The Journal of Systems & Software 201 (2023) 111678

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Empirical investigation in embedded systems: Quality attributes in general, maintainability in particular[☆]

Simona Motogna [*], Andreea Vescan, Camelia Şerban

Babeş-Bolyai University, Department of Computer Science, M. Kogălniceanu 1, Cluj-Napoca, 400084, Romania

ARTICLE INFO

Article history:
Received 23 March 2022
Received in revised form 14 January 2023
Accepted 11 March 2023
Available online 16 March 2023

Keywords:
Quality factors
Empirical study
Embedded systems

ABSTRACT

The quality of software systems is an important aspect, especially for embedded systems, thus strategies and actions for analyzing the trade-off between various quality attributes need to be improved.

Objectives: We target firstly to determine which quality attributes are important in embedded systems, and secondly to inquire about maintainability in particular, emphasizing the practices that are associated with it, i.e., coding rules, conventions, documentation, code review, and refactoring.

Method: We used interviews and surveys as means to investigate practitioners' points of view and practices. Applying quantitative and qualitative analysis, we explored a general perspective of quality attributes in embedded systems, followed by specific practices related to the maintainability attribute.

Results: At the general perspective level, we learned that the importance of security and safety is extended to all embedded systems, while maintainability remains of major importance, and there is a diversity of methods used to assure the quality of systems throughout the development cycle. At the maintainability-specific level, we learned that code review and refactoring are the most used practices and that the related activities are performed in a variety of ways.

Conclusions: Our work recognizes various quality attributes as being important with different priorities, respectively analyses which maintainability-related activities are used.

1. Introduction

The complexity of the software systems increases every day, along with the never-ending changes that impact the quality of the systems. Managing the quality attributes and conducting analysis to increase the software quality are challenging tasks, needing various strategies and actions to be taken even from the start of the development. In embedded systems, these strategies are even more demanding, at least for two reasons: the interplay between different software and hardware components, different platforms and programming languages, respectively, the specificity of applications that might impose extra conditions, in the case of real-time and safety critical systems (Carrozza et al., 2018; Papadopoulos et al., 2018).

The quality of embedded systems is an important aspect, since they are extremely ubiquitous nowadays, being used from cars to power plants, in which runtime errors can potentially be catastrophic, causing serious damage to the environment, to human lives, or financial losses. The market pressure to rapidly improve and release new products, the limited hardware resources on which the software is deployed, as well as the rapid evolution of hardware lead to the necessity to establish trade-offs regarding the importance of quality attributes to safeguard the levels of criticality against other non-critical qualities.

Various investigations (Sherman, 2008; Sas and Avgeriou, 2020; Carrozza et al., 2018) reported essential quality attributes for embedded systems, which suggest that the existing quality models for "classical" software systems may not be suitable for software embedded systems, due to factors related to complexity, size and especially to dependencies between hardware and software parts of the systems.

Despite the research results obtained so far, there is no overall accepted set of essential quality attributes for embedded systems. Even more, due to the heterogeneity of the teams, software engineering practices related to quality attributes are very diverse. *Studies addressing quality attributes of these systems can contribute to a complete definition of a software quality model for embedded systems.* The motivation is strengthened even more by the fact that in industry, processes and practices are in many cases specific requirements given by standards, client demands or

☆ Editor: Dr. P. Lago.

* Corresponding author.

E-mail addresses: simona.motogna@ubbcluj.ro (S. Motogna), andreea.vescan@ubbcluj.ro (A. Vescan), camelia.serban@ubbcluj.ro (C. Şerban).
URLs: http://www.cs.ubbcluj.ro/~motogna (S. Motogna), http://www.cs.ubbcluj.ro/~avescan (A. Vescan), http://www.cs.ubbcluj.ro/~camelia (C. Şerban).

company rules, lacking an overall perspective and inter-relation between different quality attributes. In consequence, *there is a lack of uniform strategies on how the quality attributes should be achieved and monitored.*

Previous studies (Arvanitou et al., 2017; Ampatzoglou et al., 2016) identified the critical role of maintainability in embedded systems. Teams working on embedded projects are continuously preoccupied to assure high maintainability, due to the following particular characteristics of embedded systems: continuous interdependency between hardware and software components, which require more effort to maintain even during the development of the system, and for the long-term maintenance, the rapid deprecation of hardware components implies an extra effort for adaptive maintenance. *Thus, more research is needed to report the maintainability practices for these systems.* Also, as our investigation revealed, there exist various opinions about concepts and practices involved in maintaining embedded systems, and people working on these systems express their interest in a better understanding of them.

In order to overcome the above-mentioned limitations regarding quality attributes and practices associated with maintainability in embedded systems, the goal of the paper is to investigate, through an empirical study, how embedded systems engineers prioritize the importance of quality attributes, on one hand, and how they manage maintainability, on the other hand. We augmented our investigation by examining the maintainability factor by identifying which are the practices associated with it, and what activities and tools are used for controlling it. We aim also to see if there are any differences between managers and developers regarding quality attributes in embedded systems.

We employed qualitative and quantitative analysis on information collected through interviews and surveys with management and development teams of several embedded projects, since the voice of practitioners is essential in developing new methods that can have an impact on the quality of software.

The findings of this study revealed that the most important quality attributes in embedded system development are maintainability, safety, and security, along with performance and energy efficiency. The study also emphasized the practices associated with maintainability: coding rules, conventions and documentation, code review, and refactoring.

The paper is organized as follows: Section 2 provides the needed theoretical concepts that we operate with and reports on similar work from literature. Further, Section 3 unfolds the constituent elements of the case study design, data collection, and data analysis. Section 4 reports the results with regards to quality attributes in general, followed by Section 5 that presents the findings corresponding to maintainability practices. Section 6 discusses the implications for practitioners and researchers. The threats to validity of this study and how they were mitigated are presented in Section 7. Section 8 wraps up this work and prospects for possible future directions.

2. Background and related work

This section outlines the concepts related to software quality attributes and emphasizes the related work approaches concerned with the management of quality attributes in embedded systems-based projects.

2.1. Terminology and concepts

Given the size and complexity of modern software applications, combined with high user expectations, software quality has become increasingly important. Software quality models have been continuously proposed since the first one introduced in 1977 by McCall et al. (1977) and have been standardized starting in 2001. The quality model in place (ISO/IEC 25010, 2011), introduced in 2011 comprises eight quality characteristics, referred also as quality factors or attributes, each in turn compounded by a set of sub-characteristics: functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, and portability.

The diversity of application types, such as web, mobile, and embedded, has a direct impact on the software quality domain. Quality attributes are prioritized to match client expectations, domain specificity, or standards imposed by the industry. Our study will focus on a subset of quality attributes that are declared by research studies as being essential in embedded systems (Sas and Avgeriou, 2020; Wahler et al., 2017; Ampatzoglou et al., 2016). This set includes maintainability, security, safety, energy efficiency, performance, and reliability.

We employ maintainability, security, reliability, and performance definitions provided by ISO/IEC 25010 (2011). *Maintainability* measures how effective and efficient any change of the system is, from all four perspectives: adaptive, perfective, preventive, and corrective. *Security* characterizes the ability of the system to protect from unauthorized access to information and data and to assure appropriate access depending on authorization rights. *Reliability* measures the ability of the system to perform its intended functions under required conditions (not to deviate from behavior). *Performance* together with efficiency depends on time, resources, and capacity.

Energy efficiency has become an important aspect to be addressed by mobile, embedded and time-critical systems, as it is concerned with battery and other resources usage (Manotas et al., 2016). *Safety* is defined according to IEC 61508 (2010) as "freedom from unacceptable risk of physical injury".

Several good practices have been proposed in software engineering, and studies have reported their beneficial impact on maintainability (Fowler, 1999; Lafi et al., 2019; Roehm et al., 2019), reducing technical debt (Pérez et al., 2020; Yli-Huumo et al., 2016), and improvement of the overall quality of software systems (Marinescu, 2005; Mcintosh et al., 2016; Martin, 2008; Boehm et al., 1976). Most applied practices include: coding rules, conventions, documentation, and refactoring. We started from this set of good practices for general purposes and investigated which of them are suitable for embedded systems. Throughout our study, we have considered the following explanations for these practices.

Coding rules represents a set of accepted guidelines that a team should use to ensure safety, reliability, and security, and in many cases in embedded systems should be compliant with a specific standard.

Coding conventions refers to recommended programming styles to enhance the readability and understandability of the code, and usually includes indentations, naming conventions, programming practices, and principles.

Code review represents a systematic parsing of a programmer's code to check for mistakes and defects, which help in early issue detection; it can be performed manually (by a reviewer), automatically (by a tool), or semi-automatically (reviewer assisted by the tool).

Documentation comprises textual and graphical resources accompanying the source code with the purpose of providing explanations.

Refactoring, as defined by Fowler (1999), represents "a change made to the internal structure of the software to make it easier to understand and cheaper to modify without changing its observable behavior", with the main benefits of improving the design and making the code more maintainable.

Table 1
Steps in case study design and execution.
Source: Adapted from Eisenhardt (1989).

Step	Activity	Section
Getting started	Definition of purpose and research question	Section 3.1
Selecting Cases	Specified participants	Section 3.2.1
Instruments and protocols	Multiple data collection methods	Sections 3.2.2 and 3.2.3
Analyzing Data	Case analysis	Section 3.3
Shaping Hypotheses	Interpretation of results, search evidences	Sections 4 and 5
Enfolding Literature	Comparison with similar literature	Section 2.2
Reaching closure	Theoretical saturation, conclusions	Sections 4, 5 and 8

2.2. Related work

Various studies investigated the management of quality attributes in embedded systems, focusing on specific ones or exploring the trade-off between several quality attributes.

Ampatzoglou et al. (2016) discussed the technical debt in the embedded systems domain and how practitioners perceive it. Their results revealed that maintainability becomes critical in embedded systems whose lifetime is expected to last over ten years, and that run time quality factors receive a higher priority than design time factors in the context of technical debt.

Sas and Avgeriou (2020) investigated the trade-off between quality attributes in the context of the embedded systems industry. Two qualitative data collection steps were used, i.e., interviews and a focus group, with six embedded systems-oriented companies, involving twenty participants. The results of the study showed that the interviewed subjects preferred run-time over design-time quality and that the trade-offs were often implicit. *Our approach is similar to this study, bringing extra empirical evidence and emphasis on maintainability and on the practices that maintainability is associated with, i.e., coding rules, conventions, documentation, code review, and refactoring.*

The inherent trade-off between design and run-time qualities poses challenges in embedded software development. The paper (Papadopoulos et al., 2018) came up with an investigation of the impact of transformations for improving performance and energy efficiency on software quality metrics. They study the impact of refactorings for increasing the design time quality on the execution time, memory and energy consumption. The results obtained reveal interrelations and trade-offs between the aforementioned metrics.

Redin et al. (2008) investigated the relationship between traditional (classical) software quality metrics and the relevant physical metrics for embedded systems. Their study reveals that different design decisions influence these metrics and they try to find out which software quality metrics are relevant for embedded software design. Finally, they propose to use the knowledge about the relationship between quality and physical metrics to suggest improvements in the modeling.

The quality of large-scale mission-critical software systems has an impact on production and maintenance costs. SVEVIA (Carrozza et al., 2018) is such a framework for software quality assessment and decision support, being validated in a three years long industry-academy cooperation, identifying the key challenges toward a satisfying quality-cost-time trade-off. New methods for product/process quality assessment, prediction, planning, and optimization are defined.

In order to determine whether the QA's defined for software systems, in general, are adequate when working with embedded systems, research based on trade studies was evaluated. Sherman (2008) conducted a study in which they conclude that: many of the embedded system quality attributes maps directly to existing software quality attributes, and some attributes such as portability take on a modified definition.

Arvanitou et al. (2017) conducted a mapping study in order to investigate the quality attributes and the associated metrics. The results revealed that low-level quality attributes are more frequently studied than high-level ones (i.e., cohesion and coupling than maintainability and reusability), the most frequently investigated high-level quality attribute is maintainability, a single metric is used to assess the quality attribute, and empirical validations are mostly used for metrics.

In relation to existing approaches, we also aim to understand quality attributes in embedded systems, focusing more on specific quality attributes, i.e. maintainability. Our study differs from existing similar approaches from at least the following aspects: (i) in-depth investigation on practices and approaches to achieve maintainable projects based on embedded systems, and (ii) an inspection between how these are formulated at the level of management and tracked at the level of development.

3. Overview and design of the study

This section outlines the study design (research objectives with research questions), information about data collection (interviews and survey) and data analysis. The study was designed and executed based on the format from Eisenhardt (1989), and the overall plan can be followed in Table 1, together with the section of the paper describing each step.

3.1. Purpose of the study

The methodology adopted for this work is based on Goal-Question-Metrics template (Basili et al., 1994), as follows:

Analyze *software quality related activities* in software teams ***for the purpose*** *of understanding quality related practices* ***from the point of view*** *of managers and software developers* ***in the context of*** *embedded software projects.*

In particular, we want to explore general opinions, practices, and tools used by practitioners regarding quality attributes, and then emphasized these practices for controlling and assuring the maintainability of embedded software systems. We designed the study from abstract to concrete, namely, it contains one objective which is *general*, offering an exploration at a higher perspective of quality factors, moving then to a more *specific* perspective of practices associated to maintainability. This choice is justified by the following two arguments: the higher perspective might help practitioners understand why the concrete practices are needed, in which context, and secondly, there is a divergent opinion about concepts and practices in the industry, and the results of this study are expected to contribute to their understanding.

This general purpose has been formulated in a hierarchical manner, dividing it into two main research objectives, named Objective 1 and Objective 2, further divided into three research questions each:

Objective 1*: Investigation on quality attributes in embedded systems:* Our study inquires which quality attributes are prioritized and how they are observed and controlled during the development processes, through the following questions:

RQ1.1*: Which software quality attributes are considered to be most important in embedded projects?* Software quality models

Table 2
Project details and background information from interview data.

Id	Domain	Platform	Role	Experience	
				Curr. role	Total
I1	Railway	Python, Ruby, Perl; several SO	Embedded Lead Manager	4	15
I2	Automotive	Embedded C (70%), C++	Delivery Owner & PM	1.5	5
I3	Light electric vehicles	RTOS, Python & C++	Scrum Master	0.5	12
I4	Marine	C, C++, Python, Windows, Linux	SE team leader	4	16.5
I5	Automotive	Embedded C	Software Architect	3	20
I6	Manufacturing	C++, C, Python, C#, Windows, Linux	Managing Director	3	12
I7	Automotive	ARM, assembler, C, C++, Java	Business Unit Manager	3	25
I8	Electronic solutions for Energy	ARM, assembler, C, C++, Java, C#, LabVIEW, Keil, RTOS	Embedded System Manager	3	19
I9	Testing tools for electronic boards	Fujitsu, CodeBlocks, C	Project Leader	4.5	5
I10	Automotive	C, Python, C#	Department Leader	2.3	8
I11	Health	Android, C++, Kotlin	Scrum master	2.5	3

have been designed for general-purpose software systems. Depending on the specificity of the application, some attributes may be more important than others and may be prioritized when considering the trade-off between them. Embedded systems are of particular interest for quality attributes, as they encapsulate software and hardware parts in a product and in many cases are part of real-time or safety-critical systems.

RQ1.2*: How are the essential quality attributes ensured in embedded projects?* We inquire which are the different strategies designed for different phases of project development to address quality attributes.

RQ1.3: *Is there any gap between managers and developers regarding how important software quality attributes in embedded projects are perceived?* The relation between people and existing methodologies in software development captured the interest of the research communities from '90 (Cockburn, 1996, 2003), given the differences that might exist between different roles and expertise in these complex processes. We want to understand how important software quality attributes in embedded systems are perceived at management and development levels.

Objective 2*: Exploration of aspects related to maintainability in embedded systems:* Considering the essential role of maintainability, we want to obtain a deeper understanding of practices, activities, and tools to keep it under control, by formulating the following questions:

RQ2.1*: Which are the specific practices associated to the maintainability of embedded systems?* Maintainability, defined as the easiness to improve, correct, or adapt a software system, is one of the most important quality characteristics. We are interested in identifying which practices are taken during development in order to facilitate maintenance and how they are implemented by the development teams.

RQ2.2: *Which are the activities and tools used for controlling maintainability?* This question investigates if the processes are controlled and which tools may be used to support them. We want to understand how much the practitioners adopted tools to assist them in improving the maintainability of embedded systems.

RQ2.3: *Is there any gap between managers and developers regarding activities related to the maintainability of embedded projects?* Considering the different perspectives of decision-making and actual implementation, we explore if there exist differences among them.

The design of the study can be followed in Fig. 1, where the main goal was divided in two research objectives, further divided in three research questions each, for which we used different research means (interviews and survey) in order to provide answers (metrics) to the stated research questions.

Therefore, we aim to achieve the necessary guidance to practitioners and researchers for answering the aforementioned questions. Thus, the outcome of this study provides the following contributions:

- Outline the quality attributes that are prioritized in embedded systems projects.
- Outline how the quality attributes are observed and controlled during the development process.
- Highlight the specific practices in embedded systems projects related to maintainability.
- Layout the tools adopted by practitioners to help them in improving maintainability.
- Identification of gaps between managers and developers regarding the importance of quality attributes and activities related to maintainability.

3.2. Data collection

To support our study, we conducted interviews with people involved in the management and addressed a survey to developers dealing with embedded projects. Our analysis method consists of qualitative investigation using thematic analysis in interviews, and quantitative interpretation of survey results.

3.2.1. Cases and subjects

In the first phase of the study we selected a number of companies whose main activity is in the domain of embedded system development. Each company nominated a person with a leadership role within an embedded system project. We considered them as best fitted due to their experience and management of activities and processes related to embedded systems.

Application domains and platforms used in the projects, together with background details about interviewed practitioners are summarized in Table 2. The applications domain spans from Railway, Automotive, Light Electric Vehicle to Electronic solutions for Energy and Health, while C and C++ identify themselves as the most used languages for embedded systems, followed by Python, Java, respective others specifics platforms for embedded systems. The interviewees held leading positions, with an overall experience between 3 to over 20 years, and experience in embedded projects ranging from half a year to four and a half years. Eleven interviews were performed through online meetings for approximately sixty minutes each.

The second phase of the study involved developers, and engineers working in embedded system projects. After the interview, the managers were asked to distribute the anonymous survey within their teams. The number of respondents was 25, and the average duration of the survey was 10 min. Survey questions that targeted the working experience information revealed that some of the respondents had first experience in non-embedded systems projects and then switched to embedded system projects, whereas others started directly with embedded projects. The average overall experience of the respondents is 10.6 years and the average experience in embedded projects is 7.78 years, with at least 1 year experience and a maximum of 20 years experience in embedded systems projects.

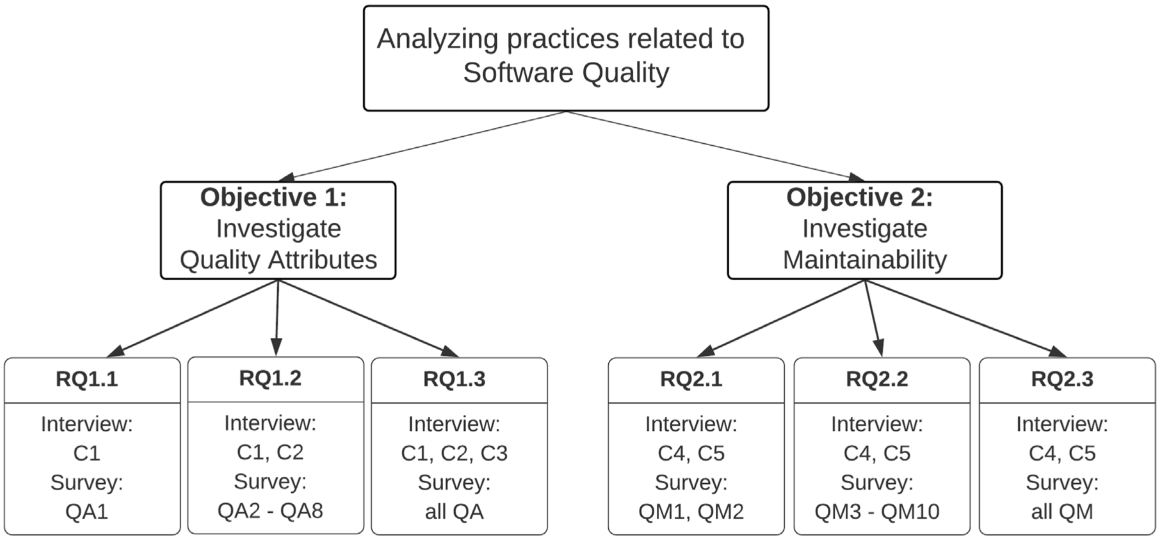
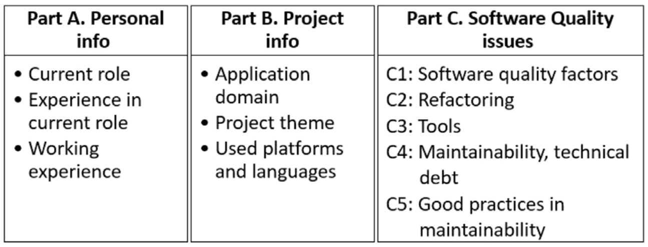


Fig. 1. Study design - A GQM approach.



Part A. Personal info	Part B. Project info	Part C. Software Quality issues
• Current role • Experience in current role • Working experience	• Application domain • Project theme • Used platforms and languages	C1: Software quality factors C2: Refactoring C3: Tools C4: Maintainability, technical debt C5: Good practices in maintainability

Fig. 2. The format of the interview.

3.2.2. *Interviews*

Interviews were designed following a predefined format, containing a set of open questions, with the possibility for the interviewers to further investigate interesting answers, and for the interviewee to freely elaborate on them.

Each interview was carried out by two interviewees and was preceded by a *Consent letter* through which the interviewees gave their explicit permission to be recorded during the interview and were informed about the purpose of the study and the procedures regarding the confidentiality of the collected data that will be followed by the whole research team. A third member of the team had the role to perform the transcript which was then agreed about all authors. This transcript was sent for review to the interviewee to be approved to avoid misunderstandings. After approval, the transcript was anonymized.

The interviews spanned three parts: the first part corresponding to the introduction gathers data about the participants, their role, and experience; the second part investigated projects information and the third part was dedicated to issues related to quality attributes in general, followed by maintainability in particular. A summary of the interview format can be seen in Fig. 2. The list of quality factors taken into consideration was constructed starting from ISO25010 (ISO/IEC 25010, 2011) and then adapted based on related work (Sas and Avgeriou, 2020; Arvanitou et al., 2017) . Regarding practices, we started from the most used ones in general. Due to the specificity of embedded software systems, including low code programming, some well established practices may not be applicable in the case on embedded systems and the last part of the interview was inquiring about the used ones.

The transcript template and the consent letter can be found in our replication package available at Vescan et al. (2023).

3.2.3. *Survey*

We constructed an exploratory survey with the purpose of obtaining developers insights for further investigation correlation with management insights of the interviews. As a consequence, the design of the survey follows closely the content of the interview.

The survey design matches the research aims (i.e., questions are mapped to research objectives) and the target population (wording and format of the questions).

The survey has a variable number of questions, depending on the answers to some of them (see Table 3), organized in three parts, referring to general quality attributes investigation (Questions QA1-QA8), to maintainability in particular (Questions QM1-QM10), and two more general questions (Questions QG1-QG2) asking their current role, experience and gender.

The questions, respectively the options for multiple choice responses were first constructed based on the data collected during interviews, and then they were validated in two ways: review from an external experienced researcher and feedback from completion of the questionnaire by a developer.

Remark. We have added an extra element to the usual Likert scale, i.e., "Extremely important" since during the analysis of interviews we have noticed that the respondents considered some of the attributes as being more than very important when it has to conform to standards.

The survey questions are also included in the replication package available at Vescan et al. (2023).

3.3. *Data analysis*

Given the type of collected data, we decided to combine quantitative with qualitative analysis. We used thematic analysis (Braun et al., 2019) as method recommended (Kiger and Varpio, 2020) for analysis of free text collected from interviews, surveys or other type of non-structured text, and which has been previously applied in Software Engineering research (Cruzes and Dyba, 2011; Gregory et al., 2015). Since we started from an existing list of codes, the deductive approach was chosen, and we applied the reflexive method (Kiger and Varpio, 2020) in order to reflect and document choices and practices. The applied steps (indexed) are described below:

After the interview transcripts have been approved, for questions C1, C2 in Part C (see Fig. 2), we started with step (1)

Table 3
Survey questions.

No	Question	Question type
QA1	Consider the following attributes, how important they are in the project? Security Safety Energy Efficiency Performance	Likert scale
QA2	How is security ensured?	Multiple choice
QA3	How is safety ensured?	Multiple choice
QA4	How is energy efficiency ensured?	Multiple choice
QA5	How is performance ensured?	Multiple choice
QA6	Do you use a static analysis tool?	Yes/No
QA7	If yes, which is the purpose?	Open text
QA8	If yes, which tool (if it is a client tool or confidential, please just mention "client" in your answer)	Open text
QM1	Are you concerned about the effort to maintain the system?	Yes/No
QM2	If yes, how do you refer to it?	Open text
QM3	Consider coding rules associated with your project: Who is responsible for formulating the coding rules? (Please select the item closest to your situation)	Likert scale
QM4	Consider coding rules associated to your project: How are the coding rules followed? (Please select the item closest to your situation)	Likert scale
QM5	If you use a tool associated with coding rules, please specify which tool (if it is client tool or confidential, please just mention "client" in your answer	Open text
QM6	Can design patterns be applied in your project?	Yes/No
QM7	If yes, how important do you consider them?	Open text
QM8	Can refactoring be applied in your project?	Yes/No
QM9	If yes, how important do you consider it?	Multiple choice
QM10	How code review is performed? Please select the item closest to your situation	Multiple choice
QG1	Role and experience	Open text
QG2	Gender	Multiple choice

Table 4
Themes, terms and examples for "Refactoring".

Theme	Terms	Examples
Done	important, very rare	*"very important", "avoid through best practices"*
Responsibility	management, development, architecture	*"part of the team together with architect", "management team is involved"*
Reason	efficiency, bugs, hardware change	*"if the hardware is change and bug exists", "long build and lots of bugs"*

extracting the data, by examining the entire transcripts and dividing them into sections (usually associated with responses to questions). We then performed (2) coding, in the following way: two persons were responsible for data coding which was then checked and classified by the third independent coder, by viewing the interview recording (verifying the initial data). (3) Classifying keywords in themes was again done in two steps, two persons doing the initial conceptualization, while the third verified the process against the initial transcript. We employed a "dictionary" in order to record the terms and concepts, to be able to constantly compare them with new terms emerged from interview processing and to store them in order to determine their associated dependencies and to promote their subsuming themes. The results were stored in a table, providing themes, a short definition, terms, and suggestive examples from the interview texts. Table 4 shows the information stored for the theme "Refactoring", defined in brief as "restructing code for improvement, without changing external behavior".

In order to (4) create a model of higher-order themes, we proceeded by comparing differences and similarities between sections of a single interview or between distinct interviews. The process was carried on as successive discussions between all three authors, by bringing interview excerpts as arguments. Throughout the model the identified dependency was *ensured by*. The resulting model is depicted in Fig. 3. The final step of (5) evaluating the trustworthiness of the approach consisted of debate between all authors, addressing clear explanations, evidence based coding and classification and consistency of the approach. In the end, the interviewers were asked to validate the results.

The same qualitative research method of thematic analysis was applied to responses to questions C4 of Part C of the interview (see Fig. 2). The same steps were applied, and the resulting model of higher-order themes is shown in Fig. 4 reflecting also identified dependencies: *is a*, *supported by*, *formulated by*, respectively *traced by*.

For the rest of the responses of the interviews and the survey, we have considered quantitative indicators, that are described when interpreting the results.

The following two sections provide responses to the research questions formulated in Section 3.

4. Investigation of quality attributes importance in embedded systems

This section contains the *results* and insights regarding the investigation that we performed on the quality attributes in embedded systems, i.e., ***Objective 1****: Investigation on quality attributes in embedded systems.*.

4.1. Results for research questions related to Objective 1

In what follows we outline the results for each research question related to the Objective 1 of the study.

RQ1.1*: Which software quality attributes are considered to be most important in embedded projects?*

In order to answer to this question, we will refer both to the interview and survey answers, thus discussing both perspectives, that of managers and developers.

Fig. 5 contains the values for both interviews and survey from *Not at all important* to *Extremely important*. *Safety* and *Security* were the most important quality attributes (excluding maintainability) considered for both interviewers and the respondents of the survey. *Energy efficiency* was considered very important by the survey respondents, however the managers gave less importance to it, as seen in Fig. 5.

Performance was mentioned as being important in several interviews, so we decided to add it as an investigated quality

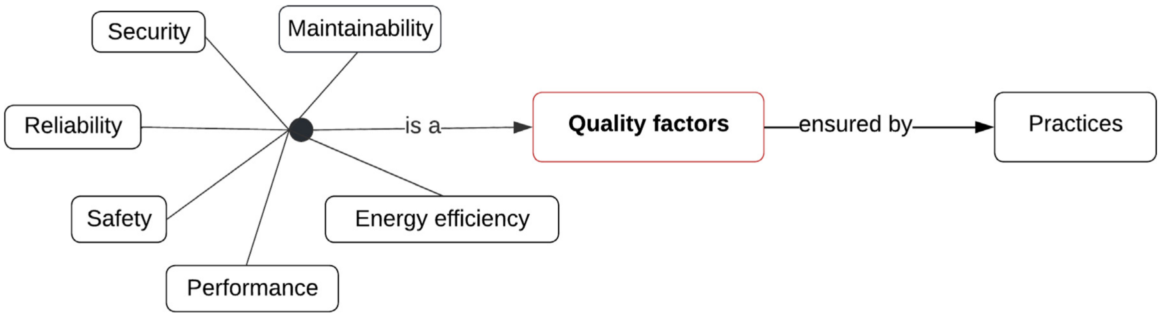


Fig. 3. Concepts network corresponding to quality factors.

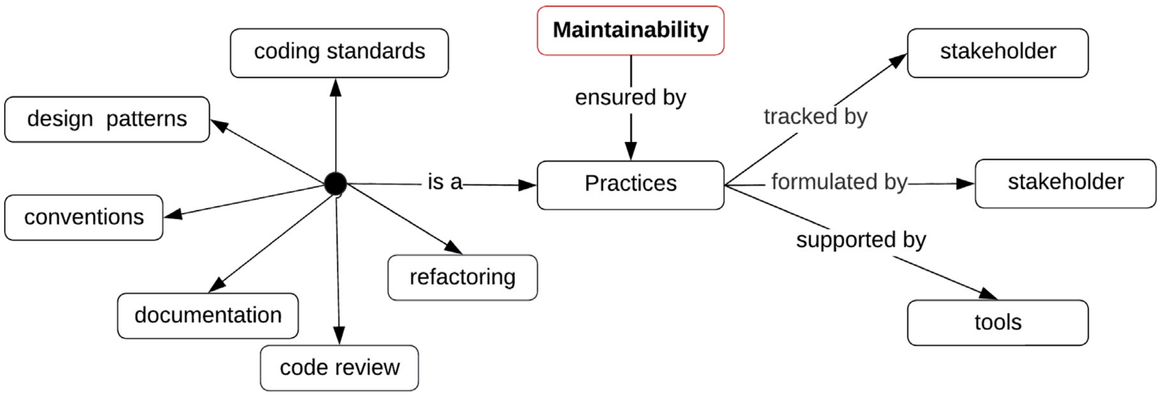


Fig. 4. Concepts network corresponding to maintainability.

attribute in the survey. *Reliability* quality attribute was considered only in the interviews questions and for some projects was not an important attribute, however 4 interviewers considered *very* and *extremely* important (As the majority of interviewed (64%) considered it as *Not at all important*, we decided to remove it from the survey).

When compared our study results with the results obtained by Sas and Avgeriou (2020) regarding other quality attributes that are considered or not important in embedded systems, we note that in the previous study *Safety* was not a major concern and *Security* had a secondary importance, whereas our study revealed that they are considered *very important* to *extremely important* by both interviewers and survey respondents. Regarding *Performance*, the Sas and Avgeriou (2020) study outline that it is not of high priority, depending mostly on the projects needs, whereas in our study having a *very* to *extremely* importance. Thus, we may conclude that *the same quality attributes were considered in both studies, but the results outline that with different priority*.

Our results considered maintainability important as 9 out of 11 respondents mentioned so, with the remark that they refer to it differently: *Maintainability* or *Technical Debt*. This result was later confirmed by the survey, as 23 out of 25 responses acknowledged the effort dedicated to maintainability. For one of the projects, maintainability was very important since the product had to work for 20 years, thus prevention was very important, along with technical debt that required methodology and regulation being put in place. Another important aspect mentioned in another project was that of "exchangeable" code, choosing clear code first over performance code, thus exchangeability being an important aspect to be considered. Even more, two of interviewers responded being concerned with *preventive maintenance*, two with *corrective maintenance*, and one with *perfective maintenance*.

When comparing our study results with the results obtained by Sas and Avgeriou (2020) we notice a similarity: in the previous study *Maintainability* was a crucial aspect in all projects, the same in our study, i.e., all the interviewers mention it as being investigated in their projects, however, the Sas and Avgeriou (2020) study concluded that even if *Maintainability* was "deemed very important, it was down-prioritized in practice". Our results gave a greater importance to maintainability.

RQ1.2*: How are the essential quality attributes ensured in embedded projects?*

We have addressed questions regarding means of assessing the considered quality attributes (*Security*, *Safety*, *Energy efficiency*, and *Performance*). This is one example of different interpretations regarding how the quality aspects procedures are followed. In some cases, respondents refer to software development processes (last three line of Fig. 6), while in others to responsibility of the process (first three line of Fig. 6).

As shown in Fig. 6, the most used method is to check these quality factors during testing (28 cases), followed closely (26 cases) by implemented them as non-functional requirements. However, the application domain is having an impact on these methods, as some of them as subject to external audit or client processes, or imposed by standards. *Security* and *Safety* are the attributes that are prioritized, as 24 out the 25 respondents mentioned aspects regarding how they are ensured.

RQ1.3: *Is there any gap between managers and developers regarding how important software quality attributes in embedded projects are perceived?*

Our data collection approach targeted two distinct groups of individuals with distinct roles involved in the development of embedded systems, i.e., managers and developers. We investigate if the strategic planning of management and their perspective on software quality attributes is in accordance with the actual implementation of them by the developers.

At a large scale, the interview and survey questions target the same quality aspects of embedded projects. The differences appeared only when we were seeking more detailed information while performing thematic analysis, as explained in Section 3.3.

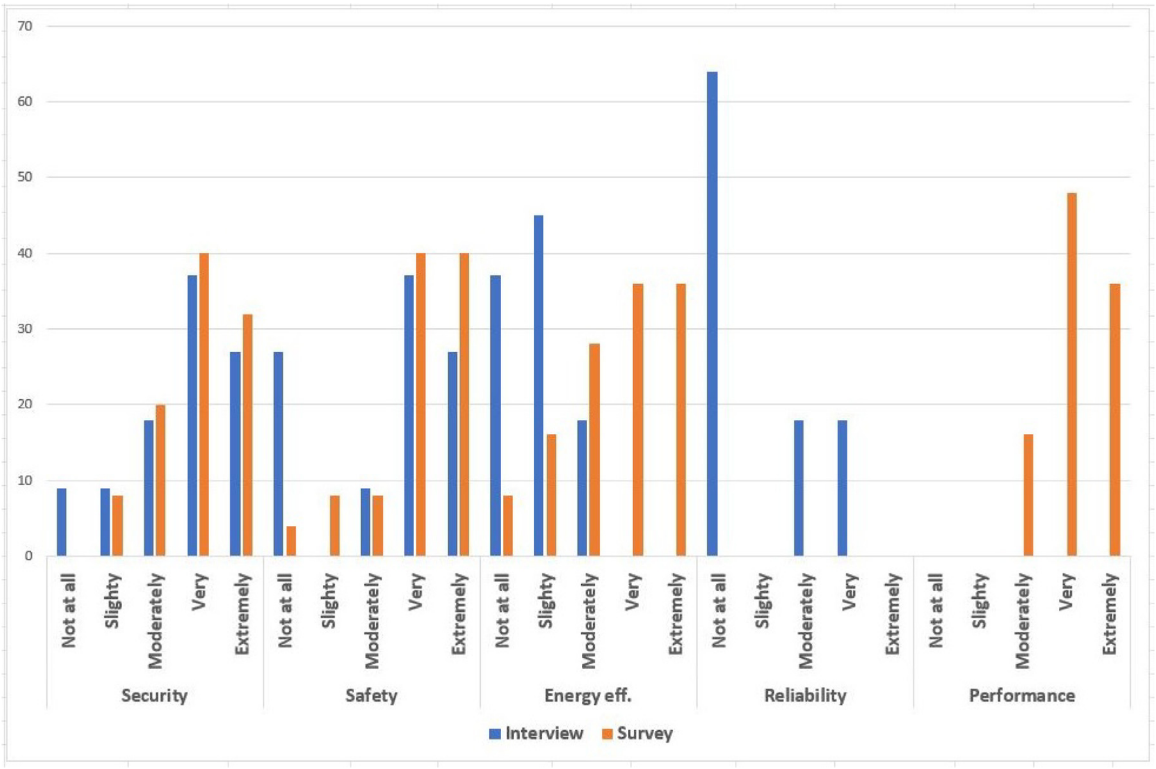


Fig. 5. Quality attributes interview and survey responses expressed in percentages.

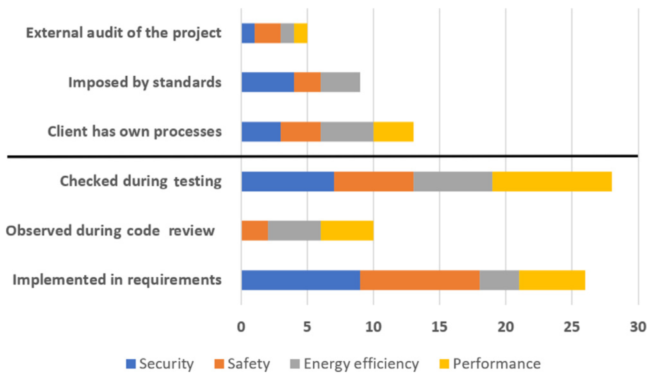


Fig. 6. Survey results regarding how quality attributes are ensured.

Our analysis has identified an unified perspective between management and development teams, with a few exceptions that we are going to reflect upon in the following.

Security and *Safety* are seen as *Very important* to *Extremely important* by both managers and developers as data provided in Fig. 5, i.e., similar values (37%–40% and 27%–32%) being reported from interviews and survey. However, regarding the *Energy efficiency* the managers and the developers have different opinions: managers expressed it having *Slightly important* to *Not at all important* (37%–45%), whereas the developers considered it *Very important* to *Extremely important* (36%). Regarding *Reliability*, in the interviews with the managers, this quality attribute was perceived as *Not at all important* for majority of managers (64%) and we decided to excluded from the survey, and in the same time another quality attribute, namely *Performance*, was mentioned. As a consequence, we incorporated this new identified quality attribute in the survey and we found that this was also considered *Very important* to *Extremely important* by majority of respondents (developers, 48%–36%). In a nutshell, we can conclude that developers assume higher responsibilities than expected when quality attributes are involve and are continuously preoccupied by them.

4.2. Lesson learned for Objective 1: Quality attributes in embedded systems

- The importance and priority given to quality attributes in embedded systems changed over the last years, with higher attention paid to *Safety* and *Security*; even if for real-time or safety critical systems this is universally accepted and imposed by standards, we noticed that also other types of embedded systems are preoccupied by these factors; *Safety* and *Security* quality attributes were investigated in other studies (Laprie, 1992; Arvanitou et al., 2017), however, found to be at low importance back then. We acknowledged this difference in our study, which could also reflect higher quality expectations nowadays.
  We outline in our study (using 11 projects from various embedded systems domains) the findings regarding *Safety* and *Security* and argue that the importance of these attributes may have changed. More recent studies are needed to investigate these attributes.
- The methods to ensure quality attributes are very diverse, and this is a point that we identify as needing more investigation and a unified proposal of research and professional communities for good practices. The diversity sometimes comes from different understandings of standards, documentation and lack of a set of acceptable good practices. Also, diversity means in some cases that only some activities are used.
- There is a consensus between managers and developers regarding quality attributes and how they should be ensured in embedded systems. The practices and activities are defined at all level, and quality is receiving constant interest throughout the development phases, with several application domain regulated by certified standards.

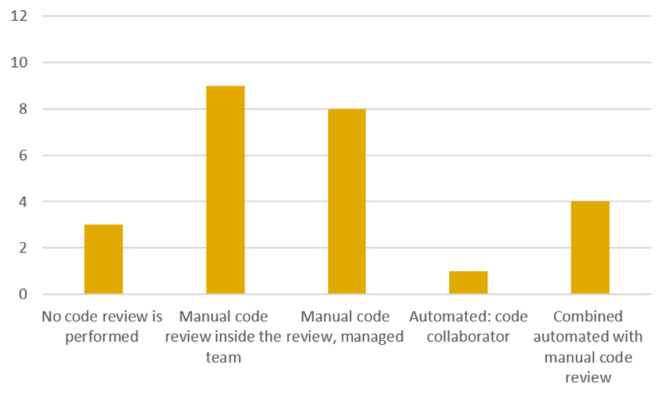


Fig. 7. Survey results regarding code review.

5. In depth investigation of maintainability

This section summarizes the findings associated with ***Objective 2:*** *Exploration of aspects related to maintainability in embedded systems*, formulated in terms of answers to RQ2.1 and RQ2.3 research question formulated in Section 3.1.

5.1. Results for research questions related to Objective 2

The importance corresponding to maintainability in our study is confirmed by the responses to the survey questions, when referred to the "effort for maintaining the system" all 25 respondents answered that they are concerned about this in their project, however 72% named it *Maintainability* and the other 28% named it *Technical Debt*.

RQ2.1*: Which are the specific practices associated to maintainability of embedded systems?*

We started the discussion in the interviews with the most known good practices (Fowler, 1999; Martin, 2008) for having maintainable software, namely: coding rules, conventions, documentation, and design patterns. In case of the first three practices: **coding rules, conventions and documentation**, the overall opinion of the interviewers was on their clear importance in embedded projects. Some of the practices are defined at the company level and respected in all projects, and in some cases they are strengthened by standards that must be followed (in the application domains such as automotive, railway, and health).

Code review receives significant attention both at the management level and by the developers. The interviews with the managers revealed how they perceive the responsibility of controlling this practice: seven of them mentioned dedicated tools such as Git and Jira, two declared that they are using spreadsheets and scripts, one mentioned that the process is controlled by the client, one did not consider code review as a distinct process to be mentioned.

From the developers' perspective, code review is performed by almost all of them (with three exceptions), as shown in Fig. 7. Manual code review either by a senior member of the team (9 cases) or managed by a project management or version control tool (8 cases) are the most used methodologies. Some companies use code review tools either automatically (1 case) or combined with manual code review (4 cases). The situations that trigger code review are diverse, and according to the developers they include: *"code coverage, detect security defects in code, detect undefined behavior and coding constructs, check DRY principle, find potential issues before they happen, eliminate program errors which may cause issues, check coding rules (MISRA)"*.

Regarding the application of **design patterns**, the interview responses revealed a strong dependence on the used programming languages, as some low level languages or C are not suitable for this. The survey responses point out that the majority of developers (20, representing 80%) use design patterns in their code, even if they do not consider it very important, as Fig. 9 shows.

A special focus in our study was dedicated to **refactoring** as it is considered the most important practice in reducing maintenance costs (Fowler, 1999). Most of the 11 interviewers stated its importance (3 considered it "very important", and 5 as "important"), while two managers said that refactoring "*is usually avoided through good practices*" as it requires client approval and implies supplementary costs.

The responsibility of decisions regarding refactoring is diverse: at the level of development, code review, architecture, or even management. In some cases, the interview respondents named which are the driving forces of deciding refactoring such as: *"exceed number of bugs, long build, improve efficiency and robustness"*.

Developers responses suggest a significant attention dedicated to refactoring as 10 consider it *extremely important* and 9 as *moderately important*, as reflected by Fig. 10. A special notice should be made to the importance the developers assign to refactoring compared to design patterns (see Figs. 9 and 10), as refactoring seems more relevant than design patterns.

At the end of this interview section, we asked the managers if they consider any other aspect as being important to be mentioned as a good practice. Four respondents considered that specifying requirements related to these practices is an important approach by which they can assure that the whole team will respect and apply these practices. In two cases, the V-model is applied in development, such that each development phase has a corresponding validation phase.

As a final remark, code review and refactoring are the most used good practices, as design patterns are not always applicable, and other practices such as coding rules, conventions, and documentation are mostly controlled by standards or company-level practices. Fig. 8 shows the rate of applying these practices by managers and developers (value zero reflects the fact that particular aspect was not covered by the survey).

RQ2.2: *Which are the activities and tools used for controlling maintainability?*

A part of our study was to investigate how good practices related to maintainability (e.g., coding rules, code review conventions, documentation, and design patterns) are *formulated and followed.* In other words we aim to find out who are responsible for formulating and following them. At the same time, we want to understand how much the practitioners adopted *tools* to assist them to assure the maintainability of the built system. We answer this question considering the interviews and the survey, discussing both the opinions of managers and developers.

The results of the interviews regarding the responsibility of *formulating* best practices to ensure maintenance reveal that most of the 11 interviewers stated that they define standards at the company level. Two respondents stated that they follow certified standards that define these good practices (ISO, respectively client's decision).

The survey outlines that processes related to **coding rules** differ at large scale, concluding that they are specific for each project, as shown in Table 6. In the majority of cases, *formulating* the rules is according to standards, followed by project level, namely depending on project specificity and at company level, meaning that all projects follow the same rules. Only in a few cases, these coding rules are defined by the client, and this might be the case of outsourcing projects (line 3 of Table 6).

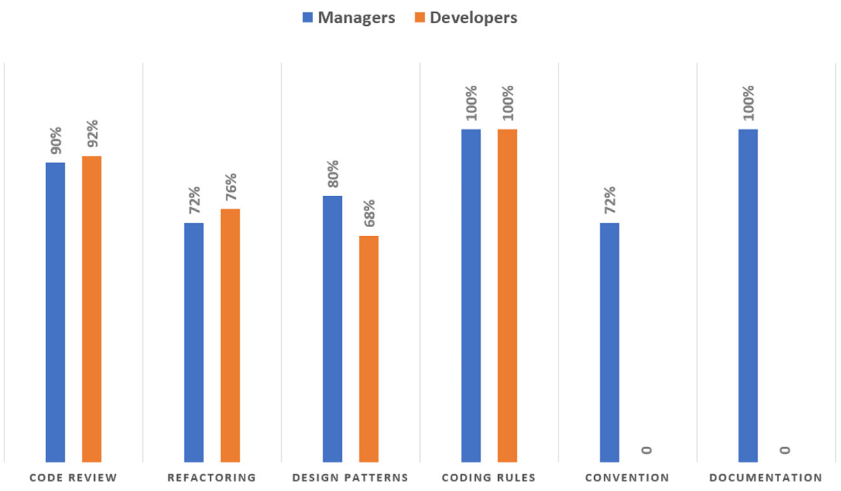


Fig. 8. Maintainability related practices at management and development levels.

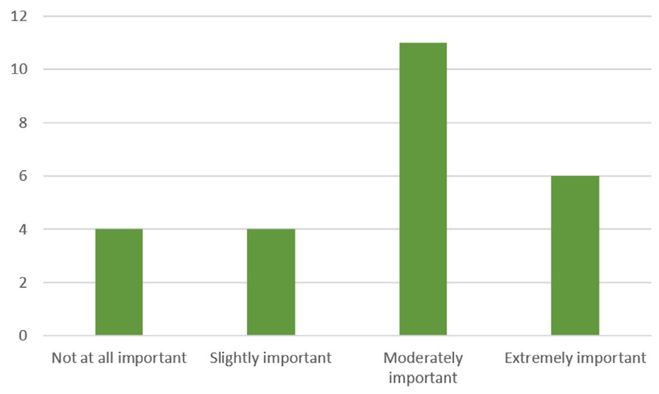


Fig. 9. Importance of design patterns.

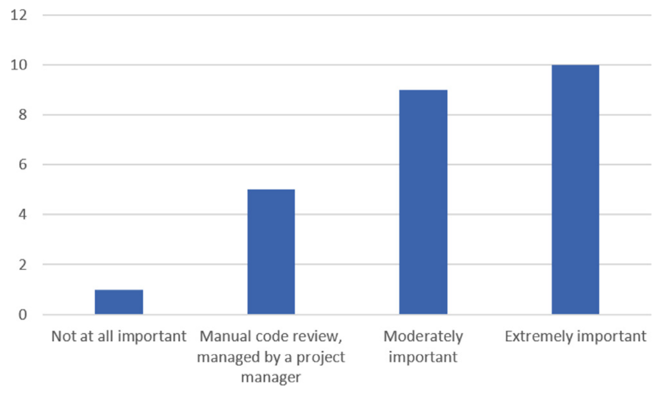


Fig. 10. Importance of refactoring.

Regarding the responsibility of *following* these good practices in the project, the answers are widely spread out. If we refer to the ones received from the interview, out of the 11 answers we have the following: 2 answers concern the development and testing team; 2 answers say that this action is performed by the client; 1 answer says that good practices are performed by a mixed team with the client, and one answer states that it implements an audit process; there are also 2 answers that refer to a code review team in combination with the QA team; 1 answer states that they are tracked by Process Capability Checks (PCC) according ASPICE Implementation, and 1 answer says that experienced developers are responsible for tracking these good practices.

The responsibility of *following* these rules is also scattered between the survey responses, as presented in Table 6, at the level of: the developer, another person, usually a senior member of the team, or the use of a tool. Two important remarks may be drawn from this situation: (i) there is a significant number of cases, 7 cases representing 28%, in which developers do not carry the responsibility of following these rules and (ii) there is a skepticism in adopting tools that might assist this software process (44%).

Three types of tools that assist the maintainability of embedded systems were considered by respondents of the interview: *code review, static analysis and issues trackers tools*. Of the 11 managers interviewed, 4 stated that they use *code review* tools, as described in Table 5.

Regarding the tools used for the *static code analysis* and/or manage activities related to maintainability (allowing them to set different thresholds depending on the importance of the quality attribute), managers' responses are very fuzzy. Most of them actually refer to tools for managing the activity of code review, such as Information Management System tools. The Coverity tool is stated by 3 out of 11 respondents, whereas Lint was stated by 2 out of 11 respondents. An important aspect is that at the management level, they oversee more than one tool for static code analysis, one of them mentioned even 5 tools.

The last type of tools that are used to assist maintainability issues refers to *issue tracker*. The range of this type of tools is wide, and the number of managers who use them is high. The most used issue tracker tool is Jira (8 out of 11 respondents), Confluence and Bitbucket (2 out of 11 respondents) or tools to assist HIL simulation (2 responses). Also, 3 out of 11 interviewees stated that they use more than 3 tools.

Table 5 presents the tools used for controlling maintainability both by managers and developers as they are emphasized by the interview and survey.

As a final remark regarding the actions and tools used for controlling maintainability, we can conclude that good practices are formulated in accordance with predefined standards or defined by the customer or at the company level. Experienced developers together with QA team, code review team, or audit are responsible for tracking these practices, and software tools are used to facilitate all this process, especially for issue tracking and static analysis and less for code review.

RQ2.3: *Is there any gap between managers and developers regarding activities related to maintainability of embedded projects?*

Table 5
Tools used by managers and developers for controlling maintainability.

Issue tracker		Static analysis and code review	
Name	Freq.	Name	Freq.
Jira	8	IMS	3
IBM legacy tools	2	client	2
HIL	2	Coverity	3
Confluence	2	Valgrind	1
Bitbucket	2	Clang tidy	1
GIT	4	SonarQube	1
Jenkins	1	PTC – Integrity	1
PTC – Integrity	1	BlackDuck	1
Trello	1	ProGuard	1
		Lint	2
		Artifactory	1
		Gerritt	2
		Swarm	1

Table 6
Survey responses regarding coding rules.

Formulated	Never	Some times	Usually	Always
Company level	7	6	9	3
Project level	1	9	9	4
Client	11	7	5	1
According to standards	2	4	11	8
Followed by	**Never**	**Some times**	**Usually**	**Always**
Developer	**7**	6	9	3
Other person	1	9	9	4
Tool	**11**	7	5	1

The managers perspectives (collected through interviews) was compared to the developers' view (observed from survey responses). Our analysis identified consistent and uniform approaches in both management and development teams, with small differences that we are going to reflect upon in the following.

Regarding maintainability practices, we have made the following observations:

Coding rules: as pointed out in the Table 6, a significant number of developers are using tools to assure that these rules are respected, a detail that was not mentioned during interviews. In this way, the importance of this practice stated by managers is actually distributed at three different levels: developer him/her-self, another person with the role of reviewer, and tool.

Code review: the actions suggested by managers are consistent with developers practices, as in most cases a manual code review is performed (8 interviews responses out of 11, respectively 17 out of 25 survey responses) and in a small percentage tools are applied (2/11, respectively 5/25), as counted in interviews and surveys (see Fig. 7).

Considering the **design patterns**, there is a slight difference in applying them, as 6 out of 11 managers appreciated them to be of higher importance than the developers acknowledged (see Fig. 9). However, considering the experience of respondents, with an average of approximately 5 years, we are justified to state that this practice is adopted and installed and does not need extra effort in the development process.

In case of **refactoring**, our results revealed a higher importance dedicated to this practice by developers (10 out of 25, as shown in Fig. 10, compared to managers (3 out of 11). This is an encouraging outcome as the developers considered themselves responsible for providing clean code even if not imposed by project management.

5.2. Lesson learned for Objective 2: Aspects related to maintainability of embedded systems

- The most used practices in assuring maintainability are code review and refactoring;
- There exists various approaches in performing maintainability related activities, in the ways they are formulated and followed. We believe more detailed investigation is needed in this regard in order to improve software processes;
- We notice that assisting tools for code review and static analysis are not used at their full potential. One explanation might reside in the specificity of the used programming languages, however things can be improved in this direction. Most teams use tools for managing the practices (issue tracking) and less for actually analyzing the source code;
- Regarding the perspectives of managers and developers, we found that development teams usually assume a higher responsibility to produce maintainable code, and even if not imposed or required, they are continuously concerned with activities related to maintainability. Our observation is that the developers feel responsible for the quality of code they produce.

6. Implication for practitioners and researchers

For the research community, our results bring evidence to the domain of quality attributes in embedded systems and practices associated with them, and also point out open problems that need to be addressed. These open problems are related to the degree of importance for quality attributes, to associated good practices, as well as the need to define and formalize a conceptual framework for the evaluation of quality attributes in embedded systems, an evaluation based on software metrics that quantify these attributes. Such an evaluation framework could guide the experts to automate the evaluation process and to interpret the measurements results. Last but not least, our proposed approach also states the need of more empirical evidence related to this topic.

For the practitioners community, lessons learned can serve as indicators of their approach of quality related aspects in their products and processes and also as a way on how they can improve them. For instance, the proposed study outline the quality attributes that are prioritized in embedded systems projects, how the quality attributes are observed and controlled during the development process, highlighting the specific practices in embedded systems projects related to maintainability. At the same time, it emphasizes the tools adopted to assist in improving maintainability.

The study identifies some gaps between managers and developers regarding both importance of quality attributes and activities related to maintainability.

The expressed interest of our interviewees shows that there is still a need for knowledge transfer from research results to applied practices.

7. Threats to validity

Our reported results are based on an empirical study, thus being subject to certain threats to validity. The study followed the practices recommended for the domain (Runeson and Höst, 2008; Ralph, 2021) by defining the objective, formulating the research questions, and associating metrics, as sketched in Fig. 1. In what follows, we present the major threats to the validity of our research and we also explain our actions to mitigate them.

Internal validity refers to factors that could have influenced the obtained results.

To increase the chance of obtaining an unbiased interview analysis and missing interpretations of the interviews, two authors participated in the interview, recorded, and shared them with the third author for making the transcript, which was then checked for conformity by one of the initial interview participants. Audio and text were available for analysis. The transcript was then sent for approval to the interviewee.

Regarding the application of thematic analysis, we followed the steps defined in Kiger and Varpio (2020) and described in Section 3.3.

External validity concerns the generalization of our findings.

The data collected expresses the subjective opinion of each interviewee. A larger number of participants should be interviewed to express a general view of a broader audience. We consider this study to be an experience report. Therefore, we are focusing on reporting our observations in this scenario, rather than validating any hypothesis. We then present some discussions, suggestions, lessons learned, and insights for future research.

As the number of participants is the biggest threat in our study, we tried to mitigate it by considering the diversity of opinions: interview respondents come from different companies, and the application domains of the projects are diverse. This might generate another threat related to variability due to different application domains. We tried to avoid it by not considering details related to them, but rather general approaches to quality attributes and practices related to it.

Construct validity concerns the relationship between theory and observation.

This study aimed at eliciting the knowledge and experience of the software teams in relation to a specific goal, expressed as research questions. The GQM protocol, as shown in Fig. 1, was carefully designed to ensure that the questions of the interviews and of the survey were in accordance with the goal to analyze the results. To build the findings, we applied thematic analysis, following the steps according to literature (Kiger and Varpio, 2020).

A possible construct validity threat comes from the risk that the interviewees have different fields of specialization and backgrounds, and their experience ranges from business to very experienced system architects. This could hamper a uniform understanding of the high-level concepts covered during the interviews. To overcome this threat, the interviewer performed a brief explanation before using any of those terms. It is ensured in this way the fact that each interviewee follows the same line as the others about the meaning of software quality attributes and associated activities discussed through the interview.

Another construct threat may come from the biased opinions of respondents about maintainability. To mitigate it, both interviews and survey were designed and carried out such that the questions and discussions about quality attributes, in general, came first, and after a pause/different page in the survey, detailed questions about maintainability were addressed.

8. Conclusions and future work

In embedded systems especially, quality plays an important aspect. Strategies and actions for analyzing various software system quality problems are of continuous interest. Investigation of good practices and activities of how to increase quality is needed.

The goal of this research study was, at a general level, to find which quality attributes are prioritized in embedded systems, and, at a specific level, practical given the essential role of maintainability, to identify the good practices and activities associated with it. We have used a qualitative and quantitative approach, gathering information from two data collection sources: interviews with managers and survey for developers.

Our study identified several quality attributes as being important with various prioritization degrees: maintainability, safety, security, performance, and energy efficiency. These results confirm the ones obtained in literature, especially (Sas and Avgeriou, 2020). Even if the run time quality factors seem to be favored over design time ones, our study reveals that maintainability, which is intrinsically related to design, is of major importance in embedded systems.

Well-established practices associated with maintainability are widely used also in embedded projects, with refactoring and code review being the most used ones for reducing maintenance costs. With reference to some gaps between managers and developers, our findings reveal a uniform perspective regarding quality factors, with slightly different emphasizes in some factors importance. Regarding practices related to maintainability in embedded systems, we identified that coding rules are distributed at different levels (developer, reviewer, tool), for code review the actions suggested by managers are consistent with developer practices, and for refactoring a higher importance is dedicated by developers.

We formulated our findings in terms of lessons learned that can be used by practitioners in improving their work, but also by researchers for future investigations.

The results of this study add up to the body of knowledge related to quality attributes relevant for embedded systems and might offer significant information for practitioners, as most of our interviewed specialists expressed their interest in the findings of our study. Lesson learned can serve both the practitioners and the research community.

A last remark should be made related to the sample size of our study. Given the difficulty in finding teams working in embedded systems, our findings should be regarded more as tendencies rather than universally accepted facts.

Due to the diverse restrictions of application domains, the practices and actions for quality assurance in embedded systems are heterogeneous, thus needing more empirical evidence that could lead to a set of good practices. Further continuation of this study will investigate differences and common approaches in different domains.

Our future plan includes extending our study to technical debt in embedded systems, understanding causes and types, comparing estimation methods, and investigating prioritizing strategies.

CRediT authorship contribution statement

Simona Motogna: Conceptualization, Data collection and qualitative analysis, Writing – original draft, Writing – review & editing. **Andreea Vescan:** Conceptualization, Methodology, Data collection, Qualitative and quantitative analysis, Writing – original draft, Writing – review & editing, Funding acquisition, Resources. **Camelia Şerban:** Conceptualization, Methodology, Data collection and qualitative analysis, Writing – original draft, Revision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

Acknowledgments

This research was partially supported by the Babeş-Bolyai University Advanced "Excellence in R&D activity" Fellowships, with the title "QUA-EASY - QUAlity Estimation And prediction of embedded SYStems".

The authors express their gratitude to all companies participating in this study and contributed with valuable information.

Ethical issues

Since our data collection methods involved human participants, we paid special attention to satisfy existing practices in the domain (European Commission, 2023). For the interview, our procedures included a consent letter to acknowledge the purpose and limitation of the content, anonymization of collected data (both personal and company information), consent about the transcript after the interview. Regarding the survey, no personal information was asked or tracked during the survey.

References

Ampatzoglou, A., Ampatzoglou, A., Chatzigeorgiou, A., Avgeriou, P., Abrahamsson, P., Martini, A., Zdun, U., Systa, K., 2016. The perception of technical debt in the embedded systems domain: An industrial case study. In: 2016 IEEE 8th International Workshop on Managing Technical Debt. MTD, pp. 9–16. http://dx.doi.org/10.1109/MTD.2016.8.

Arvanitou, E.M., Ampatzoglou, A., Chatzigeorgiou, A., Galster, M., Avgeriou, P., 2017. A mapping study on design-time quality attributes and metrics. J. Syst. Softw. 127, 52–77. http://dx.doi.org/10.1016/j.jss.2017.01.026.

Basili, V., Caldiera, G., Rombach, H.D., 1994. The goal question metric approach. In: Encyclopedia of Software Engineering.

Boehm, B.W., Brown, J.R., Lipow, M., 1976. Quantitative evaluation of software quality. In: Proceedings of the 2nd International Conference on Software Engineering. ICSE '76, pp. 592–605.

Braun, V., Clarke, V., Hayfield, N., Terry, G., 2019. Thematic analysis. In: Liamputtong, P. (Ed.), Handbook of Research Methods in Health Social Sciences. Springer Singapore, pp. 843–860. http://dx.doi.org/10.1007/978-981-10-5251-4_103.

Carrozza, G., Pietrantuono, R., Russo, S., 2018. A software quality framework for large-scale mission-critical systems engineering. Inf. Softw. Technol. 102, 100–116. http://dx.doi.org/10.1016/j.infsof.2018.05.009.

Cockburn, A., 1996. The interaction of social issues and software architecture. Commun. ACM 39, 40–46. http://dx.doi.org/10.1145/236156.236165.

Cockburn, A., 2003. People and Methodologies in Software Development (Ph.D. thesis). Faculty of Mathematics and Natural Sciences, University of Oslo, Norway.

Cruzes, D.S., Dyba, T., 2011. Recommended steps for thematic synthesis in software engineering. In: 2011 International Symposium on Empirical Software Engineering and Measurement. pp. 275–284. http://dx.doi.org/10.1109/ESEM.2011.36.

Eisenhardt, K.M., 1989. Building theories from case study research. The Academy of Management Review 14, 532–550, URL: http://www.jstor.org/stable/258557.

European Commission, 2023. Ethics - H2020 Online Manual. URL: https://ec.europa.eu/research/participants/docs/h2020-funding-guide/cross-cutting-issues/ethics_en.htm.

Fowler, M., 1999. Refactoring: Improving the Design of Existing Code. Addison-Wesley Longman Publishing Co..

Gregory, P., Barroca, L., Taylor, K., Salah, D., Sharp, H., 2015. Agile challenges in practice: A thematic analysis. In: Lassenius, C., Dingsø yr, T., Paasivaara, M. (Eds.), Agile Processes in Software Engineering and Extreme Programming. Springer, pp. 64–80. http://dx.doi.org/10.1007/978-3-319-18612-2_6.

IEC 61508, 2010. IEC safety and functional safety. URL: https://www.iec.ch/safety.

ISO/IEC 25010, 2011. ISO/IEC 25010:2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.

Kiger, M.E., Varpio, L., 2020. Thematic analysis of qualitative data: AMEE guide no. 131. Medical Teacher 42, 846–854. http://dx.doi.org/10.1080/0142159X.2020.1755030.

Lafi, M., Botros, J.W., Kafaween, H., Al-Dasoqi, A.B., Al-Tamimi, A., 2019. Code smells analysis mechanisms, detection issues, and effect on software maintainability. In: 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology. JEEIT, pp. 663–666. http://dx.doi.org/10.1109/JEEIT.2019.8717457.

Laprie, J.C., 1992. Dependability: Basic Concepts and Terminology. Springer Vienna, Vienna, pp. 3–245.

Manotas, I., Bird, C., Zhang, R., Shepherd, D., Jaspan, C., Sadowski, C., Pollock, L., Clause, J., 2016. An empirical study of practitioners' perspectives on green software engineering. In: Proceedings of the 38th International Conference on Software Engineering. pp. 237–248. http://dx.doi.org/10.1145/2884781.2884810.

Marinescu, R., 2005. Measurement and quality in object-oriented design. In: 21st IEEE International Conference on Software Maintenance, ICSM'05. pp. 701–704. http://dx.doi.org/10.1109/ICSM.2005.63.

Martin, R.C., 2008. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.

McCall, J., Richards, P., Walters, G., 1977. Factors in software quality. Nat. Tech. Inform. Serv. 1.

Mcintosh, S., Kamei, Y., Adams, B., Hassan, A.E., 2016. An empirical study of the impact of modern code review practices on software quality. Empirical Softw. Engg. 21, 2146–2189. http://dx.doi.org/10.1007/s10664-015-9381-9.

Papadopoulos, L., Marantos, C., Digkas, G., Ampatzoglou, A., Chatzigeorgiou, A., Soudris, D., 2018. Interrelations between software quality metrics, performance and energy consumption in embedded applications. In: Proceedings of the 21st International Workshop on Software and Compilers for Embedded Systems. Association for Computing Machinery, New York, NY, USA, pp. 62–65. http://dx.doi.org/10.1145/3207719.3207736.

Pérez, B., Castellanos, C., Correal, D., Rios, N., Freire, S., Spínola, R., Seaman, C., 2020. What are the practices used by software practitioners on technical debt payment: Results from an international family of surveys. In: Proceedings of the 3rd International Conference on Technical Debt. TechDebt '20, pp. 103–112. http://dx.doi.org/10.1145/3387906.3388632.

Ralph, P.e., 2021. ACM Sigsoft Empirical Standards for Software Engineering Research, version 0.2.0. URL: https://github.com/acmsigsoft/EmpiricalStandards.

Redin, R.M., Oliveira, M.F., Brisolara, L.B., Mattos, J.C., Lamb, L.C., Wagner, F.R., Carro, L., 2008. On the use of software quality metrics to improve physical properties of embedded systems. In: IFIP Working Conference on Distributed and Parallel Embedded Systems. Springer, pp. 101–110. http://dx.doi.org/10.1007/978-0-387-09661-2_10.

Roehm, T., Veihelmann, D., Wagner, S., Juergens, E., 2019. Evaluating maintainability prejudices with a large-scale study of open-source projects. In: Winkler, D., Biffl, S., Bergsmann, J. (Eds.), Software Quality: The Complexity and Challenges of Software Engineering and Software Quality in the Cloud. pp. 151–171.

Runeson, P., Höst, M., 2008. Guidelines for conducting and reporting case study research in software engineering. Empir. Softw. Eng. 14, 131–164. http://dx.doi.org/10.1007/s10664-008-9102-8.

Sas, D., Avgeriou, P., 2020. Quality attribute trade-offs in the embedded systems industry: an exploratory case study. Softw. Qual. J. 28, 504–534. http://dx.doi.org/10.1007/s11219-019-09478-x.

Sherman, T., 2008. Quality attributes for embedded systems. In: Sobh, T. (Ed.), Advances in Computer and Information Sciences and Engineering. Springer Netherlands, Dordrecht, pp. 536–539.

Vescan, A., et al., 2023. Empirical investigation of quality factors in embedded system. dataset. URL: https://doi.org/10.6084/m9.figshare.14833014.v1.

Wahler, M., Eidenbenz, R., Monot, A., Oriol, M., Sivanthi, T., 2017. Quality attribute trade-offs in industrial software systems. In: 2017 IEEE International Conference on Software Architecture Workshops, ICSAW, pp. 251–254. http://dx.doi.org/10.1109/ICSAW.2017.10.

Yli-Huumo, J., Maglyas, A., Smolander, K., 2016. How do software development teams manage technical debt? - An empirical study. J. Syst. Softw. 120, 195–218. http://dx.doi.org/10.1016/j.jss.2016.05.018.

Simona Motogna: is Prof. Ph.D., Habil. at Babeş-Bolyai University and leader of the Software Engineering Research Group. Her interests include software quality aspects, empirical software engineering, and software metrics. She authored several papers on longitudinal studies covering maintainability, technical debt and relation to software development process, and also on Software Engineering education. She serves as reviewer for several SE journals and Program Committee member at diverse conferences. See http://www.cs.ubbcluj.ro/~motogna for details.

Andreea Vescan: associate professor at Babes-Bolyai University, has been involved in higher education teaching and research for more than 18 years. Being an enthusiastic and authentic teacher, she has delivered university courses from programming to software testing, including, since 2013, through an ongoing collaboration with IT companies, courses on computational models for embedded systems. The arrival of her two children between 2010–2013 enriched her vision on research. Andreea is currently principal investigator of an international research project, where she designs, applies, and validates the proposed learning methods. In another research project, she investigates soft computing methods for test case prioritization. See http://www.cs.ubbcluj.ro/~avescan for details.

Camelia Şerban: is lecturer at the Department of Computer Science at Babe[219?]-Bolyai University, Romania. She is an enthusiastic and humorous teacher and facilitator having more than 18 years of experience in higher education. Camelia delivers university courses on Advanced Programming Methods and Data Structures. Her research interests include Software Metrics, Software Assessment, and Object-Oriented Design. She is interested in studying aspects related to software quality. Currently, she is a part of a project regarding the investigation of soft computing methods for software defects prediction. Her potential is the greatest when working in a team. See http://www.cs.ubbcluj.ro/~camelia for details.