The Perception of Technical Debt in the Embedded Systems Domain: An Industrial Case Study

Areti Ampatzoglou[1], Apostolos Ampatzoglou[1], Alexander Chatzigeorgiou[2], Paris Avgeriou[1], Pekka Abrahamsson[3], Antonio Martini[4], Uwe Zdun[5], Kari Systa[6]

[1] Department of Computer Science, University of Groningen, Groningen, Netherlands
[2] Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece
[3] Department of Computer and Information Science, National Technical University of Norway, Trondheim, Norway
[4] Department of Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden
[5] Faculty of Computer Science, University of Vienna, Vienna, Austria
[6] Department of Software Systems, Technical University of Tampere, Tampere, Finland

areti.ampatzoglou@rug.nl, a.ampatzoglou@rug.nl, achat@uom.gr, paris@csd.auth.gr, pekkaa@ntnu.no, antonio.martini@chalmers.se, uwe.zdun@univie.ac.at, kari.systa@tut.fi

***Abstract*—Technical Debt Management (TDM) has drawn the attention of software industries during the last years, including embedded systems. However, we currently lack an overview of how practitioners from this application domain perceive technical debt. To this end, we conducted a multiple case study in the embedded systems industry, to investigate: (a) the expected lifetime of components that have TD, (b) the most frequently occurring types of TD in them, and (c) the significance of TD against run-time quality attributes. The case study was performed on seven embedded systems industries (telecommunications, printing, smart manufacturing, sensors, etc.) from five countries (Greece, Netherlands, Sweden, Austria, and Finland). The results of the case study suggest that: (a) maintainability is more seriously considered when the expected lifetime of components is larger than ten years; (b) the most frequent types of debt are test, architectural, and code debt; and (c) in embedded systems the run-time qualities are prioritized compared to design-time qualities that are usually associated with TD. The obtained results can be useful for both researchers and practitioners: the former can focus their research on the most industrially-relevant aspects of TD, whereas the latter can be informed about the most common types of TD and how to focus their TDM processes.**

Keywords—technical debt; embedded systems; industry; case study

I. INTRODUCTION

Embedded Software (ES), as a type of software targeting devices that is not typically thought of as computers, is usually specialized for a particular hardware and therefore has platform-specific run-time constraints (e.g., memory usage, processing power, etc.) [24]. In recent decades, ***software*** plays an ***increasingly important role*** in the development process of ***embedded products***: as suggested by Rauscher and Smith some embedded companies have increased the percentage of staff devoted to developing software in the organization—as much as 80% in some ***software-intensive domains*** [22]. A possible explanation for that is software's negligible replication cost and its greater flexibility compared to hardware, which makes it easier to change. Thus, product development managers often allow for some software additions or changes late in the product development cycle to address hardware problems or to add new functionality [22], ***resulting in intense maintenance activities***. An additional challenge in embedded systems is the ***long lifetime expectancy***, which is normally beyond a decade (see Section III.B), requiring the management of old systems in parallel to the design and implementation of new ones. Any decision to take advantage of novel hardware and software platforms means that companies need to manage many different configurations. Based on the above, it can be concluded that maintenance is an extremely challenging and costly activity in the ES domain.

To decrease the effort spent on maintenance, companies could invest in boosting design-time quality (e.g. maintainability) [3]. However, in most cases, companies trade-off design-time qualities in favor of business qualities such as product time-to-market. The relevance of this strategy for embedded software development has already been acknowledged in the literature: embedded software development is particularly challenging in the high-end technology sector, which is characterized by shortening product lifecycles, rising market fragmentation, and rapid technological changes [5], [18]. The aforementioned compromise between design-time qualities and business qualities, leads to the creation of a ***financial overhead*** in future maintenance activities, usually termed as ***technical debt*** (TD) [8]. It has been widely accepted that in an ***industrial context*** zero technical debt is not realistic, and probably not even desirable; the investment to reduce TD to zero would be extremely inefficient [10]. Thus, establishing Technical Debt Management (TDM) methods, tools and techniques are necessary to guide the (partial) technical debt repayment, the prioritization of refactoring opportunities, and the provision of design decisions that will improve the quality of software.

Although current methods, tools and techniques for monitoring and managing technical debt are continuously improving, they do not target specific application domains, like Embedded Systems. In particular, we expect that ES are in need of domain-specific methods, due to the special requirements of such software applications, e.g., hard software constraints, the

need to guarantee run-time qualities, etc. A first step toward the development of domain-specific methods, tools and techniques is the in-depth understanding of TD in the domain under consideration. To this end, in this paper we perform an ***exploratory case study*** that aims at understanding ***how TD is perceived*** in the ES industries. This understanding can support steering research in technical debt management as well as raising awareness among ES practitioners on technical debt-related issues.

The rest of the paper is organized as follows: In Section II, we present related work on studies that report on TDM in an industrial context and studies that report on the important quality attributes for embedded software systems development. In Section III, we report on the case study design, whereas in Section IV we present our results, organized by research question and accompanied with possible interpretations. The results are discussed in Section V, with respect to the implications to researchers and practitioners that they provide. Finally, in Section VI we report threats to validity and mitigation actions, whereas in Section VII, we conclude the paper.

II. Related Work

We have identified a few studies on investigating the accumulation and management of technical debt in industry. Ernst et al. [11] conducted a survey to understand which Technical Debt type is more frequent among practitioners. The results show that architectural debt is the one that is mostly mentioned by practitioners. Although 31% of the respondents belong to the embedded software domain, the paper does not distinguish results with respect to the application domain. Ernst et al. have also found that the main challenges so far are the lack of systematic practices, and the inadequateness of the tools (too complex to install or to interpret and too many false positives), while another major issue is the difficulty of increasing the awareness of TD for stakeholders without software engineering expertise; the latter is also recognized by both Lim et al. [17] and Martini et al. [20]. According to Lim et al., [17] measuring TD is difficult for practitioners, because the impact is not uniform. Codabux et al. [7] mention that the repayment of TD is highly influenced by the customer needs and they also discuss the fact that practitioners decide to take on debt in order to achieve their short-term objectives, although they cannot be fully aware of the potential serious long-term consequences. Martini and Bosch [19] highlight how TD repayment can be prioritized in an industrial context, by discussing with practitioners. The major difference of these studies, compared to ours is that we focus our investigation on the embedded software domain.

In addition to studies that focus on TD in industry, as related work, we discuss papers that provide insights on the important quality attributes (QAs) in the embedded systems domain. This kind of work is pertinent as TD represents in its core a tradeoff between maintainability and other qualities. Concerning the interplay between QAs in the domain of embedded systems, Del Rosso presents an architectural approach for improving the performance of software products, derived from a product family for real-time embedded systems, and its possible implications to maintainability [9]. The results suggest that by analyzing the commonalities and differences among derived products, one can extract bottlenecks and problems in the core architecture (e.g., a God class). In a similar context, Oliveira et al. investigate the relationship between non-critical quality attributes, measured by metrics obtained from source code, and performance, measured by physical metrics (i.e., memory, time, and energy) obtained from run-time monitoring [21]. The results indicate the existence of trade-offs between quality and physical metrics, as well as the fact that quality metrics can provide information regarding high-level quality attributes, guiding the design solution selection at early stages, which might lead to significant gain in physical characteristics later on. Finally, Feitosa et al. [12] analyze the difference in the trade-offs among quality attributes in critical and non-critical embedded software. The quality attributes of interest were: correctness, performance, security, reusability, understandability, functionality, extendibility, effectiveness, and flexibility. Based on the results of this study, the authors provide some hints that understandability has an inverse relationship with all the other qualities. Compared to the aforementioned studies, we emphasize on (the lack of) maintainability, which can be considered a factor for the accumulated TD.

III. Case Study Design

To explore the perception of industrial practitioners on technical debt, we have performed an exploratory case study on seven embedded systems companies (see Section III-B). The main reason for conducting this case study is that domain-specific TDM requires a detailed understanding of how the TD phenomenon is perceived in software-intensive industries. In this section we describe the case study design, according to the guidelines proposed by Runeson et al. [23].

A. Objective and Research Questions

The goal of this study, described using the Goal-Question-Metric (GQM) formulation [4], is: "to ***analyze*** the perception of technical debt in the embedded systems industry ***for the purpose of*** understanding ***with respect to***: (a) the expected lifetime of components that have TD, (b) the types of technical debt that are frequently occurring, and (c) the significance of other quality attributes ***from the point of view of*** software engineers, ***in the context of*** embedded software development".

By taking into account the inherent relationship of maintainability and technical debt [15] in this study we consider maintainability as a proxy for TD. We have chosen this proxy, since the term TD is not interpreted unambiguously in industry: practitioners may mean different things when discussing TD and related concepts like principal and interest. On the other hand, maintainability, though not associated to a universally accepted definition, is widely accepted as the ability to make changes in a system. Moreover, even when a practitioner is not aware of incurred TD, any decisions aimed at enhancing maintainability, will result in a lower amount of TD. On the other end, if a development team is not interested in producing a maintainable system, then it is highly probable that shortcuts will be made, leading to the accumulation of technical debt. Thus we have used maintainability as the proxy for RQ_1 and RQ_3; this proxy is discussed further in the threats to validity (see Section VI).

Based on the abovementioned goal, we have extracted three research questions (RQs):

RQ$_1$: *What is the relationship between the expected lifetime of components and technical debt?*

This research question is considered important, in the sense that managing technical debt is important for components, whose accumulated interest, at some point becomes larger than the principal [3][6]. In other words, technical debt that is accumulated on components that are not maintained for a large time period is less probable to be harmful than for components with a high expected lifetime. This aspect becomes even more important for embedded systems, since their expected lifetime is usually long. For this research question we will study the relationship between the focus of development teams on maintainability (as a proxy for TD) and the expected lifetime of the software. In other words, we will explore if software engineers focus on maintainability (potentially managing TD) when the system under development is expected to be active for many years.

RQ$_2$: *What types of technical debt (e.g., code, architectural, etc.) are more frequently occurring in embedded systems?*

According to Li et al. [16], technical debt can be identified in all phases of software development. In particular, Li et al. have identified 10 types of TD (see Section III.C for more details). Each one of these TD types requires different technical debt management approaches. Therefore, understanding which the most frequent ones are can support prioritization and monitoring of TD.

RQ$_3$: *What is the significance of building maintainable software systems (with low TD) compared to satisfying other quality attributes?*

In embedded system industries, run-time constraints are very important for software development. Therefore, trade-offs between run-time and design-time qualities are a common practice [12]. In this research question, we examine which quality attributes are taken into account in embedded software development, and how these are prioritized. In particular we aim at comparing the importance given to maintainability (the QA that is used as a proxy for TD) against the other quality attributes. By answering this research question we gain insight into which quality attributes are prioritized in software development, and which ones are negotiable in the ES domain.

B. Case Selection

Cases and Units of Analysis. According to Runeson et al. [23], case studies can be characterized either as holistic or embedded, based on the way cases and units of analysis are defined. This study is an embedded multiple case study, because we investigate multiple units of analysis (i.e., components in which technical debt has been identified) extracted from multiple embedded systems industries (i.e., cases).

Case Selection. In this study, we consider as cases embedded systems companies, specializing in different application domains, and located in different European countries. A brief description of the companies is presented below and is summarized in Table I. The companies have been anonymized due to confidentiality reasons.

TABLE I. EXPLORED EMBEDDED INDUSTRIES

ID	Company Description			
	Application Domain	***Country***	***Type***	***#Analyzed Components***
C1	Telecommunications	Sweden	Large	1
C2	Automotive	Sweden	Large	1
C3	Mobile	Greece	SME	7
C4	Sensors	Greece	SME	3
C5	Printing	Netherlands	Large	1
C6	Smart Manufacturing	Austria	Large	6
C7	Media Devices	Finland	SME	1

From the data of Table I, we can observe that in our dataset we have selected a balanced mix of *Small/Medium Enterprises* (SMEs) and *Large Enterprises*. The last column of Table I, presents the number of units of analysis (i.e., analyzed components) that have been provided by each company. In total, 20 units of analysis have been extracted and analyzed. The components from each company and some challenges and interests of the companies are briefly described as follows:

- ***Company C1*** contributed a telecom component that has been in use for 50 releases and is expected to be maintained for approximately 30 more years. For C1, compatibility, functional suitability, and reliability are having the highest priority so far. The major challenges that the company faces while repaying TD are: (a) the prioritization of feature development before repaying debt, and (b) the difficulty in smoothly planning refactorings. Developers are mostly interested in test and architecture TD.
- ***Company C2*** contributed an automotive component that has already been released 30 times from 2013, and therefore is in need of intense maintenance activities. The expected lifespan for this component is 15 more years. While developing and maintaining this component, C2 is interested in ensuring functional suitability, reliability, and security. The company is interested in working on feature coverage test, pass rates of test cases, and prioritization of features during development.
- ***Company C3*** contributed seven mobile components, which are of various levels of maturity (1-21 releases so far), which have an infinite expected/desired lifespan (i.e., as far as the customer wants to use them). Although maintainability is of great interest to the company, since their embedded products might need to survive for a long period, so far the company is mostly focused on other quality attributes (i.e., usability, functional suitability, and performance). The form of TD that the company wants to repay is within the software architecture, source code, testing, documentation, and underlying infrastructure.
- ***Company C4*** contributed three components (i.e., "*Monitoring and processing*", "*Sensor Configurator*", and "*Reporting tool*"). The system is a rather "young" project, i.e., half-year, and is expected to be used from the company for 5 more years. Although maintainability is characterized as important from the company, other quality at-

tributes, like security, functional suitability and reliability are getting more priority. The company is interested in managing architectural, source code, test, and build TD.

- ***Company C5*** contributed the "*Printing Data Path (PDM)*" component, which has been used from the company for almost ten years, and its expected lifetime is 5 additional years. PDM is a highly reusable component that needs to be adapted to different printers' hardware. Compared to the previous years, C5 is interested in putting extra effort in repaying technical debt related to testing, documentation, and design. At the same time, the functional suitability, reliability and performance of PDM are not negotiable.
- ***Company C6*** contributed the "*Product Status Sensor*", the "*Weather Sensing*", the "*Product Data*", and the "*Data Aggregator*" components, and two components of a "*Smart Product Maintenance*" application. The expected lifetime of the aforementioned components is between 10 – 20 years, so their easy extension and maintenance is crucial to the company. Based on the current knowledge on the quality key-drivers for these components, functional suitability, reliability and performance should be kept at the highest level while managing technical debt.
- ***Company C7*** contributed a software component used for home media server applications. The component includes a platform and an SDK for application development. The software has been developed since early 2000 and during the years it has evolved significantly. The expected lifespan of the project is 20 years from now. Since the area of home automation is new and there are many players in the market, the system should be compatible with a huge number of external systems and different installations connected to different sets of home automation systems (high portability and compatibility are required). Rapid implementation of the support for these systems may create technical debt.

C. Data Collection

Every company (case) has been asked to provide us with a number of components that have accumulated technical debt and are difficult to maintain. Next, for each unit of analysis the following information has been recorded:

Project Demographics

[V1] Textual ***description of project***, in which the TDI has been identified.

[V2] First ***release date*** of the project. This variable aims at expressing how old the component is.

[V3] ***Number of releases*** until now. This variable provides an indication on how frequent the maintenance of this component is.

Technical Debt Item Data Points

[V4] ***Estimated lifespan*** of the TDI. This variable represents how significant it is to eliminate the TD from the component, since it denotes the time period during which the company is paying interest. Due to the low number of units of analysis, we preferred to recode this variable in a categorical one, so that each data class has more members. Therefore, we recode values less than 10 years to "*short-term projects*" and values between 10 and 30 years to "*long-term projects*". The selection of the threshold of 10 years was made since it is the median of our sample, and a decade is considered as a milestone in terms of time periods.

[V5] ***Types of debt identified***. In this variable we record what types of TD are identified in the selected component. The types have been extracted from the work of Li et al. [16], summarized as follows: *Requirements, Architecture, Design, Code, Test, Build, Documentation, Infrastructure, and Versioning*. We note that from the list provided by Li et al. [16], we have removed Defect Debt, since the TD community does not consider the existence of defects as TD principal [15].

[V6] ***Importance of quality attributes*** along TDI evolution. A rating, in a scale from 1(lowest) to 5 (highest), of the importance of the following QAs: *Functional suitability, Reliability, Performance, Usability, Security, Compatibility, Maintainability*, and *Portability*. The quality attributes have been selected so as to represent both design-time and run-time quality attributes.

The data collection has been performed through interviews with software engineers (i.e., designers, architects, project managers) during which data have been recorded by the researchers (a supervised questionnaire-based approach [14]). The use of supervised data collection methods ensures the understanding of the data collection instrument from the interviewees and strengthens validity. Nevertheless, the use of a supervised data collection method has restricted the size of our dataset to 20 units of analysis (obtained by 7 companies), since it was time consuming. We note that variables [V4] – [V6] have been answered by practitioners without the use of any documentation, since: [V4] represents an estimation of the respondent, in none of the companies the technical debt items ([V5]) were documented, and [V6] represents the importance of several QAs according to experts' opinion.

D. Data Analysis

To answer the research questions set in section III.A, we have performed descriptive statistical analysis and hypothesis testing. The analysis plan per research question is presented in Table II. More specifically, concerning RQ_1, we used descriptive statistics on the expected lifespan of components that carry technical debt. In addition, we explored the correlation between expected life-time and maintainability tasks to unveil a possible relationship, by performing a chi-square test. Regarding RQ_2, we report frequencies for the different types of technical debt of the highest level, as reported by Li et al. [16]. Furthermore, we present details on how industries perceive technical debt types, by synthesizing the obtained results.

Finally, concerning RQ_3, we analyzed the data in three ways: (a) we present descriptive statistics on the importance of each quality attribute separately; (b) we inspected if there are differences among the importance of quality attributes; and (c)

we performed Wilcoxon Ranks Test between QAs and maintainability (the attribute that is considered as a proxy to technical debt). Using the above, have been able to spot differences in the way different quality attributes are prioritized. Such differences can lead to intentional or unintentional trade-offs. In particular, QA that receive higher priority than maintainability are expected to be non-negotiable while managing TD.

TABLE II. DATA ANALYSIS PLAN

RQ	Analysis Plan	
	Used Variables	***Analysis***
1	V4 (Estimated Lifespan) V6 (Importance of Maintainability)	Descriptive statistics Cross-Tabulation chi-square test
2	V5 (Types of TD)	Descriptive statistics
3	V6 (Importance of QAs)	Descriptive statistics Wilcoxon Signed Rank

IV. RESULTS

In this section we present the results organized by research question. In each section, we first answer the questions using statistical analysis, and then provide a possible interpretation of the obtained results.

A. *RQ1: Technical Debt and ES Expected Lifetime*

To answer RQ_1 we have performed cross tabulation to examine the relationship between the projects' lifetime expected duration and the attention given by practitioners to maintainability. The dataset consists of components with expected lifetime that varies from less than one to 30 years. The mean expected lifetime of the components is 12.40 years, with a standard deviation of 8.94, and a median of 10 years. As mentioned above (Section III.C), we have added a categorical variable that separates the dataset into two groups. The first one contains components that are expected to last for less than ten years (*short-term projects*), and the second one consists of components with an estimated lifecycle between 10 and 30 years (*long-term projects*). The analysis suggests that 45% of the components belong to the *short-term projects* category, while 55% of the components belong to the *long-term projects* one.

The results of the cross tabulation suggest that 66.7% (6 out of 9) of the components with long estimated lifecycle receive high attention in terms of maintainability (scores '*high*' and '*very high*'). On the other hand, practitioners do not consider maintainability issues important for the majority of the short-term components, as 54.6% (6 out of 11) of them have low maintainability importance scores ('*very low*' or '*low*'). The results of the cross tabulation are presented in Table III. Each column of the table represents the level of importance that practitioners have assigned to maintainability, whereas each line counts the observed and the expected number of long-term or short-term components that correspond to each level. The observed counts represent the frequencies based on experts' opinion, whereas the expected counts, represents the frequencies that would be expected based on data distribution. The closer the observed and expected counts are, the more unrelated the two variables are [13], in the sense that the frequencies (levels of maintainability) follow the expected distribution, regardless of the values of the grouping variable (expected lifespan of the project).

TABLE III. RELATION BETWEEN ESTIMATED LIFETIME & MAINTAINABILITY

Estimated Lifespan		Maintainability				
		very low	**Low**	**neutral**	**high**	**very high**
Long	Observed Count	0,0	0,0	3,0	5,0	1,0
	Expected Count	1,4	1,4	1,4	4,5	0,5
Short	Observed Count	3,0	3,0	0,0	5,0	0,0
	Expected Count	1,7	1,7	1,7	5,5	0,6

To objectively assess this relationship, we have performed a chi-square (x^2) test. The x^2 test revealed that there is a statistically significant relationship ($x^2 = 9.89$ and sig. $= 0.042 < 0.05$). The aforementioned relationship between the expected lifetime of the project and the importance of maintainability is intuitive, since software engineers are not expected to put extra effort (representing the principal of technical debt [3]) on the development of applications that will not be maintained for long periods. This observation suggests that short-term projects are expected to be developed with more design-time compromises, and therefore with more accumulated technical debt.

The technical debt management activities (TD repayment, prevention, etc.) are expected to be more relevant for projects for which long-term maintenance periods are anticipated.

B. *RQ2: Types of Technical Debt*

In order to detect the most frequently identified types of technical debt in embedded systems industries, we have analyzed the data on the types of TD reported by practitioners and we have performed descriptive statistical analysis. Figure 1 presents a bar chart on the frequency of types of technical debt on the investigated industrial components (as a percentage).

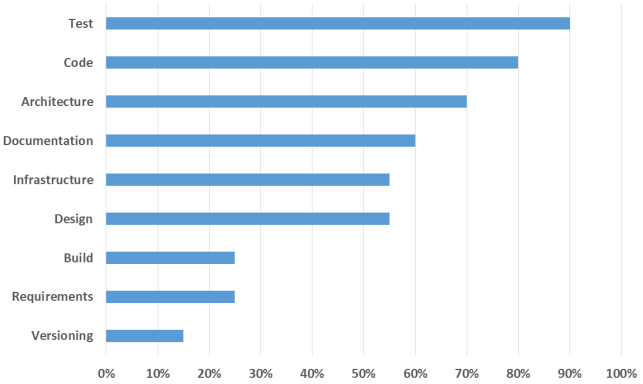

Fig. 1. Types of Debt Frequencies

As shown in Fig. 1, the most frequently identified TD types are ***test*** debt (e.g., limited number of unit tests, lack of test automation, etc.) which is identified in 90% of the components. It is followed by ***code*** debt (e.g., duplicate code, long methods, etc.) and ***architectural*** debt (e.g., anti-patterns, best practice violations, etc.), reported in 80% and 70% of the cases respec-

tively. Next, ***documentation*** (e.g., outdated documentation), ***design*** (e.g., grime, design principles violation) and ***infrastructure*** (e.g., old technology in use) debt are found in many cases (55-60%). ***Requirements*** (e.g., over-engineering), ***build*** (e.g., manual build process) and ***versioning*** (e.g., multi-version support) debt seem to be the least commonly identified types of technical debt. This fact suggest that the most "tangible" and easily detectable types of technical debt (e.g., duplicate code, test case, and design artifacts) are more easily understood and reported by practitioners. A more fine grained analysis of the most frequent types of technical debt is presented in Table IV. The terms presented in Table IV have been reused from the TD types reported by Li et al. [16]. In addition, from the results of Table IV, we can observe that two different subcategories of types of TD exist: (a) the types of debt that exist in the system (e.g., duplicate code) and therefore is more easily quantifiable and its potential impact can be estimated, and (b) the types of debt that refer to what is missing from the system (e.g., lack of test cases) for which only estimations can be performed.

TABLE IV. FREQUENT TYPES OF TECHNICAL DEBT

Type of Technical Debt Item	Frequency
Duplicate code	6
Limited number of unit tests	6
Complex code	3
Old technology in use	3
Lack of automated deployment	3
Violations of good architectural practices	2
Lack of test automation	2

The aforementioned results are in accordance to those of Alves et al. [1] who are reporting on the most frequently studied types of technical debt in the research community. In particular, according to Alves et al. design, architecture, and documentation debt are the most frequently studied types in the TD literature. The fact that our results are in accordance with those of Alves et al. suggests that until now the research directions are quite accurately focused on the industrial needs. The only possible exceptions are design debt, which does not seem to be as important for practitioners as it is for researchers, and test and code debt, which does not appear to be appealing to researchers. Apart from industrial relevance, an additional parameter that makes research on these topics popular is the existence of tools for testing, code, and design quality assurance.

The most recurring types of technical debt in industry are test, architectural design, and source code debt. Architectural and design debt are among the most frequently studied by researchers, as well. On the other hand, some types of TD (e.g., test, code, and infrastructure), which are interesting for practitioners, are understudied by the research community.

C. *RQ$_3$: Technical Debt and Quality Attributes*

To investigate the relationship between quality attributes (other than maintainability) and technical debt, we compare the importance that is assigned to them by practitioners. The importance of each quality attribute is presented in Figure 2. A quality attribute is considered more important if the most frequent answers are concentrated in '*high*' and '*very high*' scales. For example, functional suitability is considered of great importance since there was no component for which its importance was characterized as 'neutral', 'low', or '*very low*'. On the other hand, security and portability have been characterized as '*low*' or '*very low*' importance for approximately 50-55% of the investigated components.

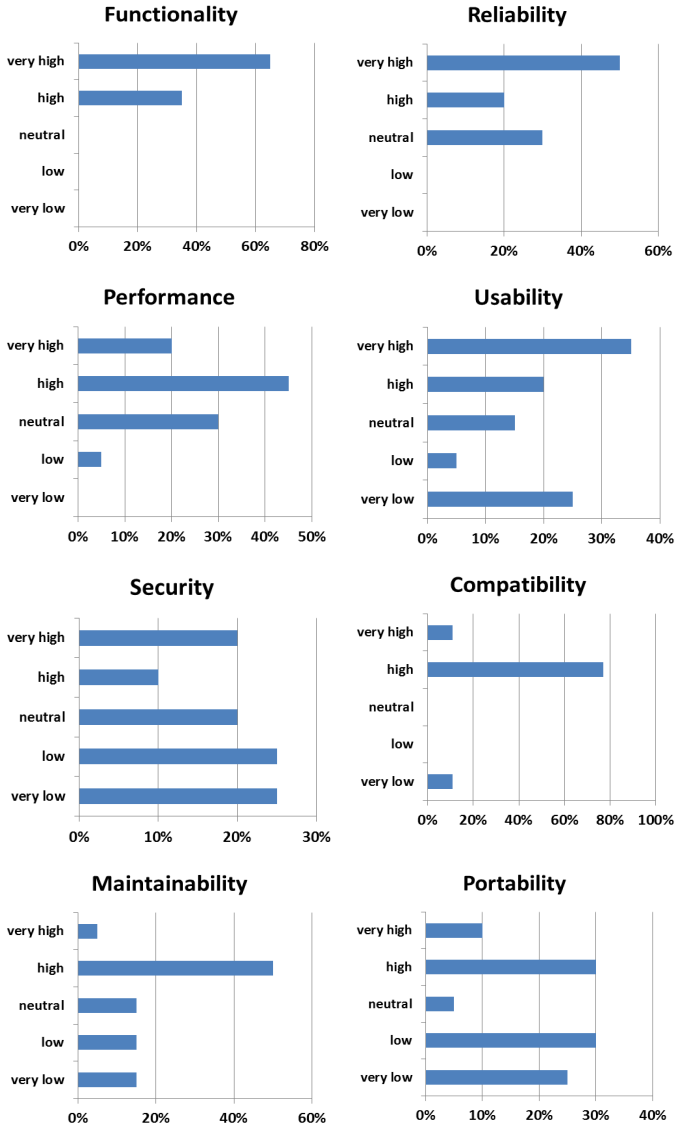

Fig. 2. Importance of Quality Attributes

We performed Wilcoxon Signed Rank tests to investigate the frequency with which each QA is given more / less priority than maintainability (which is considered as a proxy for technical debt). The results are presented in Table V. In particular, the first column of the table represents the QAs, the second and third column depict if the quality attribute under discussion is getting higher priority against maintainability or not, and the last column stands for the p-value for the test (values lower than 0.05 denote a statistically significant difference). The QAs

that receive higher importance/priority against maintainability ($p<0.05$) are denoted with light grey cell shading.

TABLE V. PRIORITY OF QAS VS. TECHNICAL DEBT

Quality Attribute	Most Prioritized	N	Sig.
Functionality	Functionality	19	0,000
	Maintainability	0	
	Ties	1	
Reliability	Reliability	16	0,001
	Maintainability	2	
	Ties	2	
Performance	Performance	9	0,014
	Maintainability	2	
	Ties	9	
Usability	Usability	11	0,738
	Maintainability	6	
	Ties	3	
Security	Security	4	0,125
	Maintainability	8	
	Ties	8	
Compatibility	Compatibility	2	0,705
	Maintainability	2	
	Ties	5	
Portability	Portability	3	0,104
	Maintainability	9	
	Ties	8	

From the results of Table V we can observe that ***functionality***, ***reliability***, and ***performance*** are given more importance than maintainability along ES development. Similarly, usability is also usually prioritized against maintainability, but this result is not statistically significant. On the other hand, for the selected systems ***security***, ***compatibility***, and ***portability*** are not the usual quality key drivers. The importance of performance in embedded systems can be justified by the limited resources (e.g., memory, execution time, energy efficiency, etc.) that are usually available in embedded systems (e.g., printers, sensors, telecommunication devices, etc.). Additionally, reliability is important for special types of embedded software, critical embedded systems (e.g., automotive, telecommunications, etc.). Finally, the functional suitability of the system cannot be negotiated, since a system that does not conform to all of its requirements, often dictated by standards and regulations, is not ready for reaching the market. The importance of the specific three QAs for CES is also discussed by Feitosa et al. [12].

Regarding the rest of the QAs, security does not seem to be the first priority of software developers, since none of the studied application domains is vulnerable to external attacks (e.g., through the internet). Furthermore, compatibility and portability seem to be important only for mobile applications, due to the needs for responsive user interfaces (i.e., for devices with different screen sizes—tablets, mobile phones, laptop, etc.) and different operating systems (e.g., Android, iPhones, etc.).

While managing technical debt in embedded software, some run-time quality attributes are given higher priority than maintainability. Specifically, the ES domain prioritizes reliability, functionality, and performance against maintainability.

V. IMPLICATIONS TO RESEARCHERS AND PRACTITIONERS

The obtained results are expected to be useful for both researchers and practitioners, since the former can focus their research on the most industrially-relevant aspects of technical debt, whereas the latter can increase their self-awareness on technical debt and target the technical debt management actions inside their companies. In particular, regarding researchers, we suggest that:

- Technical debt research is in need of more methods on the ***test, code*** and ***infrastructure*** level, since these fields are identified as problematic from the practitioners, but are understudied in the literature [1].
- ***Technical debt prioritization*** [16] methods should consider not only the urgency of resolving technical debt items, but also the ***feature prioritization*** of the company (see, e.g., the backlog in [15]).
- The development of tools and methods for ***TD prevention*** [16] is necessary, so that practitioners can timely manage the accumulated TD, while developing the software.
- The development of ***domain-specific tools and methods*** that take into account the specific requirements of given fields (e.g., run-time constraint or quality trade-offs for embedded systems) and can analyze other languages that are application domain specific (e.g., php for web, or JavaScript for mobile development).

On the other hand, the results suggest that practitioners are acknowledging the significance of ***TD management***, especially ***for long-term projects***. On the contrary, if there is no intense maintenance, the accumulated interest may be lower than the principal (see [2] and [6]). In such cases, having accumulated TD is not necessarily harmful. Additionally, the results of this study (in particular RQ_2) can provide guidance to practitioners on which TD types are the most common in the embedded systems domain (see Table IV). Based on their frequency, practitioners should be vigilant and monitor effectively the corresponding TD types. Finally, based on the importance of specific run-time QAs (see Table V) for embedded systems, we advise practitioners to carefully select TD repayment methods, favoring those that do not harm these run-time qualities.

VI. THREATS TO VALIDITY

In this section we discuss the threats to validity of this study. First, since this case study has been performed on 20 components from 7 embedded system industries, the findings and conclusions are subject to external validity threats, in the sense that using a different sample might reveal a different perception of technical debt among embedded software devel-

opers. However, the fact that the analyzed industries have different sizes and application domains provides a certain level of confidence on the obtained landscape of technical debt in ES.

The first research question on the relation between the expected lifetime and TD might suffer from construct validity threats, as the acknowledgment of maintainability as an important factor for long-lived products does not necessarily imply attention to TDM. The same holds for the third research question investigating the prioritization of TD against quality attributes; since maintainability is well-understood, but is at best only a proxy for TD, one cannot claim that the trade-off between maintainability and run-time quality attributes is directly transferrable to the notion of technical debt. Nevertheless, we consider this choice as justified since: (a) the direct use of the term TD might cause misinterpretation among practitioners (leading to a different threat to validity) and (b) the relation of TD and maintainability is established in the community [15]. Finally, an additional threat to construct validity is the possible misinterpretation on the quality attributes by practitioners while filling in the questionnaires. However, we believe that this threat is mitigated since: (a) the data extraction method was a supervised questionnaire-based one, and therefore any possible misinterpretation have been handled by the researcher who was conducting any interview, and (b) we have chosen well established and understandable QAs.

VII. CONCLUSIONS

The technical debt metaphor can prove to be a suitable means for conveying the importance of maintainability to the developers of rapidly evolving software domains, such as the embedded systems. Before any targeted research activities are undertaken to support the management of TD, it is important to understand the perception of the TD concept among ES developers. To this end, in this paper, we have performed a case study involving seven ES industries to investigate the most frequently occurring types of TD, the quality attributes usually associated with TD items, and the relation between the expected lifetime of a component and the acknowledgement of the importance of maintainability. The findings indicate that the most recurring types of technical debt in embedded software industry are test, architectural, and code debt. At the same time, some quality attributes such as functionality, reliability, and performance are given higher priority compared to managing technical debt. Finally, developers clearly acknowledge the need for low technical debt on components that are expected to have a longer lifetime, compared to more short-lived ones.

REFERENCES

[1] N. S. Alves, T. S. Mendes, M. G. de Mendonça, R. O. Spínola, F. Shull, and C. Seaman, "Identification and management of technical debt: A systematic mapping study," Information and Software Technology, vol. 70, pp.100–121, 2016.

[2] A. Ampatzoglou, A. Ampatzoglou, P. Avgeriou, and A. Chatzigeorgiou, "A Financial Approach for Managing Interest in Technical Debt", LNBIP, Springer, vol. 257, pp. 117-133, 2016.

[3] A. Ampatzoglou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "The financial aspect of managing technical debt: A systematic literature review," Information and Software Technology, Elsevier, vol. 64, pp. 52–73, Aug. 2015.

[4] V. R. Basili, G. Caldiera, and H. D. Rombach, "Goal Question Metric Paradigm", Encyclopedia of Software Engineering, John Wiley, 1994.

[5] D. J. Bowersox, T. P. Stank, and P. J. Daugherty, "Lean launch: managing product introduction risk through response-based logistics," Journal of Product Innovation Management, vol. 16, no. 6, pp. 557–568, Nov. 1999.

[6] A. Chatzigeorgiou, A. Ampatzoglou, A. Ampatzoglou, and T. Amanatidis, "Estimating the Breaking Point for Technical Debt", 7th International Workshop on Managing Technical Debt (MTD '15), IEEE Computer Society, 2015.

[7] Z. Codabux and B. Williams, "Managing technical debt: An industrial case study," 4th International Workshop on Managing Technical Debt (MTD' 13), 2013, pp. 8–15.

[8] W. Cunningham, "The WyCash Portfolio Management System," Proceedings on Object-oriented Programming Systems, Languages, and Applications (Addendum), New York, NY, USA, 1992, pp. 29–30.

[9] C. Del Rosso, "Software performance tuning of software product family architectures: Two case studies in the real-time embedded systems domain," Journal of Systems and Software, Elsevier, vol. 81(1) pp. 1–19, Jan. 2008.

[10] R. Eisenberg, "Management of Technical Debt: A Lockheed Martin Experience Report," 3rd International Workshop on Managing Technical Debt (MTD' 13), Baltimore, USA, 2013.

[11] N. A. Ernst, S. Bellomo, I. Ozkaya, R. L. Nord, and I. Gorton, "Measure It? Manage It? Ignore It? Software Practitioners and Technical Debt," 10th Joint Meeting on Foundations of Software Engineering, New York, NY, USA, 2015, pp. 50–60, 2015.

[12] D. Feitosa, A. Ampatzoglou, P. Avgeriou, and E. Y. Nakagawa, "Investigating Quality Trade-offs in Open Source Critical Embedded Systems", Quality of Software Architectures (QoSA' 15), 2015.

[13] A. Field, "Discovering Statistics using IBM SPSS Statistics", SAGE Publications Ltd., 2013.

[14] B. Kitchenham and S. L. Pfleeger, "Principles of Survey Research Part 2: Designing a survey", Special Interest Group on Software, ACM, 27 (1), pp. 18-20, January 2002.

[15] P. Kruchten, R. L. Nord, I. Ozkaya, "Technical Debt: From Metaphor to Theory and Practice", Software, IEEE Computer Society, vol. 29, no. 6, pp. 18-21, November-December 2012

[16] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," Journal of Systems and Software, Elsevier, vol. 101, pp. 193–220, March 2015.

[17] E. Lim, N. Taksande, and C. Seaman, "A Balancing Act: What Software Practitioners Have to Say About Technical Debt," Software, IEEE Computer Society, vol. 29, no. 6, pp. 22–27, Nov. 2012.

[18] D. N. Mallick and R. G. Schroeder, "An Integrated Framework for Measuring Product Development Performance in High Technology Industries," Production Operation and Management, vol. 14, no. 2, pp. 142–158, Jun. 2005.

[19] A. Martini and J. Bosch, "Towards prioritizing Architecture Technical Debt: information needs of architects and product owners," 41st Euromicro SEAA Conference, Funchal, Madeira, August 2015.

[20] A. Martini, J. Bosch, and M. Chaudron, "Investigating Architectural Technical Debt Accumulation and Refactoring over Time: a Multiple-Case Study," Information and Software Technology, July 2015.

[21] M.F.S. Oliveira, R.M. Redin, L. Carro, L. Lamb and F. Wagner, "Software Quality Metrics and their Impact on Embedded Software," 5th International Workshop on Model-based Methodologies for Pervasive and Embedded Software, (MOMPES'08), pp. 68–77, 2008.

[22] T. G. Rauscher and P. G. Smith, "From experience time-driven development of software in manufactured goods," Journal of Production and Innovation Management, vol. 12, no. 3, pp. 186–199, Jun. 1995.

[23] P. Runeson, M. Host, A. Rainer, and B. Regnell, "Case Study Research in Software Engineering: Guidelines and Examples", John Wiley & Sons, 2012.

[24] B. Stroustrup, "Abstraction and the C++ Machine Model," Embedded Software and Systems, Z. Wu, C. Chen, M. Guo, and J. Bu, Eds. Springer Berlin Heidelberg, 2005, pp. 1–13.