs are also estimated at a higher level, reaching 21.28% compared to the primary estimation.

These differences are clearly illustrated in Figure 1, which visually compares the percentage distribution of software CoQ categories for the primary estimation method (COCOMO + expert judgement) and the LLM-based estimation. Although the relative importance of internal and external failure activities remains comparable, noticeable differences appear in the distribution of Prevention, Appraisal, and Value Added effort between the two approaches.

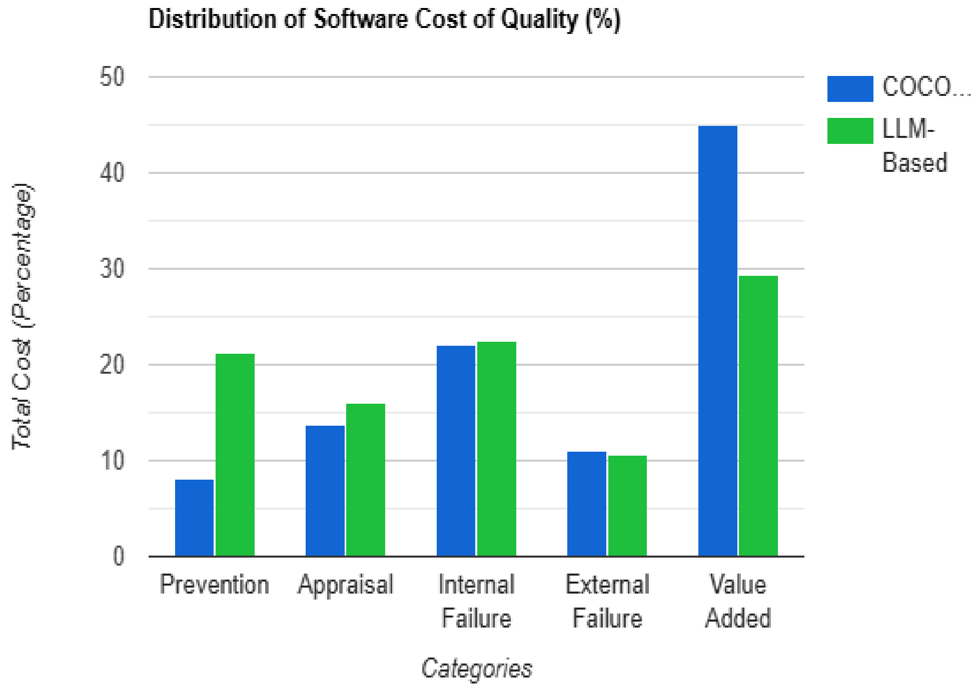

Figure 1.Distribution of Software Cost of Quality (%) Activities

Interpretation of Differences Between Estimation Approaches

The observed differences between the primary and LLM-based software CoQ distributions are related to the nature of the estimation approaches rather than to inconsistencies in the results. The two methods rely on different sources of information and therefore highlight different aspects of software effort.

The primary estimation relies on measurable project artifacts and direct expert judgement. COCOMO-based estimates mainly capture effort related to implementation and code-level corrections, such as driver development, packet encoder/decoder implementation, algorithm-related fixes, and timing-mismatch corrections. This approach therefore emphasizes Value Added and Internal Failure activities that are directly associated with observable code changes and integration rework in Project X.

Expert judgement complements this by capturing effort spent on activities such as interface clarification, execution of integration tests, log inspection, and system behavior analysis without significant code modification. In embedded system projects, these activities are common because achieving stable system behavior often requires analytical work and coordination in addition to coding.

In the primary estimation, activities estimated using the COCOMO model represent approximately 65–70% of the total software CoQ, while activities estimated through expert judgment account for the remaining 30–35%. This distribution reflects the combined presence of implementation focused and analysis oriented work in embedded software development.

The LLM-based estimation follows a different perspective, as it does not depend on project specific artifacts or measured code size. Instead, it reflects general development practices commonly associated with embedded software projects. From this viewpoint, greater relative emphasis is placed on Prevention and Appraisal activities, while Value Added activities receive a smaller share. This difference explains why the LLM-based results show higher proportions for Prevention and Appraisal compared to the primary estimation.

Software-Specific Observations

Overall, the software CoQ profile reflects an integration-driven development process. A considerable share of software cost is associated with Internal Failure, indicating that the primary challenges were related to aligning software behavior with hardware, simulation assumptions, and end-to-end system behavior rather than correcting isolated functional bugs.

External Failure costs are comparatively low and are mainly driven by field-observed behavior differences. This is consistent with the project's validation approach and current maturity level, as the system has not yet undergone prolonged or large-scale field operation.

While Prevention and Appraisal costs appear smaller in absolute terms in the primary estimation, they play an important supporting role by reducing ambiguity and limiting repeated corrective cycles during integration.

Taken together, this distribution indicates that software effort in Project X is currently focused on building functionality and resolving integration-related issues. This is consistent with the project's development stage, where core components are still being stabilized and interface assumptions are converging. As the system matures and integration stabilizes, a shift toward higher relative Prevention and Appraisal effort, accompanied by reduced Internal Failure costs, would be expected.

4.6.2 Hardware CoQ Profile

This subsection discusses the hardware related CoQ results presented in Section 4.5, with a particular focus on comparing the primary estimation results with the LLM-based CoQ estimates. Similar to the software domain, this analysis examines how hardware effort is distributed across CoQ categories and discusses the reasons behind the differences observed between the two estimation approaches in an embedded hardware development context with limited quantitative data.

Distribution Across CoQ Categories

According to the primary hardware CoQ results summarized in Table 10, Prevention activities account for the largest share of total hardware effort, at approximately 30.77% of the total hardware cost. Internal Failure follows with 27.70%, while Value Added activities represent 24.61%. Appraisal and External Failure contribute smaller portions, at 12.31% and 4.61%, respectively.

This distribution is related to the characteristics of hardware and FPGA development in Project X, where a large amount of effort is spent during early design and preparation stages. Prevention costs mainly correspond to early-stage FPGA design decisions, such as pin assignment, signal mapping, timing and I/O constraint definition, memory and bandwidth planning, and processing-system configuration. Because late-stage hardware changes are costly and risky, allocating more effort to these early activities is consistent with common industrial practice in embedded hardware development.

Internal Failure also represents a notable portion of the hardware CoQ. This indicates that, despite early preventive work, additional corrections were required during integration. In Project X, these internal failures were primarily caused by memory data-alignment issues, FPGA–software timing mismatches, and iterative adjustments to buffering or handshake logic discovered during integration. Such activities are mainly part of system tuning and stabilization rather than major design errors.

Value Added activities make up roughly one quarter of the total hardware effort and include tasks such as configuring communication interfaces, establishing stable timing and buffering behavior, and supporting low-level integration with embedded software. . Appraisal costs remain relatively limited and are mostly associated with initial bring-up tests and signal-level verification. External Failure costs are minimal, reflecting the small number of hardware issues that appeared only during field testing.

The primary hardware CoQ distribution is visualized in Figure 2 using the blue bars, which correspond to the milestone-based and expert-judgment-based estimation results.

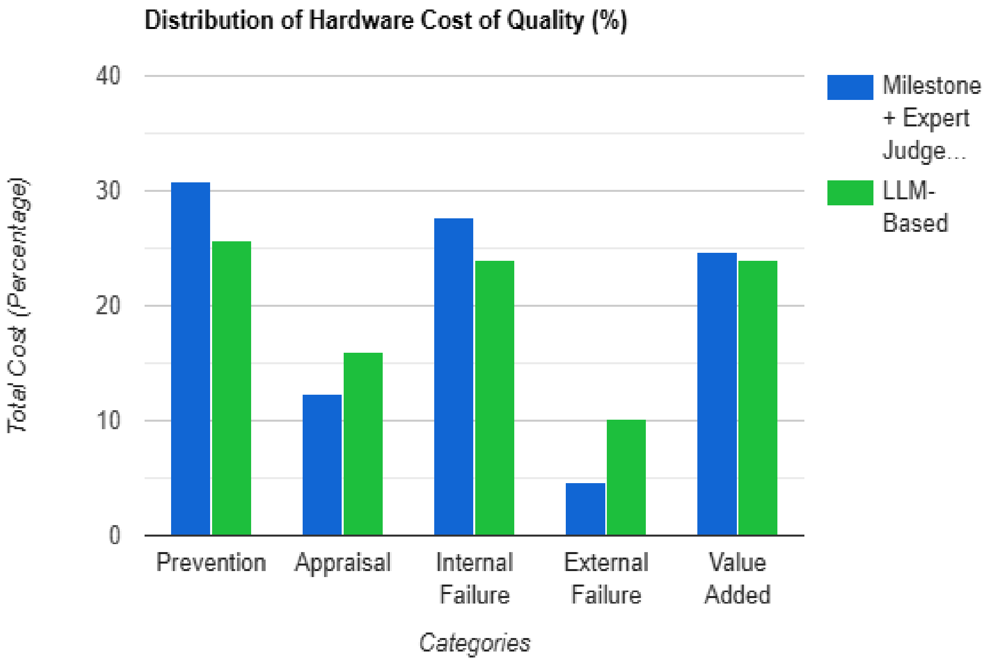

Figure 2.Distribution of Hardware Cost of Quality (%) Activities

Comparison with LLM-Based Estimation

The LLM-based hardware CoQ results, represented by the green bars in Figure 2, show a similar overall structure to the primary estimation, although some differences can be observed in the relative distribution of effort. In the LLM-based estimation, Prevention accounts for about 25.67% of the total hardware effort, while Internal Failure and Value Added each represent 24.07%. Appraisal and External Failure appear at higher levels, with values of 16.04% and 10.15%, respectively.

These differences are mainly related to the estimation approach used. The primary hardware CoQ estimation is based on concrete FPGA design milestones and expert judgment closely linked to the actual development and integration work carried out in Project X. As a result, this approach places more emphasis on early preventive activities and assigns a smaller share to External Failure, since only a limited number of hardware issues were observed during field testing.

In contrast, the LLM-based estimation reflects generalized industrial knowledge of FPGA and hardware development rather than project-specific milestones. From this perspective, activities related to Appraisal and External Failure are assigned higher relative weight, as hardware bring-up, signal-level validation, and environmental robustness are often emphasized as significant effort drivers in generic hardware development scenarios. This explains why External

Failure appears higher in the LLM-based results than in the primary estimation, even though field-observed hardware issues were limited in Project X.

Despite these differences, the overall hardware CoQ profile remains comparable across both estimation approaches. In both cases, Prevention, Internal Failure, and Value Added emerge as the dominant categories, indicating that early design work and integration related adjustments are the main contributors to quality-related cost in the hardware domain.

Hardware-Specific Observations

Overall, the hardware CoQ profile of Project X reflects a strongly prevention-oriented development approach. A significant portion of hardware effort was deliberately invested in early design and planning activities, such as FPGA block design, pin assignment, timing constraint definition, and resource planning. This early focus reduced the need for major design changes at later stages, which is especially important in FPGA-based embedded systems where late modifications are costly and risky.

Internal Failure costs indicate that some level of integration-related tuning was still required, particularly during hardware–software co-design and system integration. These activities mainly involved fine adjustments, such as timing alignment and interface tuning, rather than major redesign work. This indicates that early preventive work reduced the impact of hardware issues, even though some integration-related rework remained unavoidable.

External Failure costs remain relatively low in the primary estimation. Most hardware-related issues were identified and addressed during laboratory testing and early integration phases, which limited the number of problems observed during field operation This project specific behavior helps explain why the hardware CoQ profile of Project X differs from the distribution suggested by the LLM-based estimation, which reflects more general industrial development patterns.

Overall, the differences between the primary and LLM-based hardware CoQ profiles appear to be mainly related to the development strategy and execution characteristics of Project X, rather than to shortcomings of the estimation methods themselves.

4.6.3 System/Management CoQ Profile

This subsection discusses the system/management level CoQ results presented in Section 4.5, with a particular focus on comparing the primary estimation results with the LLM-based CoQ estimates. This analysis looks at how system-level effort is distributed across CoQ categories and discusses the reasons for the differences observed between the two estimation approaches in Project X.

Distribution Across CoQ Categories

According to the system and management CoQ results summarized in Table 14, effort is distributed fairly evenly across Prevention, Appraisal, Internal Failure, External Failure, and Value Added categories. Unlike the software and hardware domains, no single CoQ category dominates the system-level profile, and the effort is spread across different types of quality-related activities.

In the primary estimation, Value Added activities form the largest portion of system-level effort, accounting for roughly 28% of the total system and management CoQ. These activities include operating the integrated multi-board setup, defining end to end system behavior, and supporting the execution of integrated system scenarios. This reflects the amount of work required to enable and maintain stable system operation in Project X.

Prevention and Appraisal activities each account for around 20% of the system-level CoQ. Prevention work is mainly related to coordination focused tasks, such as maintaining the system-wide interface specification, clarifying requirements across teams, planning integration sessions, and aligning expectations between software and hardware teams. Appraisal activities reflect system-level verification efforts, including joint integration testing sessions, end-to-end system validation, and cross-team log and performance analysis. Together, these activities highlight the importance of early alignment and continuous validation in managing system complexity.

Internal Failure accounts for approximately 19% of the system/management CoQ. These costs are primarily driven by repeated integration sessions, corrections to system-wide specifications, and ongoing re-coordination when mismatches between documented assumptions and observed system behavior were discovered. This indicates that system-level assumptions changed over time as the system was used in integrated operation.

External Failure represents a smaller but still noticeable share of the system CoQ at approximately 13%. These costs correspond to issues that became visible only during field operation, such as temporary system instability or unexpected behavior under realistic multi-board execution. The relatively limited share of External Failure suggests that most system-level issues were identified before extended field use.

The primary system/management CoQ distribution is illustrated by the blue bars in Figure 3.

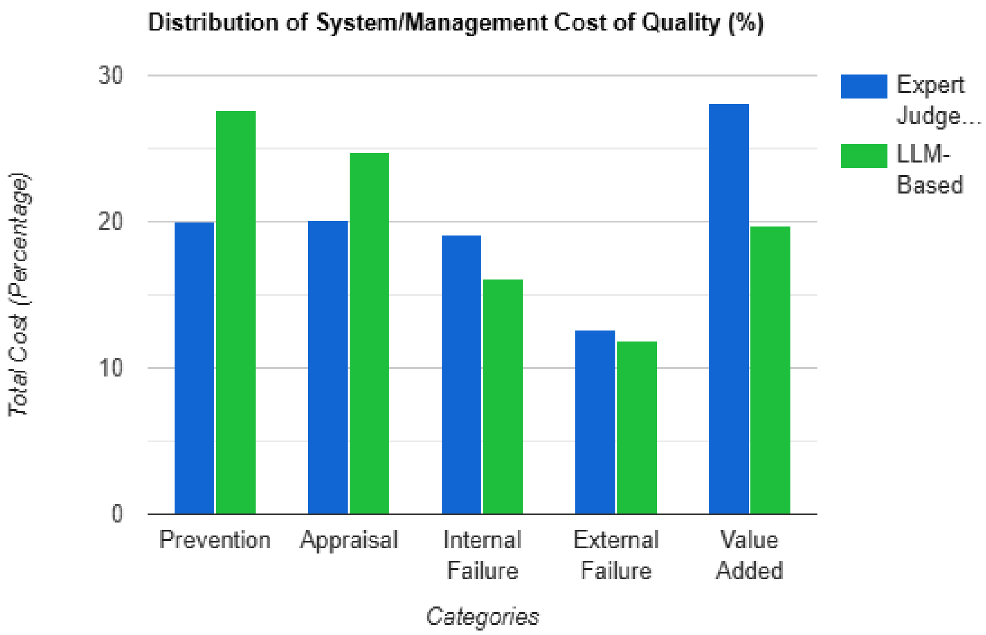

Figure 3.Distribution of System/Management Cost of Quality (%) Activities

Comparison with LLM-Based Estimation

The LLM-based system/management CoQ results, shown by the green bars in Figure 3, show a different distribution of effort across the CoQ categories. In this estimation, Prevention and Appraisal account for relatively larger portions of system-level effort, at approximately 27% and 25%, respectively. Value Added activities represent a smaller share at around 20%, while Internal Failure and External Failure contribute about 16% and 12%.

These differences can be explained by the nature of system-level work in Project X. The primary estimation reflects how effort was actually spent, where significant time was required to actively operate and support the integrated system, resolve real-time coordination issues, and adapt system behavior as integration progressed. Under these conditions, Value Added activities appear more visible in the primary results.

In contrast, the LLM-based estimation follows a more generic view of system/management activities in embedded projects. From this viewpoint, greater emphasis is placed on prevention- and appraisal-related work, such as structured coordination, early alignment between teams, and formal validation activities. This perspective leads to higher Prevention and Appraisal shares in the LLM-based results compared to the primary estimation.

System-Specific Observations

The system/management CoQ profile in Project X shows that a large part of the effort is related to coordination and integration activities. A notable share of system-level cost is linked to aligning teams, validating integrated behavior, and resolving mismatches between expected and observed system behavior.

The presence of Internal Failure at the system level indicates that many quality-related issues arose from interactions between components and teams rather than from isolated defects in individual elements. External Failure costs remain relatively low, suggesting that most system-level problems were identified and addressed before extensive field operation.

Differences between the primary and LLM-based system and management CoQ profiles appear to be related to the way Project X was executed. In Project X, direct involvement in integration activities and continuous system operation required more effort than what is typically assumed in generalized system-level estimation approaches. This explains why the observed CoQ structure differs from the profile suggested by the LLM-based estimation.

5. CONCLUSIONS

5.1 Summary of Key Findings

This study examined how CoQ measurement can be applied in hardware–software integrated embedded systems where detailed quantitative effort data is limited or not available. Rather than focusing only on the CoQ results of a single project, the study aimed to understand how CoQ analysis can be adapted to realistic embedded system environments. This motivation was derived from the systematic literature review presented in Section 2, which showed that most existing CoQ frameworks are software focused and often depend on detailed time-tracking data that is difficult to obtain in industrial embedded projects.

The literature review indicated that many CoQ measurement approaches have been proposed to classify and analyze quality-related activities (RQ1). However, their use in embedded and hardware-dependent systems is restricted by common challenges such as tight hardware–software coupling, limited traceability, and a lack of detailed effort data (RQ2). At the same time, previous studies consistently report that CoQ analysis can support better understanding of the main causes of rework and process improvement decisions (RQ3). Despite this, practical examples showing how CoQ methods can be applied under real embedded system constraints remain relatively scarce.

To explore this gap, an industrial case study was carried out on Project X. The study focused on applying a practical and flexible CoQ approach in a data scarce embedded environment. Instead of aiming for a fully quantitative or financially precise model, the analysis adopted a domain-aware structure by separating software, hardware, and system/management activities and by combining different estimation approaches to capture quality related effort.

The case study demonstrates that meaningful CoQ insights can be obtained even in the absence of detailed historical data, provided that estimation methods are aligned with the characteristics of each engineering domain. Software activities were found to be primarily influenced by integration-driven rework and ongoing development, hardware activities were shaped by early preventive design decisions, and system-level activities were dominated by coordination and end to end validation effort. These observations highlight how different quality cost drivers emerge across domains depending on integration maturity and responsibility boundaries.

Overall, the study shows that the value of CoQ analysis in embedded projects lies less in exact numerical accuracy and more in its ability to reveal where and why quality-related effort is consumed. By explicitly highlighting where effort is spent on system integration, cross-team coordination, and early preventive activities, the applied CoQ framework helps teams clearly identify process weaknesses and discuss improvement opportunities, even in situations where detailed measurement or time-tracking data is not available.

From this perspective, Project X is presented as an illustrative case rather than a unique example. The study shows that CoQ can be used as a qualitative and decision support oriented framework in embedded system development, helping to narrow the gap between theoretical CoQ models and the practical conditions of hardware–software integrated projects.

5.2 Methodological Discussions

This study follows a pragmatic methodological approach to CoQ analysis that is shaped by the practical limitations identified in the literature review. Instead of relying on a single, fully quantitative CoQ model, the methodology combines different estimation techniques and qualitative reasoning to support CoQ analysis in a hardware–software integrated embedded system where detailed effort data is limited.

One important methodological decision was to consider software, hardware, and system/management activities as separate CoQ domains. This choice directly addresses a common limitation highlighted in the literature, namely the strong focus of existing CoQ frameworks on software activities. Treating all domains in the same way would have required forcing fundamentally different types of work into a single estimation logic. By separating the domains, each type of activity can be analyzed using methods that better match its development characteristics.

For the software domain, an effort-based estimation approach combining a simplified COCOMO model with expert judgement was adopted. This choice was motivated by the availability of measurable artifacts, such as source code size, and by the relatively higher maturity of software effort estimation models. In contrast, hardware and system/management activities lack comparable quantitative artifacts. For these domains, milestone-based decomposition and structured expert judgement were used to estimate quality-related effort. While these approaches introduce subjectivity, they reflect common industrial practice in embedded system development and are consistent with the constraints reported in the literature.

A key methodological challenge addressed in this study is data scarcity, which was identified in the SLR as a major obstacle to CoQ measurement in embedded systems. To explore how

CoQ analysis can still be performed under such conditions, an LLM-based estimation approach was intentionally introduced as an alternative reference method. The purpose of this approach was not to replace traditional estimation techniques or to produce more accurate numerical results, but to examine whether a generalized, experience-based estimation could capture meaningful CoQ structures in the absence of project-specific data.

The comparison between the primary estimation results and the LLM-based estimates revealed both similarities and differences. Importantly, the analysis showed that while absolute values differ, the dominant CoQ categories and overall effort distribution patterns remain largely consistent across approaches. Where differences occur, they can be explained by project-specific execution characteristics and by the tendency of generalized estimation methods to emphasize preventive and appraisal activities. From a methodological perspective, these observations support the use of LLM-based estimation as a complementary and exploratory tool for CoQ analysis in data-scarce environments, rather than as a substitute for established methods.

Another important methodological aspect of this study is how the CoQ results are interpreted. CoQ is not used as a precise accounting or cost calculation tool. Instead, it is applied as an analytical framework to understand where quality-related effort is spent and why. This way of using CoQ helps make integration effort, rework, and coordination activities visible and supports discussions about process weaknesses and improvement opportunities. As shown in the case study, even approximate CoQ distributions can provide useful insights when they are interpreted together with the project context and development stage.

Overall, the methodological choices in this study aim to balance analytical accuracy with practical applicability. Instead of focusing on exact numerical values, the approach emphasizes feasibility and clarity, taking into account the different characteristics of software, hardware, and system-level activities. This allows CoQ analysis to be applied in embedded system projects where detailed effort tracking and traditional measurement assumptions are difficult to maintain. In this way, the study provides a practical example of how CoQ analysis can be used in hardware–software integrated environments under realistic project conditions.

5.3 Limitations of the Study

While this study provides practical insight into the application of CoQ analysis in hardware–software integrated embedded systems, several limitations should be acknowledged.

First, the findings of this study are based on a single industrial case study. Although Project X represents a realistic example of an embedded communication system, the observed CoQ distributions and process characteristics may not be directly generalizable to all embedded projects. Differences in organizational structure, development maturity, application domain, or regulatory environment can influence how quality related effort is distributed. For this reason, the results are intended to be illustrative rather than broadly generalizable.

Second, the study relies on approximate effort estimation methods instead of detailed time-tracking data. Software-related activities were estimated using a simplified COCOMO-based approach combined with expert judgment, while hardware and system or management activities were estimated through milestone-based decomposition and expert input. Although these methods are commonly used in industrial embedded projects, they involve a degree of subjectivity and limit numerical precision. As a result, the reported CoQ values should be interpreted as indicative effort distributions rather than exact cost figures.

A further limitation is related to the use of LLM-based estimation. The effort values produced by the LLM are based on generalized knowledge of embedded system development and do not reflect project-specific constraints, organizational practices, or detailed technical artifacts. This can lead to differences in the relative emphasis of certain CoQ categories, such as Prevention or Appraisal, compared to the primary estimation. For this reason, the LLM-based approach is used only as a complementary and exploratory reference in this study.

Another limitation arises from the confidential nature of the project. Since Project X was developed in a defense system environment, access to detailed effort records and technical data was restricted. This limited the level of detail that could be used for quantitative analysis and reporting. Such constraints are common in defense and safety-critical embedded projects and further highlight the need for flexible and qualitative CoQ analysis approaches.

Finally, the scope of the study focuses on the development and early validation phases of Project X. External Failure costs observed in later operational or long-term field usage are therefore not fully represented. As a result, the reported External Failure distributions may change as the system matures and is exposed to extended real-world operation.

Despite these limitations, the study demonstrates that meaningful CoQ analysis can still be conducted under realistic industrial constraintsThese limitations provide context for interpreting the results and indicate that the findings reflect the specific conditions of Project X and should therefore be treated as exploratory rather than definitive.

5.4 Future Work

This study can be extended in several ways. The same CoQ analysis approach could also be applied to other embedded system projects, including projects from application areas such as automotive or aerospace. Applying the approach in different projects would allow observations to be made across varying organizational setups, technical constraints, and regulatory contexts.

Another possible extension concerns the time span of the analysis. The results presented in this study mainly reflect the development and early phases of Project X. Repeating the analysis during later life-cycle stages, such as extended field operation or maintenance, could provide further insight into how External Failure costs and long-term quality-related effort evolve over time.

Future studies could also examine ways to improve data availability in embedded projects without introducing heavy process overhead. For example, lightweight effort tracking, simplified defect classification, or structured integration logs could be evaluated to see whether they improve CoQ analysis while remaining compatible with industrial and defense industry constraints.

In addition, the use of LLM-based estimation could be explored further. In this study, the LLM-based approach was used only as a complementary reference. Further work could examine how such estimates change when limited project-specific information is provided or when they are combined with historical data from multiple projects.

These extensions would allow the CoQ analysis approach used in this study to be examined under a wider range of project conditions and development stages.

REFERENCES

[1] Slaughter, S. A., Harter, D. E., & Krishnan, M. S. (1998). Evaluating the cost of software quality. Communications of the ACM, 41(8), 67-73.

[2] Ahmad, S. F., & Samat, P. A. (2016). Extraction Cost of Quality and Testing in Software Project. Faculty of Computer Science and Information Technology, Universiti Putra Malaysia.

[3] Ramasubbu, N., & Kemerer, C. F. (2019). Integrating Technical Debt Management and Software Quality Management Processes: A Framework and Field Tests. IEEE Transactions on Software Engineering, 45(6), 615–633

[4] O. Demirors, O. Yildiz and A. Selcuk Guceglioglu, "Using cost of software quality for a process improvement initiative," Proceedings of the 26th Euromicro Conference. EUROMICRO 2000. Informatics: Inventing the Future, Maastricht, Netherlands, 2000, pp. 286-291 vol.2, doi: 10.1109/EURMIC.2000.874440.

[5] A. S. Guceglioglu and O. Demirors, "The Application of a New Process Quality Measurement Model for Software Process Improvement Initiatives," 2011 11th International Conference on Quality Software, Madrid, Spain, 2011, pp. 112-120, doi: 10.1109/QSIC.2011.29.

[6] A. Dikici, O. Turetken and O. Demirors, "A Case Study on Measuring Process Quality: Lessons Learned," 2012 38th Euromicro Conference on Software Engineering and Advanced Applications, Cesme, Turkey, 2012, pp. 294-297, doi: 10.1109/SEAA.2012.26.

[7] Laporte, C. Y., Berrhouma, N., Doucet, M., & Palza-Vargas, E. (2012). Measuring the Cost of Software Quality of a Large Software Project at Bombardier Transportation – A Case Study. Software Quality Professional, Pre-publication version

[8] Ö. Eren, B. Özkan and O. Demirörs, "PL FSM: An Approach and a Tool for the Application of FSM in SPL Environments," 2015 41st Euromicro Conference on Software Engineering and Advanced Applications, Madeira, Portugal, 2015, pp. 119-126, doi: 10.1109/SEAA.2015.58.

[9] B. Kitchenham and S. Charters, *Guidelines for Performing Systematic Literature Reviews in Software Engineering*, EBSE Technical Report, Keele University, 2007.

[10] ISO/IEC/IEEE 12207:2017 – *Systems and Software Engineering — Software Life Cycle Processes.* Geneva, Switzerland: International Organization for Standardization, 2017.

[11] C. P. Hollocker, "Finding the cost of software quality," in IEEE Transactions on Engineering Management, vol. EM-33, no. 4, pp. 223-228, Nov. 1986, doi: 10.1109/TEM.1986.6447683.

[12] Y. Yuan and S. Gu, "Research and Establishment of Quality Cost Oriented Software Testing Model," *2006 Canadian Conference on Electrical and Computer Engineering*, Ottawa, ON, Canada, 2006, pp. 2410-2415, doi: 10.1109/CCECE.2006.277513.

[13] G. Y. Hong and A. C. M. Fong, "Cost driven six sigma analysis for CE software development," 2003 IEEE International Conference on Consumer Electronics, 2003. ICCE., Los Angeles, CA, USA, 2003, pp. 184-185, doi: 10.1109/ICCE.2003.1218877.

[14] B. A. Kitchenham, "Systematic reviews," 10th International Symposium on Software Metrics, 2004. Proceedings., Chicago, IL, USA, 2004, pp. xii-xii, doi: 10.1109/METRIC.2004.1357885.

[15] Boehm, B.W. (1981) Software Engineering Economics. Prentice-Hall, Englewood Cliffs, NJ.

[16] Jørgensen, Magne & Boehm, Barry & Rifkin, S.. (2009). Software Development Effort Estimation: Formal Models or Expert Judgment?. Software, IEEE. 26. 14 - 19. 10.1109/MS.2009.47.

[17] M. Kaya and O. Demirors, "E-Cosmic: A Business Process Model Based Functional Size Estimation Approach," 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications, Oulu, Finland, 2011, pp. 404-410, doi: 10.1109/SEAA.2011.60.

[18] B. Z. Demiray and B. Arslan, "Test Cost-Test Quality Modeling For Adaptive Test," 2022 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), Cluj-Napoca, Romania, 2022, pp. 1-5, doi: 10.1109/AQTR55203.2022.9801935.

[19] H. Jiang, V. Owall and H. Ardo, "Real-Time Video Segmentation with VGA Resolution and Memory Bandwidth Reduction," 2006 IEEE International Conference on Video and Signal Based Surveillance, Sydney, NSW, Australia, 2006, pp. 104-104, doi: 10.1109/AVSS.2006.95.

APPENDIX A: LIST OF PRIMARY STUDIES (SLR)

The following 14 publications were selected as primary studies for the Systematic Literature Review conducted in this study:

ID	Author(s) & Year	Article Title	Publication Source	Research Domain
P1	Slaughter et al. (1998)	Evaluating the cost of software quality	Communications of the ACM	IT Systems Integration
P2	Ramasubbu & Kemerer (2019)	Integrating Technical Debt Management and Software Quality Management Processes	IEEE Transactions on Software Engineering	Commercial Software
P3	Ahmad & Samat (2016)	Extraction Cost of Quality and Testing in Software Project	Universiti Putra Malaysia	Government IT Projects
P4	Yuan & Gu (2006)	Research and Establishment of Quality Cost Oriented Software Testing Model	IEEE CCECE	Software Testing Systems (Card Services)
P5	Laporte et al. (2012)	Measuring the Cost of Software Quality of a Large Software Project at Bombardier Transportation	Software Quality Professional	Real-Time Embedded (Railway Control)
P6	Hong & Fong (2003)	Cost driven six sigma analysis for CE software development	IEEE ICCE	Consumer Electronics & Telecom
P7	Demirors et al. (2000)	Using cost of software quality for a process improvement initiative	Euromicro Conference	Public Sector IT Projects

P8	Gucegli-oglu & Demirors (2011)	The Application of a New Process Quality Measurement Model (PQMM)	IEEE International Conference on Quality Software	Software Process Quality (Defense & IT)
P9	Dikici et al. (2012)	A Case Study on Measuring Process Quality: Lessons Learned	Euromicro SEAA	Project Management Processes
P10	Eren et al. (2015)	PL FSM: An Approach and a Tool for the Application of FSM in SPL Environments	Euromicro SEAA	Embedded Defense Systems (ASELSAN)
P11	Kaya & Demirors (2011)	E-Cosmic: A Business Process Model Based Functional Size Estimation Approach	Euromicro SEAA	Public Information Systems
P12	Demiray & Arslan (2022)	Test Cost-Test Quality Modeling For Adaptive Test	Euromicro Conference on Software Engineering and Advanced Applications	Digital IC Testing (Hardware/ASIC)
P13	Jiang et al. (2006)	Real-Time Video Segmentation with VGA Resolution and Memory Bandwidth Reduction	IEEE AVSS	Real-Time FPGA Systems
P14	Hollocker (1986)	Finding the cost of software quality	IEEE Transactions on Engineering Management	General Software Industry

APPENDIX B: LLM PROMPTS FOR EFFORT ESTIMATIONS

LLM Prompt for Software Domain (Effort Estimation)

Role

You are a senior software and systems engineer with extensive industrial experience in embedded systems, real-time software, and hardware–software integrated projects. You have participated in multiple FPGA-based and communication-oriented embedded system developments.

Task

You will be given a list of software-related activities performed in an embedded communication project.
For each activity, estimate the total engineering effort required in hours.

Important Constraints

- The number and role of engineers are fixed as specified for each activity.
- Do not change, optimize, or reinterpret team size or engineer roles.
- Your task is only to estimate the total duration (in hours) required to complete each activity.

Project Context

- The project is an FPGA-based real-time embedded communication system.
- The software includes RTOS-based architecture, communication drivers (UART/TCP-IP), protocol handling, and simulation-based testing.
- The project is industrial, production-oriented, and under active integration (not academic prototyping).
- Engineers are experienced, but system-level integration and timing constraints introduce non-trivial complexity.

Output Requirements

For each activity, provide:

1. Estimated total effort (in hours)

2. A short justification (1–2 sentences) explaining why this duration is reasonable based on industrial practice

Output Format

Present the results in a clear table with the following columns:

- Activity
- Estimated Total Hours
- Justification

Activities

[List of software-related activities will be inserted here, including the fixed number and role of engineers for each activity.]

LLM Prompt for Hardware Domain (Effort Estimation)

Role

You are a senior hardware and FPGA engineer with extensive industrial experience in FPGA-based embedded systems, hardware–software co-design, and real-time communication platforms. You have participated in multiple projects involving FPGA block design, bring-up, timing closure, and system-level integration.

Task

You will be given a list of hardware-related activities performed in an embedded communication project.

For each activity, estimate the **total engineering effort required in hours**.

Important Constraints

- The **number and role of engineers are fixed** as specified for each activity.
- **Do not change or reinterpret team size or engineer roles.**
- Your task is **only** to estimate the **total duration (in hours)** required to complete each activity.

Project Context

- The project is an FPGA-based real-time embedded communication system.

- Hardware activities include FPGA block design, pin assignment, timing and I/O constraint definition, bring-up testing, signal and timing analysis, and hardware–software integration.
- The project is industrial and production-oriented, not an academic prototype.
- Engineers are experienced, but real-hardware behavior, timing margins, and integration effects introduce non-trivial complexity.

Output Requirements

For each activity, provide:

1. Estimated total effort (in hours)
2. A short justification (1–2 sentences) explaining why this duration is reasonable based on industrial FPGA development practice

Output Format

Present the results in a clear table with the following columns:

- Activity
- Estimated Total Hours
- Justification

Activities

[List of hardware-related activities will be inserted here, including the fixed number and role of engineers for each activity.]

LLM Prompt for System/Management Domain (Effort Estimation)

Role

You are a senior systems engineer and technical manager with strong industrial experience in embedded systems, system integration, and cross-team coordination. You have led multiple hardware–software integrated projects involving requirement alignment, system-level validation, and field testing.

Task

You will be given a list of system and management-related activities performed in an embedded communication project.

For each activity, estimate the total engineering effort required in hours.

Important Constraints

- The number and role of engineers are fixed as specified for each activity.
- Do not change or reinterpret team size or engineer roles.
- Your task is **only** to estimate the **total duration (in hours)** required to complete each activity.

Project Context

- The project is a multi-board, FPGA-based embedded communication system.
- System and management activities include interface definition, requirement clarification, integration planning, joint integration sessions, log and performance analysis, end-to-end validation, and field-observed issue resolution.
- The project is industrial, integration-driven, and under active development.
- Coordination effort is significant due to cross-team dependencies and real-time system behavior.

Output Requirements

For each activity, provide:

1. Estimated total effort (in hours)
2. A short justification (1–2 sentences) explaining why this duration is reasonable based on industrial systems engineering practice

Output Format

Present the results in a clear table with the following columns:

- Activity
- Estimated Total Hours
- Justification

Activities

[List of system/management-related activities will be inserted here, including the fixed number and role of engineers for each activity.]