Journal of Computer Science

Review Article

Maintenance and Evolution Processes of Embedded Software: A Systematic Literature Review

Aloysio Augusto Rabello de Carvalho and Luiz Eduardo Galvão Martins

Institute of Science and Technology, UNIFESP, São José dos Campos-SP, Brazil

Article history
Received: 05-06-2024
Revised: 07-08-2024
Accepted: 28-09-2024

Corresponding Author:
Aloysio Augusto Rabello de Carvalho
Institute of Science and Technology, UNIFESP, São José dos Campos-SP, Brazil
Email: aloysio.rabello@unifesp.br

Abstract: The relevance of embedded systems has increased considerably due to industrial automation and the adoption of IoT and medical devices. This process demands the adaptation of software engineering techniques that involve the maintenance and evolution of software for use in embedded systems. We conducted this Systematic Literature Review (SLR) to investigate the state of the art on the topics of maintenance and evolution of software applied to embedded systems. The purpose was to identify their key characteristics using a total of 67 primary studies from 1992-2023. The results of this review can encourage future research into improving the software maintenance and evolution process. The current state of the art suggests a need for more research in the field, mainly in the area of unified processes to support software maintenance and evolution. The main findings in this research include the impact of maintenance and evolution on the supporting processes, the software life cycle, their relationships, the essential maintenance strategies, code inspection and analysis, review of technical debt, code refactoring, difficulties in applying protocols and strategies, component obsolescence analysis, and modification protocols.

Keywords: Maintenance, Evolution, Embedded Software, Embedded System

Introduction

Currently, the importance of embedded systems for the automotive industry, medical devices, consumer electronics, electronic voting machines, and the Internet of Things (IoT) is indisputable and their maintenance and evolution are determinants for the development of new products and technologies (Vishwakarma *et al.*, 2019; Sasirekha *et al.*, 2019).

Embedded systems are responsible for controlling and monitoring a wide range of devices, from industrial machinery to medical devices. They are vital for the operation of many critical systems and processes and their importance will only increase as the IoT continues to grow (Vii *et al.*, 2019; Choudhury *et al.*, 2008).

The maintenance of embedded software is a complex and challenging task, but it is crucial to ensure the reliable and safe operation of systems (Borges and Rodrigues, 2011). Engineers in embedded software must have a deep understanding of the systems and processes involved, as well as the tools and techniques necessary to effectively perform maintenance activities (Ruchkin *et al.*, 2015; Raghunathan *et al.*, 2005).

The evolution of embedded systems is a continuous process and maintenance engineers must always be up-to-date on the latest trends and technologies (Srovnal and Penhaker, 2007). They must also be capable of identifying and resolving issues quickly and efficiently to minimize the impact of maintenance on system operation (Zeng *et al.*, 2016; von Knethen, 2002).

The search for scientific articles was conducted using six research sources related to the maintenance and evolution of embedded software. The period from 1992 to 2023 was chosen to capture the evolution and development of embedded systems technologies over three decades. This timeframe spans from the early widespread use of embedded systems in industrial and consumer applications to recent innovations driven by the Internet of Things (IoT).

Major Changes and Trends in Embedded Systems Development (1992-2023) included:

1. In the 1990s, industrial automation began to gain traction with the implementation of embedded systems in machinery and industrial processes. This led to increased efficiency and precision in industrial operations.
2. The 2000s witnessed the explosive growth of mobile devices, such as smartphones and tablets, which heavily relied on embedded systems to

provide advanced functionalities in a compact form factor.

3. The Internet of Things (IoT) emerged as a dominant force in the 2010s, connecting a vast array of smart devices that depend on embedded systems for communication and data processing. This expanded the scope and complexity of embedded systems.
4. Integration of AI and Machine Learning (2020s). Recently, there has been an increasing integration of artificial intelligence and machine learning into embedded systems, enabling functionalities such as voice and image recognition, which are crucial for modern applications including autonomous vehicles and smart healthcare devices.
5. Security and Reliability (2020s). With the rise of cyber threats, the security of embedded systems has become a critical priority. There is a growing focus on developing robust security techniques to protect data and ensure the reliable operation of critical devices.

These changes and trends reflect the continuous evolution of embedded systems, adapting to new technological and market demands. This study examines how the maintenance and evolution of embedded software have kept pace with these transformations, providing insights into best practices and challenges encountered over this period.

The aim of this SLR is to investigate current practices and advancements regarding the maintenance and evolution process of embedded software, in an attempt to identify processes, strategies, code quality effectiveness, and the reduction of maintenance time and failures.

As an initial part of the doctoral dissertation, we conducted this SLR following Kitchenham's methodology. The SLR was divided into three stages: Planning, execution, and analysis of the results obtained, as presented throughout this study (Kitchenham and Charters, 2007).

The primary reason for the SLR was the lack of published reviews with well-defined methodologies about the maintenance and evolution of embedded software, containing objective comparisons in the literature. The principal objectives of this SLR were to investigate the maintenance and evolution of embedded software, identifying the principal methodologies, techniques, and studies in the field, strategies in use, as well as their respective advantages and disadvantages. Other objectives include analyzing the research and development advances in the area and identifying gaps or improvements that can address current issues.

Background

This section aims to provide a solid foundation for understanding the essential concepts that will be explored throughout the study. The theoretical and contextual fundamentals necessary to contextualize the analyses and results discussed later will be presented. Additionally, other relevant academic works that served as a basis for the development of this study will be highlighted.

Definitions

To define the scope and clarify the terms adopted in this review, ensuring consistency, we present the following definitions, organized in alphabetical order:

- Software life cycle: A process that encompasses all stages from creation to deprecation. It includes phases such as requirement analysis, design, implementation, testing, maintenance, and evolution. These stages are adapted to the specific needs of embedded systems, taking into account hardware and software constraints.
- Software evolution: A continuous process of enhancing and updating software over time to meet market demands and new technologies. This includes performance improvements, bug fixes, addition of new features, and adaptation to changes in system and user requirements.
- Maintenance: Aimed at ensuring the continuous and reliable operation of the system over time. This includes bug fixes, security updates, performance optimizations, and adaptations to changes in requirements or the operational environment.
- Embedded system: A specialized system designed to perform specific functions within a larger device or system. It is characterized by being embedded in dedicated hardware and executing pre-defined tasks autonomously and efficiently.
- Technical debt: Refers to the accumulation of shortcuts, compromises, or incomplete solutions made during the development process that may lead to future problems or inefficiencies.
- Code refactoring: This involves restructuring existing code without changing its external behavior to improve readability, maintainability, and efficiency. This process helps eliminate technical debt, reduce complexity, and enhance the software's adaptability to future changes.
- Code inspection: A systematic review process where developers analyze source code for defects, compliance with coding standards, and adherence to best practices. This proactive approach helps identify and rectify issues early in the development lifecycle, ensuring higher software quality and reliability.

Related Work

- Khezami *et al.* (2021) conducted a systematic literature review using 109 studies from 2006 to 2020 to address the following research questions: ”What are the software maintenance activities for Cyber-Physical Systems (CPS)?”; ”What are the techniques used for the automation of software maintenance for CPS?"; "What are the common evaluation methods used to validate software maintenance techniques for CPS?"; "What are the main types of software maintenance used for CPS?" In this SLR, it was possible to conclude about four different areas: Maintenance; techniques; types; and evaluation of methods applied to CPS. The results of this SLR can help developers and researchers identify the status, structure, and potential gaps in the field.

- Cico *et al.* (2023) conducted a systematic review of Artificial Intelligence (AI) and its application across the entire domain of Software Engineering (SE), as evidenced by an increase in SLRs. This SLR aimed to provide an overview of existing systematic reviews in the field of AI supporting the five principal software engineering processes: Requirements, design, development, testing, and maintenance. Utilizing 11 reviews published between 2000 and 2021, including results from 513 primary studies, their conclusion was that reviews on AI-assisted software testing are the most common, followed by software maintenance and development. The study can help researchers find missing evaluations on AI and SE-assisted topics to further consolidate this research area.
- Shen *et al.* (2012) conducted a systematic review involving embedded software and its relationship with agile development methods using 40 primary studies to address the research questions: "What is the general information of embedded software development that is related to Agile Methods?"; "What is the current status of applying agile software development methods to embedded development?"; and "What are the implications of these studies for the industry and the research community?". This SLR shed light on the usage of methodologies such as Extreme Programming, Scrum, Test-Driven Development, Feature-Driven Development, Adaptive Software Development, Rational Unified Process (RUP), and Crystal Agile in embedded software development.
- Queiroz *et al.* Gadelha Queiroz and Vaccaro Braga (2014) conducted a systematic review studying the Product Line Engineering (PLE) and Model-Driven Engineering (MDE) methodologies applied to the development of Safety-Critical Embedded Systems (SCES). These combined methods offer a solution to reduce complexity and time-to-market of systems. Using 19 primary studies, it was possible to conclude that the number of studies using PLE with MDE to build SCES is relatively small but has gradually increased in recent years. Approaches diverge on what is necessary to build model-driven products. Most approaches do not consider differentiating hardware and software variation. Most studies propose the use of UML and feature diagrams. The studies present implemented case studies in different tools and most of them are free. Approaches do not cover the entire development lifecycle.
- Lakshman *et al.* Lakshman and Eisty (2022) conducted a systematic review studying Internet of Things (IoT) devices and Tiny Machine Learning (TinyML) that allowed for the deployment of ML models for embedded vision, bringing together the power of IoT and ML. Through the SLR, it was possible to aggregate the challenges reported by TinyML developers and identify approaches in software engineering, machine learning, and embedded systems addressing the key challenges of TinyML-based embedded vision IoT.
- Garousi *et al.* (2018) conducted an SLR using 272 primary studies on embedded software testing. The review provides an index of the most used testing methodologies in embedded systems, such as criteria-based, test-case design and requirements, automated test-code generation, test management, and integration tests.

Although the above works address various aspects related to the maintenance and evolution of embedded software, none of these works conduct an extensive identification and mapping of applicable approaches. Our SLR offers a broader and more detailed examination of the field. It identifies key methodologies, validation techniques, and future research directions, contributing to a more holistic understanding of the challenges and opportunities in this domain.

The methodologies discussed in this section include hardware and software co-design, co-synthesis, hierarchical abstraction models, and partitioning. Our SLR extends this by cataloging all methodologies presented in the articles and analyzing their validation methods. Validation techniques used in the articles from the "Related Work" include experimentation, comparison, case studies, and simulation. Our SLR highlights the predominance of experimental validation and provides a detailed account of other validation methods employed across various studies. The key findings from the "Related Work" emphasize the need for efficient maintenance processes, the integration of AI in software engineering, and the application of agile methodologies in embedded systems. Our SLR corroborates these findings but also identifies significant gaps and areas for future research, particularly in the automation of maintenance activities and the comprehensive application of AI in embedded systems.

Research Methodology

This section presents the planning process of the SLR, how the PICOC table was filled, the search string and the databases of papers used, and the inclusion and exclusion criteria for the selected studies.

The following terms are used to fill the PICOC table adopted in the SLR:

Population: Embedded software maintenance and evolution processes,

Intervention: Embedded software maintenance and evolution strategies, for example, code inspection strategies, code refactoring, component obsolescence analysis,

Comparison: Embedded software maintenance and evolution processes,

Outcome: Code quality effectiveness, maintenance time reduction, defect reduction after maintenance,

Context: Embedded software development.

The search string used to retrieve articles was as follows: *(”Embedded Software”) AND ((”Evolution”) OR (”Maintenance”) OR (”updating”) OR (”refactoring”))*. The string was formulated for articles related to embedded systems and addressing the topics of evolution, maintenance, and a few synonyms and it was based on the keywords we used in the PICOC table.

Justification for the Choice of Databases

While we have taken several measures to minimize biases and limitations in our SLR, it is crucial to recognize that no process is completely free from biases. The discussions above provide transparency about the limitations encountered and how we attempted to mitigate them, ensuring that readers can interpret the results with a clear understanding of the potential sources of bias.

This SLR searched for scientific papers using six research sources related to the maintenance and evolution of embedded software. The databases are listed in Table (1) and represent the principal ones in the field of technology and computer science currently. They were chosen to minimize the possibility of not finding relevant papers.

Table 1: Used research sources

Source	URL
ACM Digital Library	http://portal.acm.org
IEEE Digital Library	http://ieeexplore.ieee.org
ISI Web of Science	http://www.isiknowledge.com
ScienceDirect	http://www.sciencedirect.com
Scopus	http://www.scopus.com
Springer Link	http://link.springer.com

ACM Digital Library

Reason: The ACM Digital Library is one of the leading sources of research in computer science and software engineering. It provides access to a vast collection of conference proceedings, journals, and magazines, offering comprehensive coverage of the latest research and developments in embedded systems.

Criterion: Selected for its relevance and significant impact on the research community in embedded systems and software engineering.

IEEE Digital Library

Reason: The IEEE Digital Library is renowned for its extensive collection of publications in electrical engineering, electronics, and computer science. It is a crucial source of information on technological advancements, standards, and practices in the embedded systems industry.

Criterion: Chosen for its authority and credibility in the field of embedded systems, as well as for the wide range of topics covered.

ISI Web of Science

Reason: The ISI Web of Science is a multidisciplinary database that includes comprehensive citation indices, enabling the identification of highly cited and influential articles. It is valuable for tracking the impact of research over time.

Criterion: Selected for its ability to provide a comprehensive and interconnected view of the most influential research and its citations, essential for a systematic literature review.

ScienceDirect

Reason: ScienceDirect offers access to a vast collection of peer-reviewed journals, particularly in the exact sciences and engineering. It is an important source of up-to-date and high-quality articles.

Criterion: Chosen for its broad coverage of relevant topics and the reputation of its journals, ensuring access to rigorous and peer-reviewed research.

Scopus

Reason: Scopus is one of the largest abstract and citation databases of academic literature, covering a wide range of disciplines. It is known for its ability to track global scientific production and its citations.

Criterion: Selected for its comprehensiveness and ability to provide impact metrics and research trends, essential for a complete and up-to-date analysis of the field.

Springer Link

Reason: Springer Link offers access to a wide range of books, journals, and conference papers in various scientific areas, including engineering and computer science.

Criterion: Chosen for the quality and relevance of its publications, as well as the availability of up-to-date and peer-reviewed content, fundamental for a comprehensive systematic review.

Considered Limitations

We acknowledge that the choice of these databases may have limited the inclusion of some relevant studies published in other sources. However, the selected databases were chosen due to their reputation, comprehensiveness, and specific relevance to the study area. Limitations such as access availability to certain articles and variation in journal indexing were also considered during the search process.

Discussion on Possible Biases and Limitations in the Search and Selection Process

Although we followed a rigorous methodology to ensure the comprehensiveness and quality of our Systematic Literature Review (SLR), it is important to recognize and discuss possible biases and limitations that may have impacted the search and selection process of the studies. Below are some of the main considerations:

Publication Bias

Description: Publication bias occurs when studies with positive or significant results are more likely to be published than those with negative or non-significant results.

Impact: This can lead to an overestimation of the effects or conclusions drawn from the review, as important studies with negative results may not have been included. Mitigation: To mitigate this bias, we included both conference papers and journal articles and used multiple databases to ensure a more comprehensive search.

Selection Bias

Description: Selection bias can occur due to the inclusion and exclusion criteria applied during the screening of studies.

Impact: Relevant studies may have been excluded if they did not meet specific defined criteria, even if they contained valuable information.

Mitigation: We defined clear and rigorous inclusion and exclusion criteria and conducted independent double-checking during screening to reduce the risk of inadvertent exclusion of relevant studies.

Database Limitations

Description: Each database has its own limitations in terms of coverage and accessibility.

Impact: Some relevant studies may not be indexed in the chosen databases, which can limit the comprehensiveness of the review.

Mitigation: We chose six of the most recognized and comprehensive databases in the field of software engineering and embedded systems. However, we acknowledge that studies published in other databases or non-indexed sources may not have been included.

Time and Resource Limitations

Description: Conducting a comprehensive systematic review is a time-consuming and resource-intensive process. Impact: Time and resource constraints may have limited the extent of the search and the depth of the analysis.

Mitigation: We prioritized quality over quantity, ensuring that selected studies were thoroughly reviewed and data were extracted rigorously and consistently.

Linguistic Limitations

Description: Studies published in languages other than English were excluded, which may have led to the omission of relevant research published in other languages.

Impact: This can introduce a linguistic bias in the review. Mitigation: We acknowledge this limitation and suggest that future reviews consider including studies in other languages for a more global perspective.

Inclusion and Exclusion Criteria

Papers that present strategies on how they can be integrated into a single and efficient process of maintenance and evolution of embedded software.

Papers presenting software tools available to support embedded software maintenance and evolution processes. Papers presenting current models and processes supporting embedded software maintenance and evolution.

Papers presenting an integrative process of embedded software maintenance and evolution.

Papers that describe the impact of software maintenance and evolution on its lifecycle.

Papers presenting maintenance strategies, code inspection and analysis, technical debt review, code refactoring, component obsolescence analysis, and modification protocols.

Exclusion Criteria: Duplicated papers; papers not written in English; short papers; and papers out of scope.

Research Questions

The research questions were developed based on the PICOC table, which allowed the identification of the key dimensions to be investigated in this SLR.

To conduct the SLR, the following research questions presented in Table (2) were adopted.

Study Selection Process

After conducting a search in all the articles previously mentioned, a total of 6844 articles were returned for inclusion in this SLR selection process. The selection process was carried out in five stages: The first stage was the search, the second involved reading the titles, the third stage consisted of reading the abstracts, the fourth stage involved reading the introduction and conclusion of the selected articles, and the fifth stage involved reading the complete articles. From stages one to four, articles that did not meet the inclusion criteria or met the exclusion criteria, such as duplicated articles and irrelevant ones, were discarded. In the fifth stage, along with the complete reading of the articles, the data extraction table was filled out. The following Figure (1) presents an infographic that illustrates the steps taken from the search to the completion of the article selection process to be used in this SLR.

Table 2: Research questions for the systematic review

ID	Questions	Objective
Q01	How can software evolution and maintenance modify the life cycle of embedded software?	This question aims to investigate possible modifications in the life cycle of embedded software resulting from the software maintenance and evolution process.
Q02	What are the current models and processes supporting the maintenance and evolution of embedded software? How do these processes and models compare to each other?	This question aims to identify the models currently in use in embedded systems development and whether multiple models can be used in the same project.
Q03	What are the maintenance, code inspection and analysis, technical debt review, code refactoring, component obsolescence analysis, modification protocols, and other strategies used in embedded software maintenance and evolution processes?	This question was designed to catalog the strategies and protocols adopted in the processes of maintenance and evolution of embedded.
Q3.1	What are the difficulties (or gaps) encountered in the application of the studied software maintenance and evolution strategies?	This question aims to identify the difficulties professionals encounter in applying embedded software maintenance and evolution strategies.
Q3.2	How can these strategies be integrated into a single and efficient process of embedded software maintenance and evolution?	This question attempts to find possibilities of integration among the strategies found in the SLR.
Q04	What are the available software tools to support embedded software maintenance and evolution processes? How does the tool developed in this study compare to existing tools?	This question tries to define the tools supporting embedded software development used by industry professionals.
Q05	How can an integrative process of embedded software maintenance and evolution be specified?	This question verifies how a process can be specified and what are the key characteristics necessary to support embedded software maintenance and evolution.

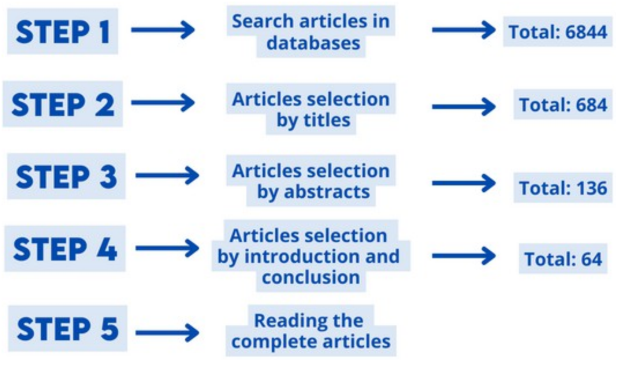
STEP 1
Search articles in databases
Total: 6844
STEP 2
Articles selection by titles
Total: 684
STEP 3
Articles selection by abstracts
Total: 136
STEP 4
Articles selection by introduction and conclusion
Total: 64
STEP 5
Reading the complete articles

Fig. 1: Study selection steps

Data Extraction Spreadsheet

For the data extraction of the articles accepted in this SLR, a spreadsheet was filled in with the relevant information of each article to facilitate future insights, conclusions, and discussions related to the context of this review.

The spreadsheet has the following layout: One row for each article and one column for each question.

Results and Analysis

This section presents the most important findings of the SLR and discusses the results in light of the research questions.

Q1 - How can Software Evolution and Maintenance Modify the Life Cycle of Embedded Software?

The initial analysis of the results was conducted with the first research question: Does the study investigate the relationship between evolution/maintenance and life cycle? In this context, 12 articles addressed this question, with 5 providing a complete answer and 7 providing a partial one. The essential characteristics related to the maintenance and evolution of software that influence the lifecycle of embedded software include:

- Lack of a systematic approach in early development stages: This can result in a rigid system posing a challenge for its maintenance and evolution, such as code or component reuse (Nam and Lee, 2003)
- Absence of continuous modeling and consistent description: Ismail and Jerraya (1995) mention that the absence of continuous modeling and consistent description of hardware and software can retard the refinement of the embedded system
- Intimate connection of lifecycle with evolution and interactions: Liao *et al.* (1997) states that the lifecycle is intimately connected to the evolution and interactions of the hardware and software components, emphasizing the importance of considering evolution and maintenance in design decisions. This article highlights that the choices during hardware-software synthesis must take into

account not only the immediate performance but also the evolution and maintenance capacity over the years.
- Flexibility and innovation in high-technology projects: Iguider *et al.* (2019) points out the flexibility, innovation, and momentum in dealing with uncertainties and complexities in high-technology projects, where system life cycles can be dynamic and subject to continuous changes
- Movement of functionalities between hardware and software: Sabella *et al.* (2017) affirms the importance of allowing the movement of functionalities between hardware and software to address changes in requirements as the design evolves
- Autonomous management of software development and project life cycles: Although the article by Lin and Chen (2017) does not directly address the relationship between maintenance and life cycle, it mentions that the ACCLIB framework can autonomously manage software development and project life cycles, as illustrated in Fig. (2).

Although little was found on the life cycles of embedded systems during our research, it is possible to infer from the selected articles that maintenance and evolution strategies can significantly impact the project's life cycle:

- Worst-case scenario: It may be necessary to revert to the initial stage (requirements gathering) if there is inadequate planning, affecting the entire project as requirements are crucial for development (Sabella *et al.*, 2017)
- Delaying the final stage: A project with good software maintenance and evolution can extend the lifespan of the embedded system without the need for replacement, resulting in resource savings such as personnel and hardware

Q2 - What are the Current Models and Processes Supporting the Maintenance and Evolution of Embedded Software? How do these Processes and Models Compare to Each Other?

The second research question adopted to assess the quality of the selected articles was: Does the article respond to how maintenance/evolution modifies the life cycle? For this question, partial answers were found in the following articles:

- The use of agile methodologies ensures a safer, more agile, and flexible approach to dealing with the uncertainty and complexity inherent in the life cycles of these systems (Iguider *et al.*, 2019)
- The importance of employing incremental testing and proto-testing to ensure the quality of software-hardware across all life cycles (Iguider *et al.*, 2019)
- Jayakumar and Khatri (2004) emphasize the use of a modular design to ensure simplified maintenance during system life cycles.
- Finally, Zhang *et al.* (2017) addresses the hardware life cycle, with its emphasis more directed towards functional verification and logical simulation of integrated circuits. It discusses the importance of simulation in verifying the correctness of an integrated circuit and presents a platform for efficient parallel simulation of high-level specifications using the RTL/C/C++ language.

In the articles selected for this SLR, it was not possible to find a specific model for software maintenance and evolution. However, many articles use some development methodology to maintain and evolve embedded systems, for example:

- The Software Development Life Cycle (SDLC) is a software development process model that describes the phases of software development, testing, deployment, and maintenance. Although it is not a specific model for embedded systems development, it has been used by Nam and Lee (2003)
- The Embedded Software Development Lifecycle (ESDL) is a software development process that is specific to embedded software. Articles typically describe the phases of specification, design, implementation, testing, deployment, and maintenance of embedded software (Jayakumar and Khatri, 2004)
- Architecture-oriented Software Development (AOSD) is a software development process that emphasizes the importance of software architecture. The selected articles typically describe the phases of requirements analysis, architecture, implementation, testing, and deployment of software (Iguider *et al.*, 2019)

No relation or comparison between the models or processes used for the evolution and maintenance of embedded systems was found in the articles.

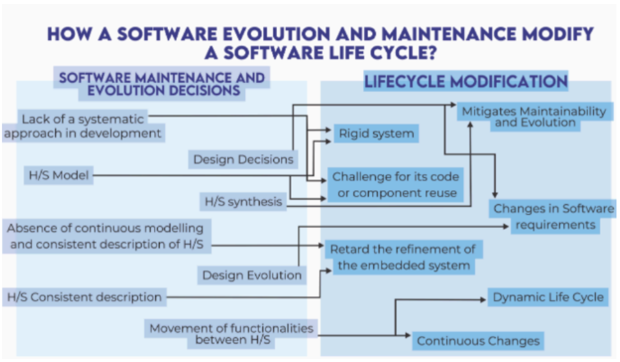
HOW A SOFTWARE EVOLUTION AND MAINTENANCE MODIFY A SOFTWARE LIFE CYCLE?
SOFTWARE MAINTENANCE AND EVOLUTION DECISIONS
LIFECYCLE MODIFICATION
Lack of a systematic approach in development
Design Decisions
H/S Model
H/S synthesis
Absence of continuous modelling and consistent description of H/S
Design Evolution
H/S Consistent description
Movement of functionalities between H/S
Mitigates Maintainability and Evolution
Rigid system
Challenge for its code or component reuse
Changes in Software requirements
Retard the refinement of the embedded system
Dynamic Life Cycle
Continuous Changes

Fig. 2: Actions and implications in software maintenance and evolution

Q3 - What are the Maintenance, Code Inspection and Analysis, Technical Debt Review, Code Refactoring, Component Obsolescence Analysis, Modification Protocols, and Other Strategies Used in Embedded Software Maintenance and Evolution Processes?

For the third research question, only two articles addressed the topic. Olukotun *et al.* (1994) partially answers this question, discussing strategies such as hierarchical modeling, formal and functional specification, the use of design and simulation in the early stages of the project, as well as the importance of effective testing to verify development and ensure system correctness. On the other hand, the article by Li *et al.* (2023) focuses on analyzing self-admitted technical debt in embedded systems and developers' attitudes toward this debt. Strategies such as analyzing sources of technical debt, applying machine learning techniques, and managing technical debt are addressed in this context.

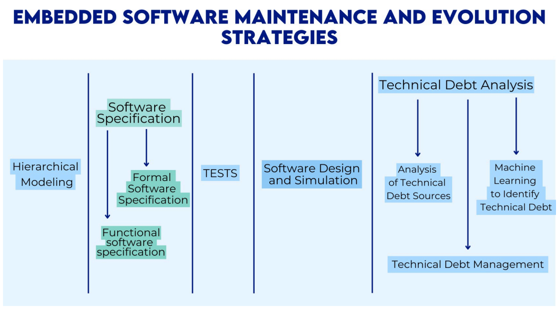
EMBEDDED SOFTWARE MAINTENANCE AND EVOLUTION STRATEGIES
Hierarchical Modeling
Software Specification
Formal Software Specification
Functional software specification
TESTS
Software Design and Simulation
Technical Debt Analysis
Analysis of Technical Debt Sources
Machine Learning to Identify Technical Debt
Technical Debt Management

Fig. 3: Strategies used in embedded software maintenance and evolution

From the data collected in this SLR, it was possible to find the key strategies for the maintenance and evolution of embedded software, such as Code inspections and analysis, technical debt review, code refactoring, component obsolescence analysis, and modification protocols, as can be seen in Figure (3):

- Code inspections and analysis: This strategy involves the software source-code review to identify quality issues such as bugs, coding errors, and excessive complexity. The inspection and analysis of code can be manual or supported by automated tools.
- Technical debt review: This strategy involves the assessment of technical debt, which is the cost of software evolution and maintenance due to quality issues, such as excessive complexity, and lack of documentation and tests. The technical debt review can be manual or supported by automated tools.
- Code refactoring: This strategy involves modifying the software's source code to improve its quality without altering its functional behavior. Code refactoring can be done manually or with the assistance of automated tools.
- Component obsolescence analysis: This strategy involves evaluating software components to identify those that are obsolete or no longer needed. Component obsolescence analysis can be done manually or with the assistance of automated tools.
- Modification protocols: This strategy involves defining a set of rules and procedures for making modifications to the software. Modification protocols help ensure that modifications are carried out in a controlled and secure manner.

Q3.1 - What are the Difficulties (or Gaps) Encountered in the Application of the Studied Software Maintenance and Evolution Strategies?

The issue of scoring on the difficulties in implementing the studied software maintenance and evolution strategies was investigated, resulting in 19 articles with comprehensive answers and 3 articles with partial answers. Among the partial responses, the following observations stand out: Article Liao *et al.* (1997) emphasizes the importance of making design decisions that consider both hardware and software, as well as evaluating the dynamic and static properties of the system and the costs associated with hardware. An article by Karthik *et al.* (2018) reveals that the difficulties in implementing the strategies lie in seeking efficient solutions in terms of hardware and execution time while respecting global area and execution time constraints.

From the articles that fully address the question:

- Ismail *et al.* (1994) discusses the challenge concerning the automatic determination of an appropriate set of instructions extensible for a specific application as well as the need for additional research in system-level design methodologies and Network-on-Chip (NoC) architectures. It also focuses on the growing gap between the technology available and the real complexities of System-on-Chip (SoCs)
- Buchenrieder *et al.* (1993) highlights difficulties such as the pressure to reduce the time to market electronic devices, the increase in the costs of engineering and manufacturing, the need to reconfigure, and the programmability of SoCs.
- Srinivasan *et al.* (1998) focuses on the specific challenges faced when designing integrated circuits given the cross-talk problem in Deep Sub-Micron (DSM) design.
- Ernst *et al.* (1993) identifies the problems in applying the strategies of embedded systems development, including the increasing complexity of the design space and the difficulty in finding an optimal solution for hardware and software partitioning.
- Ronkainen and Abrahamsson (2003) discuss challenges related to hardware/software co-

synthesis for embedded systems, highlighting the importance of minimizing communication costs and improving parallelism for system performance

- Huang *et al.* (2012) addresses challenges faced by hardware and software designers due to changes in computer architecture, including the need to consider energy consumption and parallelism constraints at unprecedented levels.
- Sabella *et al.* (2017); and Ortega-Cabezas *et al.* (2020) mention the inherent uncertainty and complexity in high-technology projects, the need to deal with emerging and unknown technologies, and the importance of addressing innovation and risk management challenges
- Sabella *et al.* (2017) highlights the difficulty of specifying a detailed hardware-software interface and the need to deal with changes in system requirements during the design process.
- Zhang *et al.* (2017) discuss challenges in developing hardware for accelerating computationally intensive algorithms, emphasizing the need for collaboration among engineers from different fields.
- Lin and Chen (2017) emphasize the design effort required to design a hardware accelerator and highlight the limitations of traditional approaches to addressing this problem.
- Zuo *et al.* (2017); and Yousuf and Gordon-Ross (2016) mention the challenge of achieving optimal SoC design due to the vast design space of hardware accelerators and the challenge of task partitioning between accelerators and CPUs
- Lora *et al.* (2019) presents difficulties in the design process of partial reconfiguration and proposes a solution to simplify it.
- Finally, Sabella *et al.* (2017) highlights the pressure and the need to meet delivery deadlines as factors leading developers to incur technical debt.

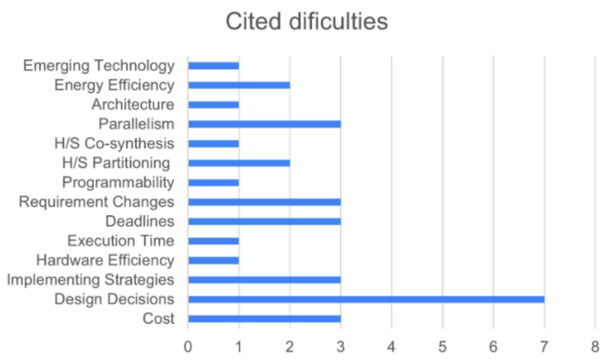

Fig. 4: Cited difficulties

A graph with the most cited difficulties can be seen in Figure (4) and the most cited gaps can be seen in Fig. (5). A total of 26 articles were examined regarding the availability of software tools to support the maintenance/evolution process. The major difficulties identified include: Difficulty in making design decisions that consider hardware and software; gaps in microcontroller complexity; pressure to reduce the time to market for electronic products; considering hardware design alternatives for each task; minimizing communication cost and improving parallelism; changes in computer architecture; inherent complexity in high-tech projects; difficulty in specifying a detailed hardware-software interface; achieving an optimal SoC design; complexity of the design processes of partial reconfiguration; the pressure and the need to meet delivery deadlines.

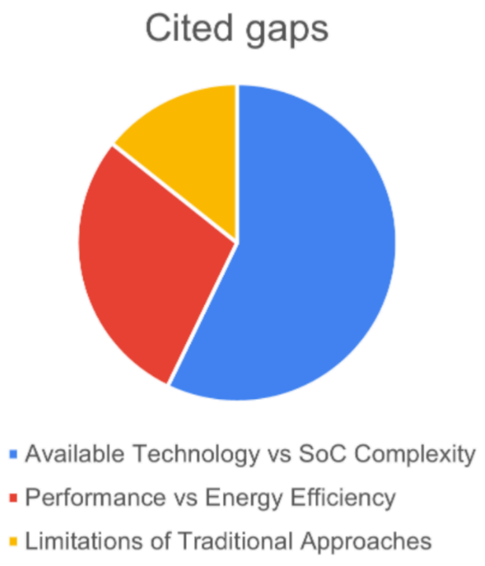

Fig. 5: Cited gaps

Q3.2 - How Can These Strategies be Integrated into a Single and Efficient Process of Embedded Software Maintenance and Evolution?

There is no right answer to this question, as the best approach to integrating two or more embedded software maintenance and evolution strategies depends on the specifics of each project.

It is important to clearly define the objectives of each maintenance and evolution strategy. This can ensure that the strategies are aligned and that efforts are directed towards the most relevant areas. Identifying integration points between strategies is important so that strategies are efficiently integrated and that data and processes are shared effectively. Developing an integration plan that outlines the necessary steps for integrating strategies is also essential because well-defined objectives guarantee successful integration and mitigate risks (Nam and Lee, 2003; Ortega-Cabezas *et al.*, 2020; Lora *et al.*, 2019), as can be seen in Figure 6.

The integration of two or more strategies for embedded software maintenance and evolution can be a complex process, but it can be extremely beneficial for the project.

Q4 - What are the Available Software Tools to Support Embedded Software Maintenance and Evolution Processes? How Does the Tool Developed in this Study Compare to Existing Tools?

The analysis of the articles revealed a variety of crucial tools to support the maintenance and evolution process of embedded software. These tools were categorized as follows:

- Solar: This is a co-design environment that includes a partitioning tool, significantly enhancing the efficiency of the development process
- Vivado: Developed by Xilinx, Vivado is a widely used software in the semiconductor industry. It offers a vast range of tools and supports various platforms, including processors and Field Programmable Gate Arrays (FPGAs)
- Xilinx's SDK: This tool is used for programming in the C language and also for fault injection campaigns controlled by MicroBlaze processors
- LLVM: A compiler infrastructure developed to optimize the compilation, linking, and execution times of software written in various languages
- HIFSuite: A framework in a customized virtual platform that facilitates the integration of models from different domains and abstraction levels
- ACCLIB: A framework that automates the identification, matching, and integration of hardware accelerators into computing systems, simplifying the integration process for developers
- SnoopP (Snooping Profiler): This tool provides accurate profiling information that can significantly influence system partitioning and design
- Bluespec Codesign Language (BCL): A tool for handling hardware-software development, allowing hardware and software partition specifications in a single language
- CoDeveloper and Impulse C: These tools are used for developing reconfigurable architectures and for generating Very High-speed Integrated Circuit Hardware Description Language (VHDL) from a standard C language description
- Retargetable Tool Generation: Used for evaluating customizations and rapid iteration during design exploration
- Computer-Aided Design (CAD) and Drafting Tools: Used to create hardware electrical schematics
- SystemC Language: Used to generate Hardware Description Language (HDL) and model description
- Green Flash Project: A project providing highly configurable and accurate simulation of node architectures
- Matlab and LabView: High-level applications used for generating HDL.
- ROSE Compiler Framework: A framework offering automatic extraction and extrapolation of memory traces and interconnection
- Structural Simulation Toolkit (SST)/macro: A large-scale interconnection simulator

It's important to note that the data was collected only from articles that cited tools available for use or provided detailed descriptions of their functionalities. There were no comparisons with classical tools for supporting the maintenance and evolution of embedded software. A graph depicting the types of all tools listed by the accepted articles is presented in Figure (7).

No data was found presenting comparisons between developed tools and any classical tool for supporting the maintenance and evolution of embedded software.

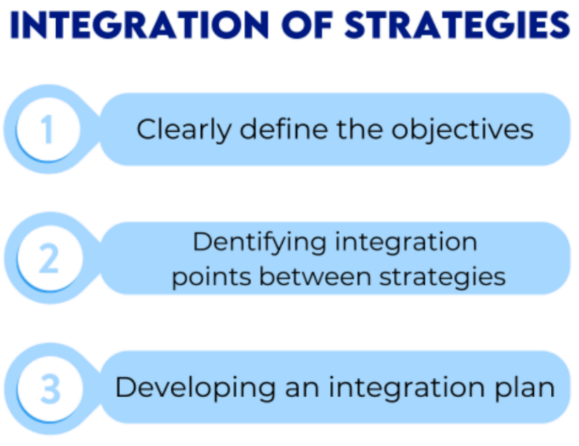
INTEGRATION OF STRATEGIES
1
Clearly define the objectives
2
Dentifying integration points between strategies
3
Developing an integration plan

Fig. 6: Steps to integrate maintenance and evolution strategies

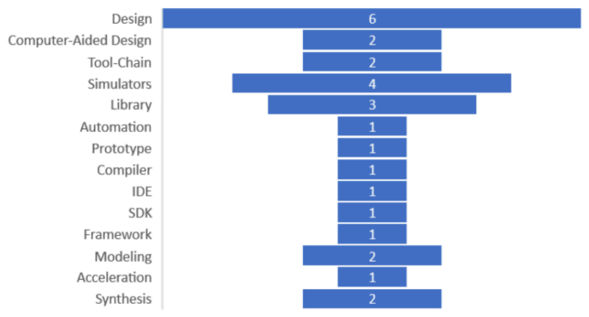

Fig. 7: Type of Tools Found in This SLR

Q5 - How Can an Integrative Process of Embedded Software Maintenance and Evolution Be Specified?

No evidence was found to answer this research question.

Development Methodologies Used in the Selected Articles

During the execution of this SLR, all methodologies presented in the articles were cataloged. For this purpose, the methodologies used (even when more than one methodology was used) and their validations were taken into consideration. Below, we present the data collected in light of the methodologies used in the articles.

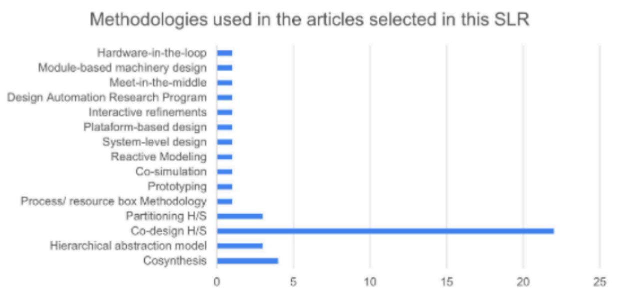

Fig. 8: Methodologies used in the articles selected in this SLR

In Figure (8) there is a chart representing all the methodologies used in the articles. Only methodologies that contain some degree of validation were counted.

A significant difference can be observed between the numbers attributed to hardware and software codesign compared to the numbers of other methodologies. Although this methodology was created in the 1990s, it can be inferred that it is still essential for the development of embedded systems:

- Hardware and software co-design is a development methodology that involves integrating hardware and software design from the beginning of projects. It is typically defined in 5 stages, such as System requirements definition; hardware design; software design; hardware and software integration; and system testing.
- Hardware and software co-synthesis, which ranked second, is a development methodology that combines both hardware and software to transform a high-level description into a hardware project (logical and layout) and a low-level software description (which can be a detailed description or even code).
- The hierarchical abstraction model methodology is based on the four main levels of abstraction during development: business rule level; conceptual level; logical level; and finally, the physical level.
- Hardware and software partitioning is a development methodology that involves dividing a system into independent hardware and software components. This can be done to facilitate system development, maintenance, and updates.

Validations Used in the Selected Articles

During the execution of this SLR, all forms of validation described in the accepted articles were cataloged. The validations used were taken into consideration (even when more than one validation was used). Below, we present the data collected regarding the validations in the articles.

In Figure (9) you can find a chart representing all the validation methods used in the articles. A significant advantage can be observed for experimental validation, followed by comparison, case study, and simulation:

- Experimentation: This validation consists of manually testing the system. It was often used in conjunction with comparison (where two systems are compared) or even conducted with the user.
- Comparison: This involves comparing the system with others, typically well-established ones. Comparisons can be manual or automatic and can evaluate performance, energy efficiency, execution time, failures, and even metrics such as function points per number of lines.
- Case study: This is a detailed description of an event or situation that occurs in the real world. Case studies are used to document and analyze experiences, identify problems and opportunities, and develop solutions. Articles that compare expected behavior with the actual behavior of the system were identified and validated. For example, finding how the embedded system handled different data inputs and how the system behaved under different loads (usage intensity)
- Simulation: This validation technique was predominant in the articles of this SLR, particularly in virtual simulations, where the entire hardware and software of the embedded system were virtualized to conduct unit, integration and some comparative tests

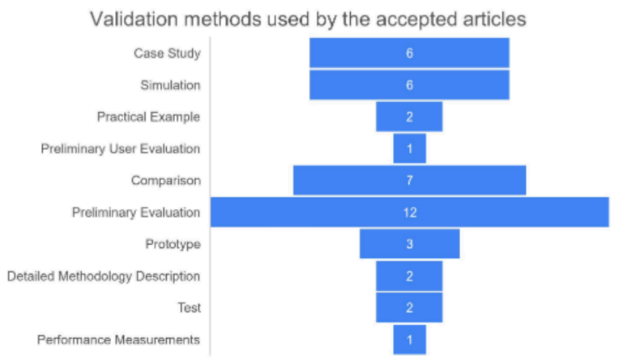

Fig. 9: Validation methods used by the accepted articles

Analysis of the selected studies according to the established criteria

During the execution of this SLR, ten questions with scores were used. These questions served to give each article an impact coefficient. For each question that the article answered, one point was assigned. Partial answers were assigned half a point. If the article did not contain any answer to the question, no score was assigned. Figure (10) shows the score of each article selected in this SLR.

The use of this scoring was assigned to provide a quantitative measure of impact for each article. All these articles were selected for the SLR and none was excluded due to low relevance. The most relevant article was Cauwels *et al.* (2018) with a score of 6.5 out of 10.

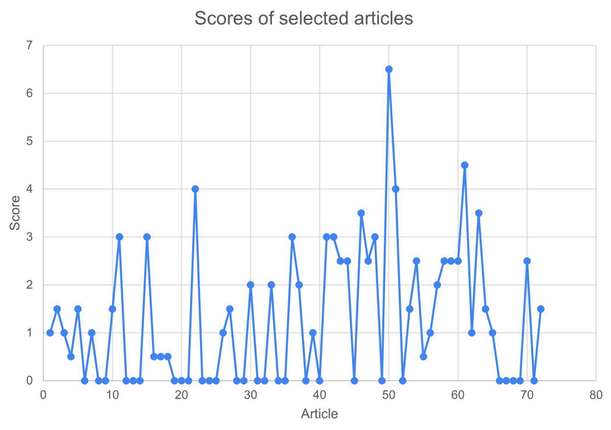

Fig. 10: Scores of selected articles

Conclusion

The execution of this SLR offers insights into how the maintenance and evolution of embedded software can modify project lifecycles, the current support processes, as well as their relationship. It includes maintenance strategies, code inspection and analysis, technical debt review, code refactoring, component obsolescence analysis, modification protocols, and other strategies used in the maintenance and evolution processes of embedded software. In addition, it presents the difficulties encountered in the application of maintenance and evolution strategies, how these strategies can be integrated into a single process and the software tools used to support the maintenance and evolution of software.

In this SLR, it was possible to obtain some information about the impact of maintenance and evolution on the lifecycle of embedded systems. There was evidence that evolution can reveal when the project requirements gathering was mistakenly executed and caused rework, bringing a negative impact. This phase is the most complex to revise as it can generate a cascade effect that requires extensive modifications throughout the project to meet new requirements. There was also evidence that maintenance and evolution can delay the last phase of the lifecycle, discontinuation, meaning that a good maintenance and evolution process can extend the life of an embedded system.

The three main data on the support models for the maintenance and evolution of embedded software found were: SDLC, ESDL, and AoSD. It was not possible to find sufficient data in the papers on how these models compare.

The selected articles provided us with information on code inspection, technical debt review, code refactoring, and component obsolescence analysis, which were the techniques found for modification protocols and support strategies for the maintenance and evolution of embedded software.

The challenges for applying a protocol or a support process for software maintenance identified were time, cost, project complexity, emerging and unknown technologies, appropriacy to overall runtime, gap in microcontroller complexity, requirement changes, and interaction with the client.

It presents how strategies can be integrated into a single support process for the maintenance and evolution of embedded software, which can be summarized in three steps: Clearly define the objectives of each maintenance and evolution strategy; identify integration points between the strategies; develop an integration plan that outlines the necessary steps to integrate the strategies.

From the results obtained regarding the main tools used to support the development, maintenance, and evolution of embedded software, we can mention Solar; Vivado; Xilinx's SDK; LLVM; HIFSuite; CoDeveloper; Impulse C; ACCLIB; SnoopP (Snooping Profiler); Bluespec Codesign Language (BCL); CAD tools; SystemC language; Matlab and LabView.

SLR Limitations

The conduct of a systematic literature review is valuable because its results are evaluated objectively following defined protocols, which are rigorously carried out, thus attempting to eliminate any bias that may arise.

However, even by strictly following all the steps proposed by articles and taking all precautions, there are still threats that can affect the results of the systematic review and bias the conclusions drawn from this SLR. The major threats are:

- Search libraries: Although the 6 most used libraries in the field of Computer Science were selected (ACM Digital Library, IEEE Digital Library, ISI Web of Science, Science@Direct, Scopus, and Springer Link), there are probably other relevant papers to this SLR that are not published in these libraries
- Definition of search strings: The search string selection was developed following guidelines for conducting the SLR and tests were performed with modifications of terms (such as synonyms, for example), order, and logical operators. However, even following guidelines, it is difficult to determine if the string used is an optimal string.

Recommendations for Future Research

Based on the findings of this systematic literature review, several areas require further investigation to advance the understanding and development of maintenance and evolution models for embedded software. The following recommendations are proposed to guide future research endeavors:

1. In-depth Exploration of Specific Models for Maintenance and Evolution of Embedded Software: During the execution of this SLR, limited evidence was found regarding specific models tailored for the maintenance and evolution of embedded software. Future research should aim to conduct a more detailed and focused search on existing models within this niche. Emphasis should be placed on validating models through empirical studies and case analyses in real-world embedded systems environments.
2. Comprehensive Analysis of Relationships and Comparisons Involving Evolution and Maintenance Models: There is a need for more extensive research into the relationships and comparative analyses of different models or processes used for the evolution and maintenance of embedded software. Future studies should systematically compare existing models to identify strengths, weaknesses, and contextual suitability. This involves evaluating various approaches.
3. Investigation of Integrative Maintenance and Evolution Processes: Embedded software maintenance and evolution often require integrative processes that differ from classical software engineering approaches. Future research should focus on specifying these integrative processes, detailing how they accommodate the unique constraints and requirements of embedded systems.
4. Development of Tailored Metrics and Evaluation Frameworks: The development of specific metrics and evaluation frameworks for assessing the effectiveness of maintenance and evolution processes in embedded software is crucial. Future research should aim to establish standardized metrics that reflect the unique aspects of embedded systems, such as real-time performance, power consumption, and hardware dependency.

By addressing these areas, future research can significantly enhance the understanding and effectiveness of maintenance and evolution in embedded software, ultimately leading to more reliable and efficient embedded systems.

Acknowledgment

I am deeply grateful to my advisors for their invaluable guidance and to all the professors at UNIFESP for their continuous support and encouragement. I extend my special thanks to CAPES for providing financial assistance through a scholarship. Finally, I offer my heartfelt gratitude to my family for their unwavering support and understanding throughout this journey.

Funding Information

This study was supported by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Funding Code 88887.809591/2023-00.

Ethics

This research did not involve human or animal subjects and no ethical approval was required.

Author's Contributions

- **Aloysio Augusto Rabello de Carvalho**: Data collection, Conceptualization, methodology, formal analysis, validation, writing original draft & editing.
- **Luiz Eduardo Galvão Martins**: Conceptualization, methodology, formal analysis, validation, writing review & editing.

References

Borges, R. W., & Rodrigues, E. L. L. (2011). Embedded System Design: An Overview of Brazilian Development. *2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications Workshops*, 141-146. https://doi.org/10.1109/ispaw.2011.13

Buchenrieder, K., Sedlmeier, A., & Veith, C. (1993). HW/SW Co-Design With PRAMs Using CODES. *Computer Hardware Description Languages and Their Applications*, 65-78. https://doi.org/10.1016/b978-0-444-81641-2.50010-1

Cauwels, M., Zambreno, J., & Jones, P. H. (2018). HW/SW Configurable LQG Controller Using a Sequential Discrete Kalman Filter. *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 1-8. https://doi.org/10.1109/reconfig.2018.8641738

Choudhury, T., Borriello, G., Consolvo, S., Haehnel, D., Harrison, B., Hemingway, B., Hightower, J., Klasnja, P., Koscher, K., LaMarca, A., Landay, J. A., LeGrand, L., Lester, J., Rahimi, A., Rea, A., & Wyatt, D. (2008). The Mobile Sensing Platform: An Embedded Activity Recognition System. *IEEE Pervasive Computing*, *7*(2), 32-41. https://doi.org/10.1109/mprv.2008.39

Cico, O., Cico, B., & Cico, A. (2023). AI-Assisted Software Engineering: A Tertiary Study. *2023 12th Mediterranean Conference on Embedded Computing (MECO)*, 1-6. https://doi.org/10.1109/meco58584.2023.10154972

Ernst, R., Henkel, J., & Benner, T. (1993). Hardware-software cosynthesis for microcontrollers. *IEEE Design & Test of Computers*, *10*(4), 64-75. https://doi.org/10.1109/54.245964

Gadelha Queiroz, P. G., & Vaccare Braga, R. T. (2014). Development of Critical Embedded Systems Using Model-Driven and Product Lines Techniques: A Systematic Review. *2014 Eighth Brazilian Symposium on Software Components, Architectures and Reuse*, 74-83. https://doi.org/10.1109/sbcars.2014.19

Garousi, V., Felderer, M., Karapicak, C. M., & Yilmaz, U. (2018). What We Know about Testing Embedded Software. *IEEE Software*, *35*(4), 62-69. https://doi.org/10.1109/ms.2018.2801541

Huang, P. M., Darrin, A. G., & Knuth, A. A. (2012). Agile Hardware and Software System Engineering for Innovation. *2012 IEEE Aerospace Conference*, 1-10. https://doi.org/10.1109/aero.2012.6187425

Iguider, A., Bousselam, K., Elissati, O., Chami, M., & En-Nouaary, A. (2019). Embedded Sstems Hardware Software Partitioning Using Minimax Algorithm. *Proceedings of the 4th International Conference on Smart City Applications*, 1-6. https://doi.org/10.1145/3368756.3369009

Ismail, T. B., Abid, M., & Jerraya, A. (1994). COSMOS: A Codesign Approach for Communicating Systems. *Third International Workshop on Hardware/Software Codesign*, 17-24. https://doi.org/10.1109/hsc.1994.336727

Ismail, T. B., & Jerraya, A. A. (1995). Synthesis Steps and Design Models for Codesign. *Computer*, *28*(2), 44-53. https://doi.org/10.1109/2.347999

Jayakumar, N., & Khatri, S. P. (2004). A Metal and Via Maskset Programmable VLSI Design Methodology Using PLAs. *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.*, 590-594. https://doi.org/10.1109/iccad.2004.1382645

Karthik, S., Priyadarsini, K., & Jeanshilpa, V. (2018). Vlsi Systems for Simultaneous in Logic Simulation. *2018 International Conference on Recent Trends in Electrical, Control and Communication (RTECC)*, 23-27. https://doi.org/10.1109/rtecc.2018.8625665

Khezami, N., Kessentini, M., & Ferreira, T. D. N. (2021). A Systematic Literature Review on Software Maintenance for Cyber-Physical Systems. *IEEE Access*, *9*, 159858-159872. https://doi.org/10.1109/access.2021.3126681

Kitchenham, B. A., & Charters, S. (2007). *Guidelines for Performing Systematic Literature Reviews in Software Engineering*.

Lakshman, S. B., & Eisty, N. U. (2022). Software Engineering Approaches for TinyML Based IoT Embedded Vision. *Proceedings of the 4th International Workshop on Software Engineering Research and Practice for the IoT*, 33-40. https://doi.org/10.1145/3528227.3528569

Li, Y., Soliman, M., Avgeriou, P., & Somers, L. (2023). Self-Admitted Technical Debt in the Embedded Systems Industry: An Exploratory Case Study. *IEEE Transactions on Software Engineering*, *49*(4), 2545-2565. https://doi.org/10.1109/tse.2022.3224378

Liao, S., Tjiang, S., & Gupta, R. (1997). An Efficient Implementation Of Reactivity For Modeling Hardware In The Scenic Design Environment. *Proceedings of the 34th Design Automation Conference*, 70-75. https://doi.org/10.1109/dac.1997.597119

Lin, C.-W., & Chen, C.-H. (2017). A Processor and Cache Online Self-Testing Methodology for OS-Managed Platform. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *25*(8), 2346-2359. https://doi.org/10.1109/tvlsi.2017.2698506

Lora, M., Vinco, S., & Fummi, F. (2019). Translation, Abstraction and Integration for Effective Smart System Design. *IEEE Transactions on Computers*, *68*(10), 1525-1538. https://doi.org/10.1109/tc.2019.2909209

Nam, T.-J., & Lee, W. (2003). Integrating Hardware and Software: Augmented Reality Based Prototyping Method for Digital Products. *CHI '03 Extended Abstracts on Human Factors in Computer Systems - CHI '03*, 956-957. https://doi.org/10.1145/766090.766092

Olukotun, K. A., Helaihel, R., Levitt, J., & Ramirez, R. (1994). A Software-Hardware Cosynthesis Approach to Digital System Simulation. *IEEE Micro*, *14*(4), 48-58. https://doi.org/10.1109/40.296157

Ortega-Cabezas, P. M., Colmenar-Santos, A., Borge-Diez, D., & Blanes-Peiró, J. J. (2020). Application of Rule-Based Expert Systems in Hardware-in-the-Loop Simulation Case Study: Software and Performance Validation of an Engine Electronic Control Unit. *Journal of Software: Evolution and Process*, *32*(1), e2223. https://doi.org/10.1002/smr.2223

Raghunathan, V., Kansal, A., Hsu, J., Friedman, J., & Srivastava, M. (2005). Design Considerations for Solar Energy Harvesting Wireless Embedded Systems. *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, 457-462. https://doi.org/10.1109/ipsn.2005.1440973

Ronkainen, J., & Abrahamsson, P. (2003). Software Development under Stringent Hardware Constraints: Do Agile Methods Have a Chance? *Extreme Programming and Agile Processes in Software Engineering*, 73-79. https://doi.org/10.1007/3-540-44870-5_10

Ruchkin, V., Romanchuk, V., Fulin, V., Kostrov, B., & Ruchkina, E. (2015). Parallelism in Embedded Microprocessor Systems Based on Clustering. *2015 4th Mediterranean Conference on Embedded Computing (MECO)*, 45-50. https://doi.org/10.1109/meco.2015.7181951

Sabella, D., Serrano, P., Stea, G., Virdis, A., Tinnirello, I., Giuliano, F., Garlisi, D., Vlacheas, P., Demestichas, P., Foteinos, V., Bartzoudis, N., Payaro, M., & Medela, A. (2017). A Flexible and Reconfigurable 5G Networking Architecture Based on Context and Content Information. *2017 European Conference on Networks and Communications (EuCNC)*, 1-6. https://doi.org/10.1109/eucnc.2017.7980669

Sasirekha, G. V. K., Sai Venketesh, D., Adhisaya, T., Aswini, P., Bapat, J., & Das, D. (2019). Robotic Extension to IoT Testbed for Indoor Environment Supervision. *2019 Third International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, 29-35. https://doi.org/10.1109/i-smac47947.2019.9032498

Shen, M., Yang, W., Rong, G., & Shao, D. (2012). Applying agile Methods to Embedded Software Development: A Systematic Review. *2012 Second International Workshop on Software Engineering for Embedded Systems (SEES)*, 30-36. https://doi.org/10.1109/sees.2012.6225488

Srinivasan, V., Radhakrishnan, S., & Vemuri, R. (1998). Hardware Software partitioning with integrated hardware design space exploration. *Proceedings Design, Automation and Test in Europe*, 28-35. https://doi.org/10.1109/date.1998.655833

Srovnal, V., & Penhaker, M. (2007). Health Maintenance Embedded Systems in Home Care Applications. *Second International Conference on Systems (ICONS'07)*, 17-17. https://doi.org/10.1109/icons.2007.29

Vii, S., Kaur, H., Sharma, N., & Jain, A. (2019). A solitary approach to test path prioritization using significance of centrality measures. *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*, 1-6. https://doi.org/10.1109/iot-siu.2019.8777639

Vishwakarma, S. K., Upadhyaya, P., Kumari, B., & Mishra, A. K. (2019). Smart Energy Efficient Home Automation System Using IoT. *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*, 1-4. https://doi.org/10.1109/iot-siu.2019.8777607

von Knethen, A. (2002). Change-oriented requirements traceability. Support for evolution of embedded systems. *International Conference on Software Maintenance, 2002. Proceedings.*, 482-485. https://doi.org/10.1109/icsm.2002.1167808

Yousuf, S., & Gordon-Ross, A. (2016). An Automated Hardware/Software Co-Design Flow for Partially Reconfigurable FPGAs. *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 30-35. https://doi.org/10.1109/isvlsi.2016.73

Zeng, D., Gu, L., Guo, S., Cheng, Z., & Yu, S. (2016). Joint Optimization of Task Scheduling and Image Placement in Fog Computing Supported Software-Defined Embedded System. *IEEE Transactions on Computers*, *65*(12), 3702-3712. https://doi.org/10.1109/tc.2016.2536019

Zhang, P., Zambreno, J., & Jones, P. H. (2017). An embedded scalable linear model predictive hardware-based controller using ADMM. *2017 IEEE 28th International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, 176-183. https://doi.org/10.1109/asap.2017.7995276

Zuo, W., Pouchet, L.-N., Ayupov, A., Kim, T., Lin, C.-W., Shiraishi, S., & Chen, D. (2017). Accurate High-level Modeling and Automated Hardware/Software Co-design for Effective SoC Design Space Exploration. *Proceedings of the 54th Annual Design Automation Conference 2017*, 1-6. https://doi.org/10.1145/3061639.3062195