Noname manuscript No.
(will be inserted by the editor)

SDK4ED - A Platform for building Energy Efficient, Dependable, and Maintainable Embedded Software

Miltiadis Siavvas* · Dimitrios Tsoukalas · Charalambos Marantos · Lazaros Papadopoulos · Christos Lamprakos · Oliviu Matei · Christos Strydis · Muhammad Ali Siddiqi · Philippe Chrobocinski · Katarzyna Filus · Joanna Domańska · Paris Avgeriou · Apostolos Ampatzoglou · Dimitrios Soudris · Alexander Chatzigeorgiou · Erol Gelenbe · Dionysios Kehagias · Dimitrios Tzovaras

Received: date / Accepted: date

Miltiadis Siavvas
Centre for Research and Technology Hellas, Thessaloniki, Greece
*Corresponding Author
E-mail: siavvasm@iti.gr

Dimitrios Tsoukalas
Centre for Research and Technology Hellas, Thessaloniki, Greece
E-mail: tsoukj@iti.gr

Charalampos Marantos
School of Electrical and Computer Engineering, National Technical University of Athens, Athens, Greece
E-mail: hmarantos@microlab.ntua.gr

Lazaros Papadopoulos
School of Electrical and Computer Engineering, National Technical University of Athens
E-mail: lpapadop@microlab.ntua.gr

Christos Lamprakos
School of Electrical and Computer Engineering, National Technical University of Athens
E-mail: cplamprakos@microlab.ntua.gr

Oliviu Matei
R&D Department, Holisun SRL, Baia Mare, Romania
E-mail: oliviu.matei@holisun.com

Christos Strydis
Neuroscience Department, Erasmus Medical Center, Rotterdam, The Netherlands
E-mail: c.strydis@erasmusmc.nl

Muhammad Ali Siddiqi
Neuroscience Department, Erasmus Medical Center, Rotterdam, The Netherlands
E-mail: m.siddiqi@erasmusmc.nl

Philippe Chrobocinski
AIRBUS Defence and Space, France

Abstract Developing embedded software applications is a challenging task, chiefly due to the limitations that are imposed by the hardware devices or platforms on which they operate, as well as due to the heterogeneous non-functional requirements that they need to exhibit. Modern embedded systems need to be energy efficient and dependable, whereas their maintenance costs should be minimized, in order to ensure the success and longevity of their application. Being able to build embedded software that satisfies the imposed hardware limitations, while maintaining high quality with respect to critical non-functional requirements is a difficult task that requires proper assistance. To this end, in the present paper, we present the SDK4ED Platform, which facilitates the development of embedded software that exhibits high quality with respect to important quality attributes, with a main focus on energy consumption, dependability, and maintainability. This is achieved through the provision of state-of-the-art and novel quality attribute-specific monitoring and optimization mechanisms, as well as through a novel fuzzy multi-criteria decision-making mechanism for facilitating the selection of code refactorings, which is based on trade-off analysis among the three main attributes of choice. Novel forecasting techniques are also proposed to further support decision making during the development of embedded software. The usefulness, practicality, and industrial relevance of the SDK4ED platform were evaluated

E-mail: philippe.chrobocinski@airbus.com

Katarzyna Filus
Institute of Theoretical & Applied Informatics, Polish Academy of Sciences, Gliwice, Poland
E-mail: kfilus@iitis.pl

Joanna Domańska
Institute of Theoretical & Applied Informatics, Polish Academy of Sciences, Gliwice, Poland
E-mail: joanna@iitis.pl

Paris Avgeriou
Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, Groningen, The Netherlands
E-mail: p.avgeriou@rug.nl

Apostolos Ampatzoglou
Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece
E-mail: a.ampatzoglou@uom.edu.gr

Dimitrios Soudris
School of Electrical and Computer Engineering, National Technical University of Athens, Greece
E-mail: dsoudris@microlab.ntua.gr

Alexander Chatzigeorgiou
Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece
E-mail: achat@uom.edu.gr

Erol Gelenbe
Institute of Theoretical & Applied Informatics, Polish Academy of Sciences, Gliwice, Poland
E-mail: seg@iitis.pl

Dionysios Kehagias
Centre for Research and Technology Hellas, Thessaloniki, Greece
E-mail: diok@iti.gr

Dimitrios Tzovaras
Centre for Research and Technology Hellas, Thessaloniki, Greece
E-mail: dimitrios.tzovaras@iti.gr

in a real-world setting, through three use cases on actual commercial embedded software applications stemming from the airborne, automotive, and healthcare domains, as well as through an industrial study. To the best of our knowledge, this is the first quality analysis platform that focuses on multiple quality criteria, which also takes into account their trade-offs to facilitate code refactoring selection.

Keywords Embedded Software · Software Quality Evaluation · Energy Consumption · Dependability · Maintainability · Trade-off Analysis

1 Introduction

The increased utilization of embedded systems in our daily lives, which can be attributed mainly to the rising popularity and utilization of IoT systems that is observed recently (e.g., Smart Home, Smart Driving, etc.), has led to an increase in the production of software applications that are meant for embedded devices. The development of embedded software is a challenging task, chiefly due to the limitations in the available resources (e.g., available memory, energy capacity, etc.) that are imposed by the hardware platforms on which the software runs, as well as due to the often-conflicting non-functional requirements that it has to satisfy. For instance, embedded software applications need to be energy efficient, since embedded devices are normally battery dependent, as well as highly dependable, due to their high interconnectivity and accessibility through the Internet, while their maintenance costs should be kept as low as possible in order to ensure their longevity.

An important means for improving non-functional requirements during software development is code refactoring, which is the process of altering the source code of a software application in order to improve the target non-functional requirement while preserving its original functionality. However, it has been observed in the literature that a code refactoring that is employed for improving a specific quality attribute may have a negative impact on other critical quality aspects of the system (Holzmann 2017; Mohammed et al. 2016). Nevertheless, it is difficult, if not impossible, for a developer to estimate the impact that their code changes may have on various critical quality aspects, as well as to simultaneously satisfy the limitations imposed by the hardware platforms, as it requires experience and relevant expertise. Hence, there is a strong need for mechanisms able to assist developers in monitoring and optimizing critical quality attributes of embedded software, as well as in determining the impact that their code changes may have on critical quality attributes, in order to make more informed decisions.

Several quality attribute-specific methods, techniques, and mechanisms have been proposed over the years for quantifying various quality aspects of software, as well as for suggesting code changes that could improve those quality aspects. However, existing approaches face specific challenges that must be addressed. For instance, various dynamic energy indicators have been proposed and are widely used for measuring the energy efficiency of embedded systems, as well as the optimizations that could be employed for reducing their energy footprint (Eder et al. 2017; Zheng et al. 2016, 2017). However, there are no indicators able to estimate the energy-hungry hotspots of a given software, nor an approach that could a priori estimate the energy consumption of a given application in various hardware

architectures without requiring its actual execution, which would be highly useful during the development of low-energy applications. With respect to maintainability, the Technical Debt metaphor (Cunningham 1993) is the de facto standard for its quantification. However, current literature lacks approaches able to assess the maintainability of the new code that is added to a system, or to effectively prioritize maintainability-enhancing code refactorings. Finally, with respect to dependability, although optimization solutions exist (Mohammed et al. 2016), no effective and well-accepted indicators have been proposed in the literature so far.

Apart from the aforementioned challenges, the major problem of existing quality monitoring and optimization mechanisms is that they focus exclusively on a specific quality attribute. No mechanism (or platform) exists in the literature that provides quantitative indicators and improving capabilities for multiple non-functional requirements, especially for the case of embedded software. In addition to this, none of the existing quality attribute-specific monitoring and optimization mechanisms provide information on the potential impact that the proposed optimizations (i.e., code refactorings) may have on other critical quality attributes.

To this end, in order to address the aforementioned challenges, as part of the SDK4ED EU H2020 Project[1], we introduce the SDK4ED platform, which aims at facilitating the development of high quality embedded software, focusing mainly on the aspects of energy consumption, dependability (i.e., security and reliability), and maintainability. In particular, it provides solutions for monitoring and optimizing the three targeted quality attributes individually, both by utilizing state-of-the-art concepts, such as Technical Debt, and by introducing novel quality attribute-specific models and approaches. The SDK4ED platform, in an attempt to support decision making during the development cycle, also introduces advanced forecasting models able to provide projections for the future evolution of the quality attributes that the platform supports, helping in that way project managers better prioritize their testing and fortification efforts. Finally, a novel fuzzy multi-criteria decision-making technique is provided, which facilitates the selection of the best subset of code refactorings that should be applied to the source code of an embedded software application, i.e., those refactorings that improve a quality attribute of choice, without affecting (at least significantly) the other attributes, based on trade-off analysis among the often-conflicting criteria of energy consumption, dependability, and maintainability. This mechanism enables developers to make more informed decisions with respect to the code transformations that can be applied, reducing the possibility of unintentionally introducing new issues to other quality attributes.

The SDK4ED Platform was evaluated in a real world-setting, through three case studies, which were based on embedded software actively developed by three companies, coming from the automotive, healthcare, and airborne domains. In particular, the SDK4ED platform was utilized by the developers of these companies in order to monitor and improve specific quality attributes (among those supported by the platform) that they considered more critical for their applications. A broader industrial study was also conducted, in order to further evaluate the usefulness, practicality, and industrial relevance of the platform, as well as the potential financial benefits that it may provide in practice.

[1] https://sdk4ed.eu/

The SDK4ED Platform has been implemented in the form of a service-oriented platform, which can be deployed locally on the premises of an interested party. In addition, the novel mechanisms of the SDK4ED platform have been implemented as independent microservices, which are separately deployable and accessible through their dedicated REST API, enabling, in that way, their potential future integration into third-party applications. To the best of our knowledge, this is the first quality analysis platform that not only focuses on multiple quality attributes, but also takes into account the interdependencies among these quality attributes through trade-off analysis for facilitating code refactoring selection.

The purpose of the present paper is to provide a complete overview of the SDK4ED platform and the novel features that it provides for monitoring and optimizing the energy consumption, the dependability, and the maintainability of embedded software products. This is also the first paper that presents the results of the qualitative evaluation of the proposed platform and its novel features on a real-world setting through its application on three use cases coming from the automotive, healthcare, and airborne domains.

The rest of the paper is structured as follows: Section 2 discusses the vision of the SDK4ED project, whereas Section 3 provides an overview of the related work focusing mainly on the main challenges that the SDK4ED platform attempts to address. Section 4 provides an overview of the SDK4ED platform, along with a detailed description of the state-of-the-art and novel features that it exhibits, whereas in Section 5 a description of the technical implementation of the platform is given, along with information about its installation and utilization. Section 6 presents the evaluation of the SDK4ED platform in a real-world setting through three use cases with real companies and through an industrial study, whereas Section 7 provides some observations and lessons learned from the qualitative evaluation. Finally, Section 8 concludes the paper and provides directions for future work.

2 The Vision of the SDK4ED Project

The implementation of embedded applications requires horizontal expertise on both software and hardware aspects, and therefore the close collaboration between software and hardware engineers. The vision of the SDK4ED project is to bring together the software and hardware communities, by encapsulating the required expertise that each group lacks. In particular, software engineers operate chiefly on the application layer focusing on writing code to satisfy the desired functionalities of the system. Although they are aware of application-level challenges (e.g., functional correctness, user satisfaction, etc.), they usually lack knowledge and expertise with respect to the underlying hardware on which the application is running, leading to hardware-related overheads (e.g., energy consumption or performance degradation), which could have been avoided if they knew how to avoid these issues or even utilize hardware features in a proper way. On the other hand, hardware engineers operate at lower levels of the application stack, focusing on the effective utilization of the available hardware resources. However, due to their lack of software expertise, optimizations that they apply on the code for improving resource utilization (e.g., optimizing memory hierarchy utilization, using acceleration units etc.), may affect important high-level quality attributes, such as

the understandability and, in turn, the maintainability of the source code, which is a critical aspect for the longevity of the broader application.

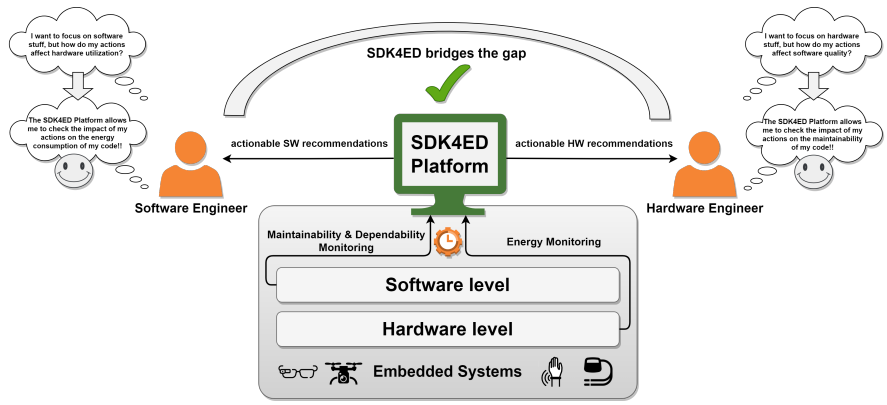


Fig. 1 The vision of the SDK4ED Project.

To this end, the SDK4ED platform attempts to bridge the gap between these two communities. In particular, the main goal of the platform is to allow the software and hardware engineers to focus on their activities, encapsulating all the required expertise that they lack, which is provided to the stakeholders in the form of actionable recommendations. For instance, a software developer can focus on the implementation of a specific feature without thinking about the underlying hardware, and the SDK4ED platform could inform him about the energy consumption, without the need of executing the code on the targeted devices to measure energy, that would increase the development time and cost. Also, in case their code is causing excessive energy consumption, the platform could provide information about why this is happening or how they could avoid it. Similarly, a hardware engineer could focus on applying energy and performance optimizations on the system, and the SDK4ED platform could inform them on the impact that their changes would have on the maintainability of the system, along with recommendations on how to reduce this impact. Hence, the SDK4ED platform could act as a link between the software and hardware knowledge, and facilitate the collaboration of software and hardware experts in the context of embedded systems in which this collaboration is necessary.

3 Related Work and Advances

The SDK4ED platform aims at facilitating the development of high-quality embedded software. This is achieved through the provision of novel mechanisms for enabling the independent monitoring and optimization of the energy consumption, dependability, and maintainability of embedded software, as well as through a decision support mechanism that considers the interplay among these three quality attributes for facilitating more informed code refactorings selection. In the present section, we provide the related work focusing mainly on the main challenges that

the SDK4ED Platform attempts to address and the advancements beyond the state-of-the-art that it provides, both for the individual quality attributes and for the unified consideration of software quality.

Energy Monitoring and Optimization: With respect to Energy Consumption, existing works vary significantly, based on the level at which the energy efficiency is treated. A large number of studies aimed at estimating energy consumption either by using hardware-specific performance/energy models (Eder et al. 2017) or by performing dynamic instrumentation to collect profiling features that feed machine learning models (Zheng et al. 2016, 2017). The impact of system calls on energy efficiency is also studied extensively (Aggarwal et al. 2015), while energy savings are usually achieved by focusing on data structure selection (Manotas et al. 2016).

The importance of designing a software development tool that offers continuous energy consumption monitoring and suggests optimizations is a challenging task that is also emphasized in recent survey studies (Georgiou et al. 2019). Pinto et al. (Pinto and Castor 2017), also, highlights software developers' lack of knowledge. According to this study, 50% of the participated developers could not improve energy in their applications.

Existing tools are also based on a variety of different approaches from estimation models and performance counters to direct measurements through hardware energy sensors. *Running Average Power Limit (RAPL)* (David et al. 2010) is a dynamic tool, supported by specific Intel architectures, which estimates power consumption from the CPU's performance counters. *Jalen* (Noureddine et al. 2015) is a popular tool in the Java community that estimates energy by analyzing JVM and the executed Java instructions. Mature and well-structured monitoring tools target Android Smartphones e.g. *Android Energy Profiler, Anandroid, GreenScaler, Trepn. PEEK* (Hönig et al. 2014) makes energy monitoring on the function level, by simply searching on a set of alternative power mode configurations, compiler flags, and libraries. SEEP (Hönig et al. 2012) makes coarse-grain estimations based on symbolic execution and *SEEDS* (Manotas et al. 2014) tries to make simple data structures optimizations.

Some approaches try to offer cross-device solutions based on simple machine learning algorithms (Bazzaz et al. 2013). However, they target only specific microcontrollers and instruction sets. More recent works that provide increased accuracy, utilize machine learning techniques (Zheng et al. 2016, 2017). However, they are based on dynamic instrumentation, imposing problems with regard to being integrated into a Software Development Toolbox. More specifically, they need application execution, adding also a large time overhead. In addition, they require a lot of manual actions by developers (e.g. adding annotations). Another category of tools used by practitioners to help them improve energy efficiency includes system emulators like Gem5 (Binkert et al. 2011; Lowe-Power et al. 2020). However, using emulators like Gem5 is extremely slow and also requires special knowledge from the users.

Embedded systems practitioners usually perform exhaustive design space exploration to select proper microarchitectural configurations (e.g., regarding the cache memory) in order to save energy (Wang et al. 2011; Reddy and Petrov 2010). Dynamic Voltage and Frequency Scaling is also used in CPU-based systems, leading to trade-offs between power and performance that enable energy consumption management (Awan and Petters 2011). Optimizing energy consumption is usually

the goal of studies that use custom SoCs, DSPs, or propose application-specific hardware designs. Also, the modern heterogeneous embedded devices, that include FPGA or GPU units on the same chip, offer acceleration capabilities that can also lead to significant energy savings (Llamocca et al. 2011; Fowers et al. 2012). One could argue that as the years go by, the burden of programming this type of devices falls more on libraries. For example, in machine learning applications, libraries such as Tensorflow, Pytorch, etc. offer the choice of using GPUs or TPUs easily from the Python code level. However, software developers still need advice about how and when to use all these features, as well as the configurations they need to make. An important motivation for the SDK4ED Energy Toolbox, is the lack of tools to assist developers in deciding upon using the heterogeneity capabilities of modern embedded devices to save energy. The few existing tools of deciding upon acceleration, focus only on speed-up prediction and target general purpose systems (CPU-GPGPU) (Wang et al. 2017; Lee et al. 2015; Ardalani et al. 2015), without supporting the prediction of energy savings.

SDK4ED Energy Toolbox offers three key features: (i) identification of energy consumption hot-spots and monitoring of energy indicators using dynamic instrumentation, (ii) cross-device energy consumption estimation solely based on static analysis, and (iii) identification of energy optimization opportunities, with the most significant being the estimation of energy consumption gains by utilizing accelerators (e.g., GPU). Although the proposed solution partially relies on existing tools, it also introduces new individual components that aim to offer advancements beyond the state-of-the-art. SDK4ED offers energy consumption estimation and optimization suggestions *without the need of executing the code on the targeted devices* (cross-device), lowering the barrier of access of embedded systems hardware and energy sensors for software engineers. By analysing the code on the backend (programmer's workstation or a host server), SDK4ED estimates the potential energy consumption of applications across various embedded devices, eliminating the need for access to those devices. While executing the code and measuring energy directly on targeted devices yields the most accurate results, not all hardware alternatives are accessible and such processes may require sophisticated equipment (e.g., special sensors) or expertise, increasing development time and costs. While the monitoring components, integrated into SDK4ED, report energy indicators and suggest optimizations based on code execution on the SDK4ED platform backend, the static analysis estimation component facilitates energy consumption estimation without requiring code execution, even on the SDK4ED backend. The developer, through static analysis, can get an early estimation of the energy consumption of their source code from the very early stages of the development, even when no working version of their application is available. Later on, when a working version is available, they can utilize the energy estimators that require actual execution of the source code, in order to gain better and more accurate results with respect to energy consumption. In addition, no accurate hardware modeling is required and the proposed solution is extensible in the sense that it allows users to add estimation models for other devices easily. Finally, the acceleration prediction tool, integrated in SDK4ED, focuses on the estimation of the potential energy savings by using performance-related machine learning features, extending the relevant State-of-the-Art approaches.

Maintainability Monitoring and Optimization: As far as Maintainability is concerned, the Technical Debt (TD) metaphor has recently become the de

facto standard for its quantification (Cunningham 1993). TD draws an analogy to the concepts of Principal and Interest from loans in classical economics. In the context of software maintenance, TD Principal expresses the time (or effort) that is required for fixing all the maintainability-related issues that reside in the source code of an application and is widely used as the main indicator of code or design quality (Li et al. 2015) (Ampatzoglou et al. 2015). On the other hand, TD Interest quantifies the additional cost (or effort) that needs to be paid for future maintenance, exactly because issues have not been resolved early enough during the development process and thus hinder the addition of new features or the fixing of errors (Seaman and Guo 2011). TD interest probability is the risk for an artifact that exhibits TD issues to undergo maintenance, thereby incurring additional maintenance costs (i.e., interest) (Ampatzoglou et al. 2018). In essence, the TD metaphor expresses in monetary terms, the consequences of 'sweeping problems under the carpet' in software development.

The impact of TD on the productivity and cost of software development and maintenance is tremendous: An empirical study surveying 43 developers about wasted time revealed that developers waste, on average, 23% of their time due to the presence of TD and that developers are frequently forced to introduce new TD (Besker et al. 2019). Another study showed that, if not repaid promptly, TD can even lead to a completely unmaintainable software (technical bankruptcy) (Suryanarayana et al. 2014). Results from the InsighTD family of surveys with researchers from eight countries indicate that the effects of TD which are more likely to be felt in software projects are quality issues and planning and management issues (such as delivery delay and need for rework) (Rios et al. 2019). Various types of TD exist depending on the software lifecycle phase in which it is incurred and the artifacts in which it resides, including code, design, architectural, documentation, test, and build debt (Brown et al. 2010). Nevertheless, code TD is the most studied type of technical debt in the literature and the most supported type of TD by existing tools for TD management (Fontana et al. 2016).

Several commercial tools and research prototypes have been released to measure TD through static analysis (Avgeriou et al. 2021). SonarQube is by far the most popular tool based on its popularity in the literature and the Web. Existing tools quantify the level of maintainability (i.e., TD principal), but very few tools focus on the consequence of these issues (i.e., TD interest) (Amanatidis et al. 2020), thereby weakening the use of TD as a means for convincing managers about the extra maintenance costs (interest) and the probability of additional maintenance (interest probability) and arguing about repaying TD.

The SDK4ED Platform supports all aforementioned TD concepts, namely principal, interest, and interest probability. It relies on SonarQube for the quantification of principal and introduces novel approaches for assessing interest and interest probability. Through the prioritization mechanisms that it offers, as well as visualizations targeting the concepts that matter most to software practitioners, it supports various activities of TD Management such as identification, quantification, ranking, and resolution of TD issues. Furthermore, the TD toolbox of SDK4ED supports not only the repayment of existing TD in software projects but also the prevention of TD by assessing the quality of new code that is to be committed against the quality of past versions. Finally, through the forecasting toolbox, it is also feasible to predict the anticipated evolution of TD at the level of individual software modules or at the level of the entire software project.

Dependability Monitoring and Optimization: With respect to Security, which is an important facet of Dependability, current literature lacks a well-accepted and reliable method for its quantification (Morrison et al. 2018; Ansar et al. 2018; Sentilles et al. 2018). Although several static and dynamic techniques have been proposed over the years for detecting security issues and suggesting security optimizations (Mohammed et al. 2016), which provide useful information that could be leveraged for the derivation of quantitative security measures, no meaningful indicators have been proposed so far (Morrison et al. 2018). Existing approaches are either subjective and unreliable (e.g., (Lai 2010; Alshammari et al. 2011; Medeiros et al. 2018)), as they are based on questionable parameters defined arbitrarily by the authors, or they are relatively reliable, but they are not operational, and therefore they cannot be used in practice (Colombo et al. 2012; Xu et al. 2013; Zafar et al. 2015; Dayanandan and Kalimuthu 2018). To fill this gap, the SDK4ED platform introduced a hierarchical security assessment model (SAM), which is able to provide a quantitative expression of the security of embedded software, based mainly on security information statically retrieved from the source code (Siavvas et al. 2021). The proposed security model is sufficiently reliable as it is in line with international quality and security standards (e.g., ISO/IEC 25010 (ISO/IEC 2011) and ISO/IEC 27001 (ISO/IEC 2013)) and its parameters are defined based on data and expert knowledge retrieved from well-accepted sources of information like the Common Weakness Enumeration (CWE). It is also practical as it is operationalized in the form of a web service, which is either directly invokable or accessible through the SDK4ED platform.

Building secure embedded software, apart from quantitative security indicators, also requires mechanisms able to highlight software components (i.e., classes, methods, etc.) that require attention from a security viewpoint. Vulnerability prediction models, which are machine learning models that are able to detect potentially vulnerable software components, are suitable for this task. Several vulnerability prediction models have been proposed over the years, with text mining-based models to demonstrate the best predictive performance (Scandariato et al. 2014; Dam et al. 2018; Li et al. 2018; Zhou et al. 2019; Hanif and Maffeis 2022; Kim et al. 2022; Hanif and Maffeis 2022). Software metrics have also shown promising results (Chowdhury and Zulkernine 2011; Shin et al. 2011; Zagane et al. 2020). Hence, the SDK4ED platform proposes novel vulnerability prediction models that combine both text features and software metrics and utilize Random Neural Networks as a bonding model for building hybrid models that combine both types of features (Filus et al. 2021b,a). Very limited attempts can be found in the literature that combine both text features and software metrics, whereas it is the first time that Random Neural Networks were used for vulnerability prediction purposes.

For optimizing the Reliability of embedded software, which is another important facet of dependability, the checkpoint and restart (CR) (Egwutuoha et al. 2013; Arora 2017; Shahzad et al. 2018) mechanism is widely used in practice, especially in High-Performance Computing (HPC) systems (Elnozahy et al. 2002; Takizawa et al. 2011; Losada et al. 2016; Rodríguez et al. 2010; Hursey et al. 2007; Moody et al. 2010), in which reliable execution is a critical concern. In the CR mechanism snapshots of the program (i.e., checkpoints) are periodically generated, and in case of a failure the execution restarts from the most recent checkpoint (instead from the beginning), avoiding in that way excessive re-executions. However, the CR mechanism is known to introduce significant overheads in the execution

of the program, which have been found to be affected by the checkpoint interval, i.e., the interval between two consecutive checkpoints. Although several CR libraries and tools have been proposed over the years, none of them provide recommendations for the selection of the inter-checkpoint interval. To this end, the SDK4ED platform introduces novel mathematical models for modeling a program with and without the presence of checkpointing, as well as for computing the optimum checkpoint interval, i.e., the interval that minimizes the checkpointing-induced overheads (Siavvas and Gelenbe 2019a,b; Gelenbe et al. 2020; Gelenbe and Siavvas 2021).

Quality Attribute Forecasting: Existing quality attribute-specific monitoring mechanisms provide quantitative indicators for determining the current status of a given software application with respect to an important non-functional requirement. Apart from its current status however, estimating the future evolution of critical quality attributes of a given application is also important for making more informed decisions during its development. For instance, if the Technical Debt of an application is expected to increase significantly in the near future, immediate TD repayment activities may be employed in order to prevent the accumulation of TD, and, in turn, minimize the risk of reaching the point at which the application becomes unmaintainable (i.e., the breaking point (Chatzigeorgiou et al. 2015)). Despite the apparent importance of forecasting the evolution of quality attributes, no relevant attempts could be found in the literature. To this end, the SDK4ED Platform introduced novel models for forecasting the evolution of a quality attribute of choice based on both time series and machine learning techniques, focusing mainly on the aspect of TD, as the evolution of TD is important for the success and longevity of an application, due to its relation to maintenance costs (Tsoukalas et al. 2019, 2020). The novel TD-specific forecasting models have been successfully extended and applied for the cases of energy consumption and dependability (particularly, security) as well.

Trade-off Analysis: As can be seen by the above analysis, the quality monitoring and optimization models that have been proposed so far focus exclusively on a specific quality attribute (i.e., non-functional requirement). These models are able to provide quantitative indicators of a specific quality attribute (e.g., energy consumption, maintainability, etc.), and suggest optimizations that are meant to improve the quality attribute of choice. To the best of our knowledge, and despite the fact that there is empirical evidence for the conflicting nature of critical non-functional requirements, no model, technique, or mechanism exists in the literature that also reports an estimate of the potential impact that its suggested optimizations may have on critical quality attributes other than the target quality attribute that it attempts to optimize. The most relevant solution is the Quamoco quality platform (Wagner et al. 2015). This platform enables the user to define their own quality models, giving them the opportunity to define multiple quality attributes and determine how they can be quantified through static analysis. However, the platform does not support runtime quality attributes like energy consumption, which is important for embedded systems. In addition to this, it does not report and account for the interplay (i.e., trade-offs) among different quality criteria, and the impact of code refactorings on the critical quality attributes. Finally, Quamoco is not operational in the form of a tool and it seems to have become obsolete.

The SDK4ED platform attempts to address the aforementioned challenges by providing support both for design-time (e.g., maintainability) and runtime (e.g.,

energy consumption) quality attributes, through the integration of state-of-the-art monitoring and optimization mechanisms in a unified manner. It also accounts for the interplay among the often-conflicting quality attributes of energy consumption, dependability, and maintainability, by conducting trade-off analysis among these quality aspects, in an attempt to assist code refactoring selection. In particular, a novel mechanism is proposed that computes the impact of code refactorings on these three quality attributes, in order to assist developers in selecting the best subset of code refactorings according to their needs, e.g., those code refactorings that will improve a specific quality attribute, without affecting (at least significantly) the other quality attribute of choice (Lamprakos et al. 2022). To the best of our knowledge, no mechanism or platform exists in the literature that facilitates code refactoring selection by reporting the impacts of suggested optimizations (i.e., code refactorings) on multiple critical non-functional requirements. In addition to this, no quality platform exists that provides support for multiple design-time and runtime quality attributes, especially for the case of embedded software.

An overview of the SDK4ED platform and its core functionalities has been provided in (Marantos et al. 2022c). However, in (Marantos et al. 2022c) the main functionalities were not described in detail, whereas no emphasis on the actual evaluation of the platform was given. In the present paper, a detailed description of the SDK4ED platform is provided, giving a complete overview of the novel features that it provides, allowing the reader to gain a deeper understanding of the proposed novelties. In addition to this, the present paper gives specific emphasis on the qualitative evaluation of the SDK4ED Platform on a real-world setting through its application on three use cases from the automotive, healthcare, and airborne domain. It also demonstrates the main conclusions and lessons learnt that were derived from the three years of the project.

4 Innovations of the SDK4ED Platform

The purpose of the SDK4ED platform is to facilitate the production of high-quality embedded software, focusing mainly on the quality aspects of energy efficiency, dependability, and maintainability. It achieves this by providing both state-of-the-art and novel quality attribute-specific monitoring and optimization mechanisms, as well as through the provision of a novel mechanism for assisting the selection of the code refactorings that should be applied to the source code of the embedded software. Code refactorings selection is based on their impact on the three quality attributes of interest, which is computed through trade-off analysis that is based on fuzzy multi-criteria decision making techniques. The high-level overview of the SDK4ED platform is illustrated in Figure 2.

As can be seen by Figure 2, the SDK4ED platform consists of five different components, i.e., toolboxes, namely the Energy Toolbox, the Maintainability Toolbox, the Dependability Toolbox, the Forecasting Toolbox, and the Decision Support Toolbox. The first three components, as their name indicates, provide solutions for monitoring and optimizing the energy consumption, the maintainability, and the dependability of embedded software respectively. The Forecasting Toolbox is responsible for providing projections of the future evolution of the three quality attributes that the SDK4ED platform focuses on. The last toolbox is responsible for supporting decision-making with respect to the selection of appropriate code

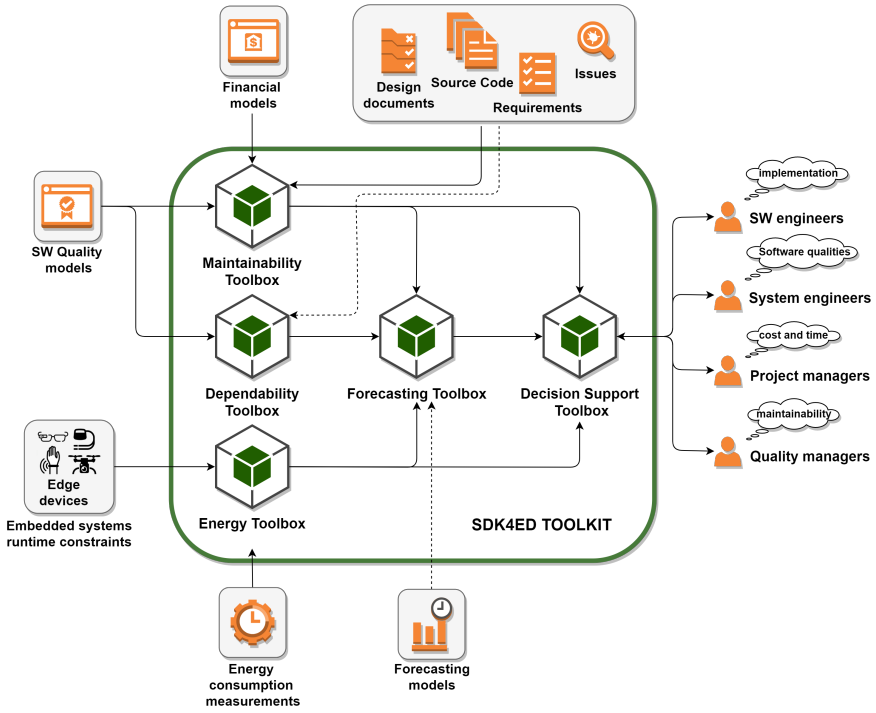


Fig. 2 The high-level overview of the SDK4ED Platform.

refactorings to be implemented in the code, based on their impact on energy consumption, maintainability, and dependability, which is determined through trade-off analysis among these often-conflicting quality attributes. As shown in Figure 2, the Decision Support Toolbox depends on the outputs of the other four modules, acting like a bonding model among the various novelties provided by each toolbox. The SDK4ED platform is able to analyze software applications that are written in C, C++, and Java programming languages, which are highly popular in embedded software development. In the rest of the present section, each one of the aforementioned components is described, putting emphasis on the novel concepts that the SDK4ED platform introduces.

4.1 Energy Toolbox

The SDK4ED Energy optimization methodology consists of two parts. The first one (Consumption Analysis) is responsible for analyzing the application source code in terms of energy consumption either by monitoring energy consumption indicators and identifying the most energy-consuming parts of the code (hotspots), where developers should focus, or by estimating the energy consumption of individual application code blocks if executed on a number of devices (cross-device) through static analysis. The second component (Optimization Suggestions) aims on suggesting source code transformations that will potentially reduce the energy

consumption of an identified hotspot. The flow of the SDK4ED Energy Toolbox methodology is depicted in Figure 3 and is presented in detail in the following paragraphs.

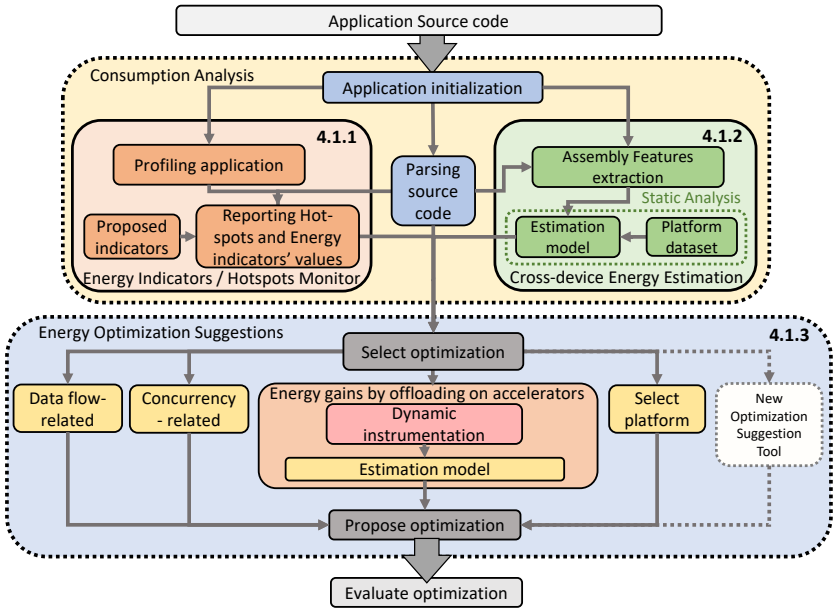


Fig. 3 Energy Support Toolbox flow

4.1.1 Energy Indicators and Hotspots identification (Dynamic analysis)

The analysis flow starts with the *Application Initialization*, which includes both retrieving information about the targeted source files by the user and building/running the application. Then it proceeds to *Parsing source code*. In this step, by generating the Abstract Syntax tree (AST) of the application using CLANG[2], *for* and *while* code blocks are identified in the application source code.

The first step of the dynamic analysis is *Profiling*, which is supported by tools that perform dynamic instrumentation, Valgrind[3] (Cachegrind and Callgrind) as well as Linux Perf[4]. The generated output of the application profiling is recorded in log files that are processed returning the *Hot-spots and the Energy Indicators*.

We selected energy consumption indicators that can be monitored by using tools that are mature, active, and widely used by the embedded systems community. Their selection is also based on the following three criteria:

– The values of the selected indicators should be directly related to the source code of the application. In other words, source code refactorings should have a

[2] https://clang.llvm.org/
[3] https://valgrind.org/
[4] https://perf.wiki.kernel.org/

Table 1 Selected Energy Indicators

Indicators (supported)	Indicators (to be added)
CPU related	
# CPU cycles	Ratio of CPU stalls
# Instructions	
Branch miss ratio	
Memory related	
# Memory accesses	# Page faults
I-cache miss ratio	# Heap memory blocks lost
D-cache miss ratio	
Multi-threading related	
# Data races	Lock contention
	Lock order violation
Acceleration specific indicators	
Instruction level parallelism	
Memory/control/integer/mul/div/fp operations	
Number of cold misses	
Branch divergence	
Memory stride	

direct impact on their values. Energy indicators that are mainly controlled by the operating system or by the hardware architecture-level techniques are not selected.
- There exist source-to-source optimizations that may improve their values. The values of the indicators monitored by the energy consumption toolbox are expected to indicate source-to-source optimizations of energy consumption. Therefore, indicators for which no optimization has been proposed in the literature are not selected.
- A subset of these indicators shows the efficiency of assigning the execution of a part of CPU source code on an accelerator. We selected indicators that can be used to estimate the energy gains of offloading a piece of application source code on an accelerator (Acceleration specific indicators).

The selected energy consumption indicators are presented in Table 1, highlighting the metrics currently supported by the SDK4ED platform and the metrics that will be monitored in future versions.

Energy Hotspots identification Procedure: In the context of the SDK4ED Energy Toolbox, an energy hotspot is defined as a block of CPU source code, in which a significant number of CPU cycles are spent, compared to the application's total. Each identified hotspot is considered a candidate place to check for energy-related optimizations. The application is dynamically analyzed to monitor CPU cycles, by leveraging a widely used dynamic binary instrumentation profiler: Callgrind by the Valgrind suite. By combining the information generated by the dynamic analysis (i.e., Callgrind output) and the statements identified by the AST processing, the number of CPU cycles spent in each statement is calculated. The code blocks in which the number of CPU cycles spent is above a threshold (1% of the total applications cycles) are considered hotspots. For each hotspot, the corresponding values of energy indicators such as CPU cycles and cache misses are provided. All this information is forwarded to the next component, which is responsible for suggesting suitable optimizations.

Table 2 List of static analysis features

Static Analysis Feature
Estimated throughput (by LLVM-mca)
Number of instructions
Number of LOAD instructions
Number of STORE instructions
Number of OP (operations) instructions
Number of class 1 instructions (add, sub, shift, mul)
Number of class 2 instructions (conv, arrays, div)
OP, LOAD and STORE instructions order

4.1.2 Energy Consumption Estimation (through Static Analysis)

The alternative static analysis approach comes as a mitigation of the constraints of the aforementioned dynamic analysis that adds a large time overhead and requires the execution of the programs under analysis. The static analysis mechanism aims to make the SDK4ED Energy estimation component easier to use and more similar to the components of the rest of the SDK4ED toolboxes, namely the Maintainability and the Dependability Toolboxes (see Section 4.2 and Section 4.3).

The first part of this component uses the code blocks identifier from the Hotspot identification step (described in Section 4.1). Due to the fact that only the source code is analyzed (without dynamic information), the static analysis focuses on code blocks (e.g., loop bodies, function bodies, etc.) and does not contain iterations, branches, or calls. Then, the object file (generated by the compiler) is analyzed for *Extracting Features*: information to be used as input to an energy *Estimation Model*. The SDK4ED Energy Toolbox uses features that model the application's behavior and computational requirements that energy consumption based on the application's assembly and the output of the LLVM-mca tool analysis. Assembly instructions are also categorized, based on their energy. We made this choice because the proposed solution aims to be cross-device, supporting a wide range of architectures and instruction sets. The selected features are presented in Table 2. With regard to the order of the assembly instructions (the last feature presented in Table 2), a simple sliding window approach was employed. To extract order features, the window runs through the assembly instructions of the application and each combination of the basic instruction categories (LOAD, STORE, and OP) corresponds to a new feature (Marantos et al. 2021) resulting in 27 new features. Of course, not only the type of instructions, but also the registers being used and the location of the accessed data affect the consumption. However, the goal of the presented approach is not to achieve the maximum accuracy but to offer flexibility and wide applicability (cross-device).

The most important part of the SDK4ED energy estimation is the model. The procedure of selecting the best model is rather straightforward: We compare the accuracy of using alternative models expressed to make predictions on a subset of the dataset. We used the Nvidia Tegra TX1 platform, which incorporates an integrated ARM-Cortex A57 processor and a built-in energy sensor (INA 3221). Figure 4 shows the accuracy of the most suitable models. For each model, the Mean Absolute Error between the actual values and the predicted values is presented. According to these results, we conclude that the Orthogonal Matching Pursuit model makes better predictions. The error refers to the execution of just one loop

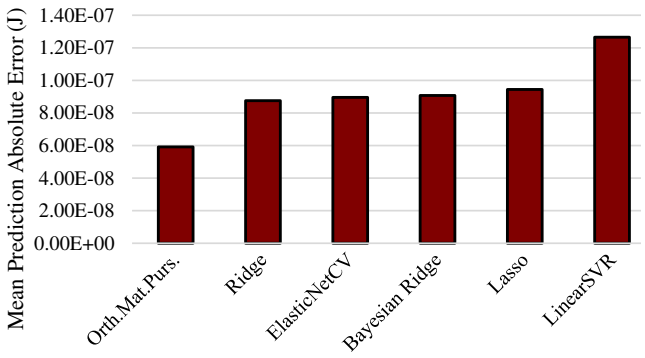


Fig. 4 Alternative static analysis based energy estimation models comparison

iteration, as the presented process estimates the basic block execution energy, without using any dynamic information such as the number of iterations.

4.1.3 Energy Optimizations

The energy toolbox apart from providing means for monitoring and estimating the energy consumption of embedded applications and identifying energy hotspots, it also provides recommendations for optimizing energy consumption. In particular, it suggests three categories of optimizations (*Select optimization* step in Figure 3), namely: (i) *Data-flow optimizations*, (ii) *Concurrency-related optimizations*, and (iii) *Acceleration optimizations*. These categories are described in the rest of this section. An additional type of optimization is *(iv) Select platform*, which refers to selecting a less energy consuming platform based on the energy estimations provided by SDK4ED toolbox (described in Section 4.1.2).

a. Data-flow optimizations: The first category of energy optimization techniques, at the application level, aims to improve the memory hierarchy utilization (Catthoor et al. 2002). Since the energy consumed by memory references depends on whether the access hits or misses in the cache memory, we can claim that the cache behavior is very important for optimizing energy/performance. Typical examples of these techniques are the loop transformations (Table 3) that aim to improve the cache performance, by improving data locality and reducing the overhead of the loops, which are often the most computationally intensive parts of an application. Furthermore, due to the fact that each memory access has a cost in terms of energy and performance, this kind of transformation aims also to improve memory utilization and reduce memory allocation and the number of memory accesses. Data-flow optimizations are proposed in the case that the hotspot under analysis includes nested loops that have a number of cache misses that is beyond a threshold (3%).

b. Concurrency-related optimizations: Concurrency (i.e., multiple threads simultaneously accessing the same data) often imposes significant challenges. As modern embedded systems typically integrate multiple cores, embedded developers are facing challenges imposed by concurrency. Misuse of available methods for protecting data from corruption may lead to significant losses in terms of performance and energy (Fowers et al. 2012). A typical example is the deadlock bug,

Table 3 Indicative loop transformations for improving energy

Before	After
Loop Merge:	

`for (i=0; i<N; i++) {`
`  //do something...`
`}`
`for (i=0; i<N; i++) {`
`  //do something else...`
`}`
	`for (i=0; i<N; i++) {`
	`  //do something...`
	`  //do something else..`
	`}`

Loop Interchange:	

`for (j=0; j<N; j++) {`
`  for (i=0; i<N; i++) {`
`  sum += a[i][j];`
`  }`
`}`
	`for (i=0; i<N; i++) {`
	`  for (j=0; j<N; j++) {`
	`  sum += a[i][j];`
	`  }`
	`}`

Loop Tiling:	

`for (i=0; i<MAX; i++) {`
`  for (j=0; j<MAX; j++) {`
`  A[i][j] = A[i][j] + B[i][j];`
`  }`
`}`
	`for (i=0; i<MAX; i+=BLOCKSIZE) {`
	`  for (j=0; j<MAX; j+=BLOCKSIZE) {`
	`  for (ii=i; ii<i+BLOCKSIZE; ii++) {`
	`  for (jj=j; jj<j+BLOCKSIZE; jj++) {`
	`  A[ii][jj] = A[ii][jj] + B[ii][jj];`
	`  }`
	`  }`
	`  }`
	`}`

which occurs when two or more threads expect the release of a resource held by the other thread. In this case, no thread can make progress, and the application stalls. The solution to this problem is the proper use of locking mechanisms to avoid deadlocks. Another example is the lock contention: multiple CPU cores stall for a significant amount of time as they constantly try to acquire a resource held by another thread (often called "polling"). Thus, the cores consume energy without making actual progress. One solution is the use of another locking mechanism, that may reduce contention (e.g. locking with a back-off policy, in which the core makes each new attempt to acquire the held resource periodically instead of constantly).

The SDK4ED energy toolbox reports Data races in the application source code: Data races (also known as "race conditions") occur when the order of accesses on shared data is not deterministic and the computation may give different results in each execution. The application may generate incorrect results, which apparently result in energy losses. The solution that eliminates data races and is suggested to developers is the proper use of locks so that the access to the shared resources will be deterministic.

c. Energy gains by offloading on accelerators: A massive improvement of performance and reduction of energy consumption can be achieved by using accelerators (Fowers et al. 2012) (Marantos et al. 2022b). Nowadays, a plethora of heterogeneous embedded computing architectures provides increased performance at limited energy consumption. A complementary infrastructure, which is part of modern heterogeneous System-on-Chip (SoC) embedded devices usually includes both CPU and GPU or CPU and FPGA. Offloading computationally intensive

parts of an application to the acceleration unit cannot be considered a trivial task, due to the large number of source code refactorings that have to be performed manually. Depending on the kind of accelerator different tools and programming or hardware description languages have to be used in a proper way.

The role of the Energy Toolbox is to provide acceleration gains prediction for identified hotspots. This analysis component is based on a *Dynamic instrumentation* approach. We decided to use estimation models based on metrics resulting from the profiling tools, after investigating which of the metrics are suitable for predicting energy gains by acceleration (Marantos et al. 2022b). To identify these features, we used the stepAIC method that identifies an optimal set of features by selectively adding and removing features in each step and using regression methods to evaluate the importance of each one. The resulting features are *acceleration specific indicators* presented in Table 1. Similar features are also used in the literature ((Ardalani et al. 2015) (Wang et al. 2017)) for estimating the potential speedup of using General Purpose GPUs (GPGPUs).

The values of the acceleration-specific indicators are forwarded to the classification *Estimation model*. This classification process aims at assisting developers to decide if it is worth developing GPU code for the hotspots for which energy consumption gains are predicted. In order to select a proper classification model, we applied k-fold cross-validation techniques to test the estimation accuracy of alternative models. Figure 5 summarizes and quantifies the accuracy level of the investigated methods. As it can be observed, the three most accurate models are the Random Forests, the Bagged Trees, and the Gradient Boosting. Their performance can be further improved by using an Ensemble Voting Method that combines their results, reaching a final accuracy level of 85%. For building the dataset we combined an equal number of synthetic benchmarks and real-world applications (i.e. non-synthetic and taken from existing benchmark suites) to reduce the danger of over-fitting caused by a small dataset size that could lead to biased results (Marantos et al. 2022b).

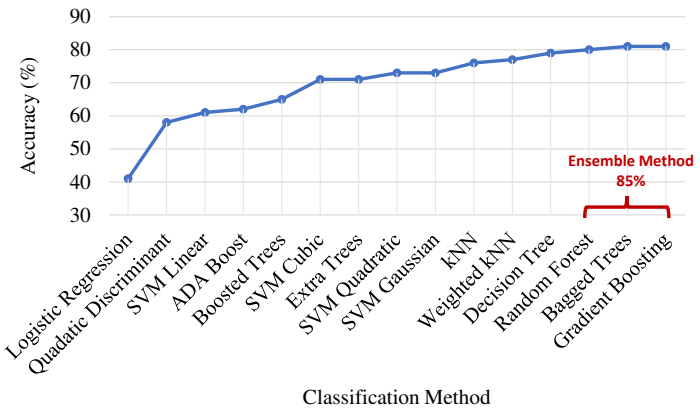


Fig. 5 Comparing alternative models for predicting energy gains class

4.2 Maintainability Toolbox

The Maintainability Toolbox provides support for all aspects of Technical Debt Management (TDM), namely: identification, quantification, and repayment. We remind that Technical Debt as a metaphor in software engineering expresses, in monetary terms, the consequences of not addressing existing software inefficiencies (TD Principal) leading to increased effort and costs during future maintenance (TD Interest). Next we describe the major axes along which the maintainability of a given software system is analyzed through the SDK4ED Platform.

4.2.1 TD Analysis of Existing Code

The pillars of TD theory are two concepts borrowed from economics: principal and interest. On the one hand, *Principal* corresponds to the effort that needs to be spent so as to refactor the existing system to an optimal one with respect to structural quality and maintainability. In practical terms, principal refers to the effort that is needed to mitigate all rule violations that have been identified by a static analysis tool like SonarQube. On the other hand, *interest* corresponds to the additional costs that occur along with maintenance, due to poor software quality.

For TD Principal quantification there is an abundance of available tools (Avgeriou et al. 2021) (e.g., SonarQube, CAST, Squore, etc.). In the context of SDK4ED, we have opted to use SonarQube, since it is the most commonly used tool in the industry and academia and is open-source, avoiding the dependence of SDK4ED on closed-source software (Avgeriou et al. 2021). TD Principal is estimated by identifying code smells and calculating the sum of the time required to remediate all of them.

The quantification of TD Interest is far more challenging, as it entails the anticipation of future maintenance activities as well as the estimation of the effort that would be spent on maintaining an 'optimal' version of a given system. For the quantification of TD Interest, we relied on the FITTED framework (Chatzigeorgiou et al. 2015) (Ampatzoglou et al. 2018). The central idea behind the calculation of TD interest is that for any given system that has an actual implementation, we can assume that there exists a hypothetical optimal state. We obtain an approximation of the optimal state for any given software module by identifying its peers (i.e., other modules that have similar characteristics, such as size) and calculating the best values for a selected set of metrics. The hypothetical peer exhibiting the best metric values plays the role of the "optimal" implementation.

Maintaining the optimal system would require less effort than maintaining the actual system, while maintenance effort is assumed to be inversely related to the maintainability of the system. Thus, the maintenance effort for the optimal system (which we seek to assess), can be estimated as the product of the maintenance effort for the actual system and the ratio of the maintainability of the actual over the maintainability of the optimal system. Based on its definition, TD Interest can be calculated using the difference between the actual and the optimal effort, where the effort for the actual system is calculated as the average past maintenance load (lines of code changed, per version). Principal and Interest are graphically depicted in Figure 6 assuming a feature A that has to be added in the actual and (hypothetical) optimal system.

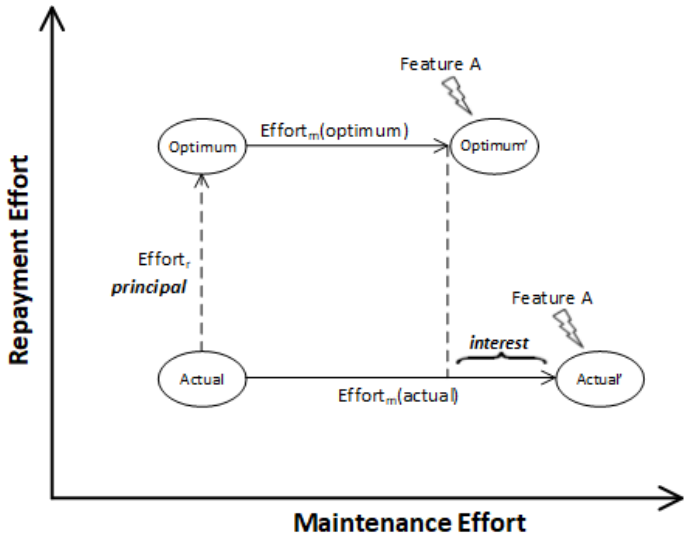


Fig. 6 TD Principal and Interest (Chatzigeorgiou et al. 2015)

Beyond TD Principal and Interest, the SDK4ED maintainability toolbox supports the quantification of TD Interest Probability which refers to the probability of software modules to undergo maintenance. Interest probability is highly relevant to maintenance prioritization, since the urgency to resolve a code or design inefficiency depends on whether the corresponding piece of code is going to be frequently changed or not. TD Interest Probability is calculated by considering the number of commits in which an artifact has changed and the probability of an artifact to change due to ripple effects propagated from other artifacts.

Finally, the TD analysis of an existing codebase in SDK4ED entails the calculation of the TD Breaking Point (Chatzigeorgiou et al. 2015). During the evolution of software systems, the accumulated debt in the form of interest can in some cases quickly sum up to an amount that at some point, exceeds the effort that was originally required to repay the initial amount of TD. The number of versions that will elapse until the Breaking Point occurs is obtained through the ratio of Principal over Interest. Anticipating how late the breaking point is expected to come can inform decision making with respect to investments in improving software quality.

4.2.2 TD Analysis of New Code

To eliminate the TD phenomenon there are two general strategies: the first is to adopt an after-the-fact approach, that is, to address the existence of problems after their identification. Problems can manifest themselves in many complementary forms such as code or design smells, excessive metric values, lack of design patterns, anti-patterns, violations of heuristics, etc. This approach will be discussed in the next subsection.

The second approach is to ensure that new software artifacts are problem-free in the first place, that is, when code is committed to a repository. Even if 'new

code' is not entirely TD-free, consistently adding new code whose Technical Debt is lower than the system average can lead to a gradual improvement of quality. A key concept in the assessment of TD for 'new code' is TD density. We relied on the notion of $TD_{density}$ because absolute measures of TD (such as the number of identified rule violations) usually increase monotonically with the addition of code. $TD_{density}$ is simply obtained as the ratio of TD identified in a piece of code, over the corresponding lines of code.

The Maintainability Toolbox of SDK4ED contains all the necessary functionality to (a) clone a git repository (from a user-provided URL), (b) checkout code through the history of the commits, (c) measure TD for each revision with the help of SonarQube, (d) detect the specific issues that incur TD, (e) detect the changes (at the method-level) from one revision to the other (added, modified, and deleted methods), (f) measure how these method-level changes affect the $TD_{density}$ of the project along evolution, and (g) perform quality gate analysis between any given planned commit (e.g., a commit residing in a development branch, which is to be merged to the production code branch), to detect if the "new" code contributes positively or negatively to the project. The platform's front-end visualizes the $TD_{density}$ of the code that is to be integrated into the projects against the $TD_{density}$ of the codebase, allowing the developer to grasp whether the new code is of better, equal or lower quality.

An insight into how the quality of new code can affect the overall system quality along its evolution is depicted in Figure 7. The plot illustrates the evolution of the system's $TD_{density}$ for project CommonsIO[5] (black dots) and the corresponding trend line, depicting a gradually increasing quality (black line declines over time). Blue dots correspond to the revisions, in which the $TD_{density}$ of new methods was better (lower), while red dots indicate the cases where the $TD_{density}$ of new methods was worse (higher) than that of the host system. As it can be observed, for the vast majority of revisions, the $TD_{density}$ of new methods is lower than that of the host system and in many cases the new code is entirely TD-free (see blue dots along the x-axis). Consequently, one can argue, that the 'cleanness' of new code was the key driver for the improvement in the systems' quality.

4.2.3 Refactorings to address TD

The most popular strategy for repaying TD is the identification of code smells and the subsequent application of refactorings to remove these smells (Fowler 1999). While various code smell detectors exist (Tsantalis et al. 2018) it is impractical to remove all smells in a software system. Therefore, it is imperative to perform some sort of refactoring prioritization by focusing on pieces of code that suffer from severe symptoms of low maintainability and at the same time are often subjected to maintenance.

The Maintainability Toolbox, since it is built on top of SonarQube, is able to detect and report all the code smells, along with their corresponding refactoring opportunities, which are officially supported by SonarQube. In particular, SDK4ED offers the possibility to explore various refactoring opportunities, by getting indications on the severity of code smells, as retrieved by SonarQube and by retrieving the smells that have the higher probability of producing interest in an

[5] https://github.com/apache/commons-io.git

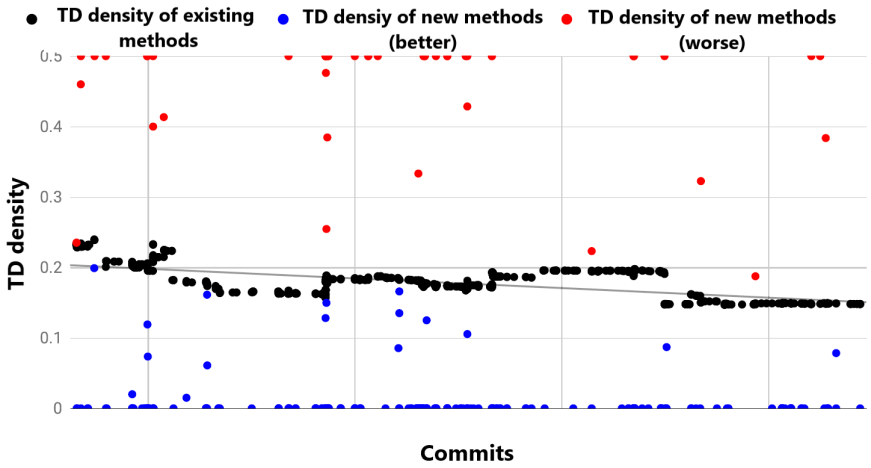


Fig. 7 Impact of new code quality on system's $TD_{density}$ (adapted from (Digkas et al. 2022)

upcoming version of the system. As an example, the user is informed that the "Methods Should not be Empty" smell occurs in 31% of the classes in the project, and these files have on average 5% probability to change. By combining these probabilities, the total urgency to resolve these smells' instances is calculated.

In addition to the refactoring opportunities identified through SonarQube, SDK4ED supports two types of refactorings which can have a large impact on system quality. The first one aims at the identification of long methods which are good candidates for applying the extract method refactoring. The identification of extract method refactoring opportunities relies on the Single Responsibility Principle (SRP). The proposed SRP-based Extract Method Identification (SEMI) approach recognizes fragments of code that collaborate for providing functionality by calculating the cohesion between pairs of statements, as presented by (Charalampidou et al. 2017). The approach is particularly efficient for identifying functionally related statements within very long methods (e.g. methods with 500 or 1000 lines of code). Second, SDK4ED supports a package-level re-organization method, that ensures conformance to the low coupling and high cohesion principle. In particular, through the application of a genetic algorithm that suggests Move Class refactorings (i.e., the move of classes from one package to another, or to new packages), system modularity can be optimized. The application of the SRP principle at the package level to recommend Move Class refactorings is facilitated by the computation of two metrics, namely the average afferent coupling of packages (i.e. the number of outgoing dependencies of a package to other packages) and cohesion among package Classes which assesses how closely two classes that belong to the same package collaborate with each other, introduced by (Ampatzoglou et al. 2019).

It should be noted that Long Method smells, which can be resolved by extracting methods from the method's body, and packages exhibiting high coupling and low cohesion, which can be mitigated by moving classes to other packages, are two non-trivial, yet frequently occurring problems in software development.

SonarQube and other Technical Debt management toolboxes do not identify such inefficiencies, as their detection goes beyond the mere violation of rules.

4.3 Dependability Toolbox

The Dependability Toolbox is responsible for providing solutions for monitoring and optimizing the dependability of embedded software applications. In particular, the SDK4ED platform focuses on the aspects of security and reliability, which constitute two important facets of dependability, according to Sommerville (Sommerville 1995). During the course of the SDK4ED project, novel mechanisms have been developed and important challenges have been addressed with respect to these two dependability aspects, which are described briefly in the rest of this section.

4.3.1 Software Security

Quantitative Security Assessment
Being able to quantify the security level of an embedded software application during its development is critical, since it is commonly believed in the literature that "you cannot control something that you cannot measure" (DeMarco 1986). However, no well-established model for providing quantitative assessment of software security is available on the market, even though the existence of quantitative security indicators is considered highly important for secure software development (Morrison et al. 2018). Although a small number of evaluation methodologies and models have been proposed recently, they are either unreliable (i.e., based on questionable parameters) or impractical (i.e., non operational) (Lai 2010; Xu et al. 2013; Colombo et al. 2012), preventing them from being adopted in practice.

To this end, as part of the Dependability Toolbox of the SDK4ED platform, we provide a novel hierarchical security assessment model (SAM) (Siavvas et al. 2021) for evaluating the internal security level of software applications in a quantitative manner, based on low-level indicators retrieved from their source code through static analysis. The model aggregates the low-level indicators based on a set of parameters (i.e., thresholds and weights) in order to compute a high-level security score, i.e., the Security Index, which reflects the internal security level of the analyzed software, based on the ISO/IEC 25010 (ISO/IEC 2011) international standard.

The model, following the guidelines of ISO/IEC 25010 (ISO/IEC 2011), and based on a set of thresholds and weights, systematically aggregates these low-level indicators in order to produce a high-level security score, i.e., the Security Index, which reflects the internal security level of the analyzed software. For better understanding, the high-level overview of the proposed model is illustrated in Figure 8.

As can be seen by Figure 8, the proposed model hierarchically decomposes the notion of security into a set of security characteristics (e.g., Confidentiality, Integrity, and Availability), which are further decomposed into a set of security properties (e.g., Null Pointer, Buffer Overflow, etc.). Those properties are more tangible and can be quantified directly from the source code through static analysis. The security properties that are supported by the SDK4ED Dependability

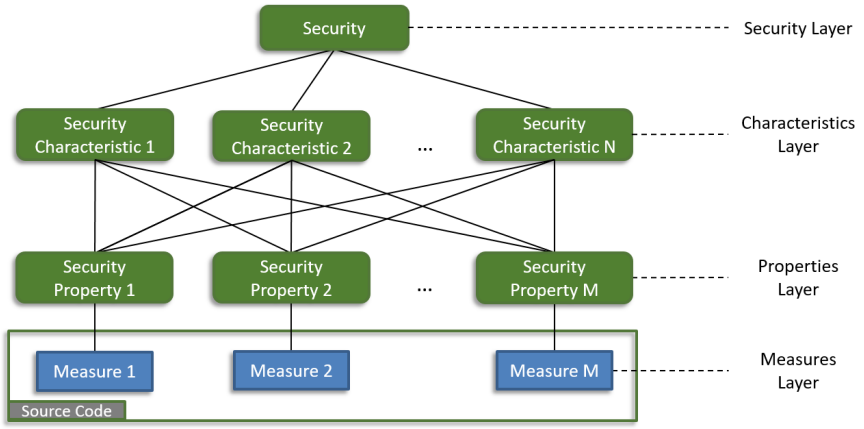


Fig. 8 The Security Assessment Model (SAM) of the SDK4ED Dependability Toolbox.

Toolbox for each one of the supported programming languages are reported in Table 4. The security properties are, in fact, groups of closely-related static analysis alerts (i.e., vulnerability categories). For instance, the Null Pointer property contains issues that are related to null pointer dereferences. For measuring the security properties, the static analysis vulnerability density (SAVD) metric (Walden et al. 2009) is utilized, which corresponds to the number of static analysis alerts that belong to this property, per thousand lines of code. In brief, the proposed model applies static analysis in order to compute the SAVDs of the selected security properties, and through several levels of normalization and aggregation (as shown in Figure 8), computes the Security Index of the analyzed application, which reflects its internal security level.

Table 4 The security properties (i.e., vulnerability categories) that are supported by the proposed security assessment model (SAM), along with the programming language for which support is provided by the SDK4ED platform.

Vulnerability Category	**Description**	**Programming Language**
Null Pointer	This security property provides rules the violation of which are indicators of null pointer dereference. A null pointer dereference can cause a program to crash or behave unpredictably.	Java, C, C++
Assignment	This security property provides rules for validating the secure variable assignment and declaration. For example, local variables should be initialized and declared as final, otherwise they may be targets for injecting malicious payloads.	Java, C, C++
Exception Handling	This security property provides rules for validating whether exceptions are handled properly by the software progam. If an exception is not properly handled, it may lead to important security-related issues, including system crash, exposure of sensitive information, and repudiation issues.	Java, C, C++

Table 4 – continued from previous page

Vulnerability Category	Description	Programming Language
Resource Handling	This security property provides rules for validating the proper handling of the system's resources. Improper resource handling may have important consequences to the security of the system, potentially leading to degradation or even denial of service.	Java, C, C++
Logging	This security property provides rules for validating the proper utilization of the logging mechanism. Improper logging can lead to important security-relate issues, such us omission of important security incidents and disclosure of sensitive information.	Java
Misused Functionality	This security property provides rules for validating the correct utilization of functionalities that are provided by popular APIs.	Java, C, C++
Synchronization	This security property provides rules the violation of which indicate the existence of synchronization issues. Such issues include deadlocks and race conditions, which may lead to important security issues, including denial of service, unauthorized access, etc.	Java, C, C++
Overflow	This security property provides rules for checking the existence of buffer overflows. A buffer overflow is mainly caused by accessing the contents of a buffer, without previously checking its size. Buffer overflows are exploited by malicious individuals mainly for performing remote code execution, which is a critical security breach.	C, C++
I/O Issues	This security property provides rules for checking whether the I/O functions that are provided by C and C++ programming languages are used in a secure way.	C, C++
String Issues	This security property provides rules for checking whether there are instances of misusage of C-style strings. A notable example of such type of issues is the usage of the strcpy() method in C/C++ programs, which is known to be insecure and needs to be avoided.	C, C++
Dead Code	This security property provides rules for checking whether instances of unreachable code, unused functions, and unused variables exist in the source code of the analyzed application.	C, C++

To enhance the practicality of the model, it has been implemented as a web service that is accessible through the SDK4ED platform. The parameters of the model (i.e., its thresholds and weights) were carefully derived in order to enhance the reliability of the produced model. More specifically, the thresholds were derived through the application of benchmarking techniques on a dataset of 100 real-world and widely-used software products, which were retrieved from the Maven Repository. The weights of the model were derived based on a multi-criteria decision-making techniques (e.g., (Saaty 2008)) so as to reflect the knowledge expressed by the Common Weakness Enumeration (CWE)[6].

The evaluation of the model was based on a set of 150 open-source software projects, as well as on the test cases provided by the well-knwon OWASP Benchmark[7]. The proposed model is the only model that was built and evaluated on such a large pool of data (i.e., 250 real-world software applications, comprising

[6] https://cwe.mitre.org/

[7] https://owasp.org/www-project-benchmark/

approximately 20 million lines of code), whereas no other practical security model is available on the market or in the literature that can be used in practice (Siavvas et al. 2021).

The Security Assessment Model (SAM), apart from the high-level quantitative security indicators, also enables the identification of low-level security problems that a program may have. In fact, the transparency and hierarchical structure of the proposed model enables root-cause analysis, allowing the developers to pinpoint problematic aspects of the analyzed application that must be optimized. In particular, the developers can start from the Security Index, and, following a top-down inspection approach, they can detect those Security Characteristics and Security Properties that received the lowest score by the model. Hence, the developers can start their refactoring activities by fixing the static analysis alerts that correspond to a Security Property with a low score, instead of fixing them in an arbitrary manner. The toolbox also provides details about the alerts that are identified, including their location in the source code, their severity, and their type, along with external links with further information and potential fixes.

Vulnerability Prediction

The purpose of vulnerability prediction is to identify security hotspots that reside in software applications, i.e., software components (e.g., methods, classes, etc.) that are likely to contain vulnerabilities. This is achieved mainly through the construction of vulnerability prediction models (VPMs), which are machine learning (ML) models that judge whether a given software component is likely to contain a vulnerability or not, by searching for vulnerable patterns in its source code. The vast majority of the VPMs that have been proposed in the literature so far are based either on text mining (i.e., lexical analysis of the source code) or software metrics. However, none of the existing models have demonstrated a sufficient balance among accuracy, practicality, and performance.

To this end, as part of the Dependability Toolbox of the SDK4ED platform, we propose novel vulnerability prediction models, based on Deep Learning (DL), text mining, and software metrics (Filus et al. 2021b). More specifically, with respect to text mining, we considered the Bag of Words (BoW) approach as the main way to represent the source code in a numerical manner. According to the BoW approach, the source code of a specific component is represented by a vector, which contains each token (i.e., keyword) retrieved from the source code of the component, along with its corresponding frequency (i.e., number of its occurrences inside the source code of the component). With respect to software metrics, popular metrics including complexity, coupling, and cohesion were considered, since they have been found to be closely related to the existence of vulnerabilities in software (Shin et al. 2011; Chowdhury and Zulkernine 2011).

The high-level overview of the proposed VPMs is illustrated in Figure 9. As can be seen by Figure 9, text mining and static analysis are employed in order to derive the BoW and the software metrics of a given software component (i.e., class). A Convolution Neural Network (CNN) is used in order to generate more meaningful features from the BoW of the component. The Random Neural Network (Gelenbe 1989), a spiked Neural Network that mimics human brain function, is used as a bonding model for combining both software metrics and the new features acquired after processing the BoW with the CNN model. The produced models were able to detect vulnerable components with sufficient accuracy, whereas the

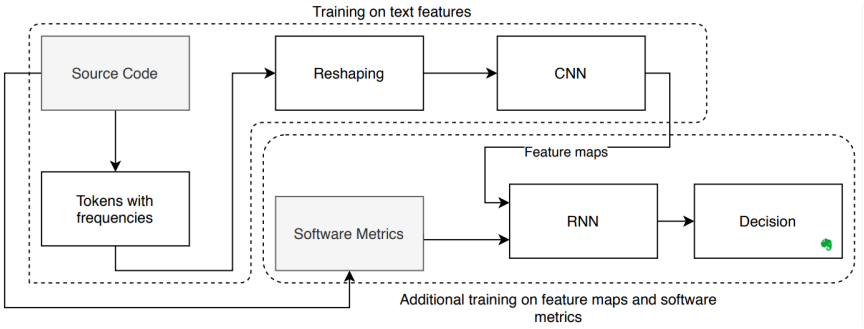


Fig. 9 The high-level overview of the Vulnerability Prediction Models (VPMs) of the SDK4ED Dependability Toolbox.

lightweight nature of the RNN lead to better performance compared to its counterparts, rendering it an attractive alternative. In addition to this, to the best of our knowledge, this is one of the very few attempts in combining software metrics with text features for building vulnerability prediction models.

The identification of security hotspots is highly useful for developing more secure software. It allows developers and project managers better prioritize their testing and fortification efforts, by allocating limited test resources to high-risk (i.e., potentially vulnerable) areas. For instance, more exhaustive code reviewing and dynamic testing can be applied to the potentially vulnerable components, in order to increase the probability of detecting and removing actual vulnerabilities.

4.3.2 Software Reliability

Reliability is a critical aspect for long-running software applications, especially for time- and energy-critical embedded software applications, since failures may lead to the re-execution of a large number of operations, inducing, in that way, significant overheads with respect to execution time and energy consumption. In order to avoid excessive re-executions and enhance the reliability of the program, the Checkpoint and Restart (CR) mechanism is widely used, which periodically keeps a safe copy of the program's execution state (i.e., a checkpoint), and uses it to restart the program in case of a failure.

Among the existing CR mechanisms, the application-level CR (ALCR) is the most efficient, as it leaves the smallest memory footprint, since it allows developers to define explicitly what data to be stored in each checkpoint. Although several ALCR libraries have been proposed over the years, which facilitate the insertion of checkpoints in long loops since they are considered the main source of failure-related re-executions, none of them provides suggestions for the selection of the intercheckpoint interval, i.e., the interval between two successive checkpoints. However, the arbitrary selection of the checkpoint interval may negatively affect the performance and the energy consumption of the application, due to the accumulation of checkpointing-induced overheads, which is critical for embedded systems.

To this end, as part of the Dependability Toolbox, the SDK4ED platform introduces novel mathematical models for modelling the overheads induced both

by failures and by the CR mechanism itself, and for computing the optimum intercheckpoint interval, i.e., the interval between two successive checkpoints that achieves a satisfactory balance among reliability, energy consumption, and performance (Siavvas and Gelenbe 2019b,a; Gelenbe et al. 2020; Gelenbe and Siavvas 2021). In particular, starting from first principles, we develop a mathematical model to estimate the average execution time and energy consumption of a program with a long loop that operates in the presence of failures, with and without the adoption of application-level checkpointing. This model is used to compute the checkpoint interval that minimizes execution time, energy consumption, or a combination of those two parameters, based on user preference. The analysis led to a closed-form expression of the optimum checkpoint interval, which is given by the following formula:

$$y^* = -\frac{1}{\ln a}[W(\frac{B-A}{e.A}) + 1] \tag{1}$$

In the above equation, y^* is the optimum number of instructions that should be executed between two successive checkpoints, which leads to the minimization of the overall cost (expressed in terms of execution time and/or energy consumption). The parameter a corresponds to the success rate of an instruction, i.e., the probability of an instruction to be executed without failure. The $W()$ function is the well-known Lambert w function (Lambert 1758; Euler 1783). The A and B parameters are cost parameters that are given by the following formulas:

$$B = B_0 + \frac{B_1 Y}{2} \tag{2}$$

$$A = b_0 + \frac{c + b_1}{1 - a} \tag{3}$$

In the equations above, B_o corresponds to the initialization cost of establishing a checkpoint, whereas B_1 is the cost of generating a checkpoint. In addition, b_0 corresponds to the cost of re-initializing the program from the most recent checkpoint after a failure occurs, whereas b_1 is the cost of having to empty the memory from the failed instructions. Finally, c corresponds to the average cost of a given instruction of the program.

As already mentioned, the y^* is the checkpoint interval that minimizes the overall execution cost. We used the generic term cost in the description of the formula, since the above formula can be used either for minimizing the execution time of a program or its energy consumption. It can be also used, as explained later, for minimizing a cost function that combines both quality parameters. This is expressed in the above equations through two parameters α and β that are used for the computation of the B_0, B_1, b_0, b_1, and c costs. In particular, these costs are computed as follows:

$$B_0 = \alpha B_0^c + \beta B_0^e, \tag{4}$$

$$B_1 = \alpha B_1^c + \beta B_1^e, \tag{5}$$

$$b_0 = \alpha b_0^c + \beta b_0^e, \tag{6}$$

$$b_1 = \alpha b_1^c + \beta b_1^e, \tag{7}$$

$$c = \alpha c^c + \beta c^e \tag{8}$$

The e exponent in the equations corresponds to the energy cost factor, whereas the c corresponds to the execution time cost factor. For instance, c is the cost of a given instruction. c^c corresponds to the average execution time of a given instruction, whereas c^e corresponds to the average energy consumption of a given instruction. These parameters are determined at the beginning, based on the hardware characteristics of the embedded platform (e.g., CPU architecture, power supply, etc.). It should be also noted that when a combination of the two parameters (i.e., execution time and energy consumption) is desired, the values should be expressed in a "per unit" manner, in order to get rid of the computation units.

The α and β parameters are used in equations (4)-(8) to define the importance of energy consumption or execution time for the calculation of the optimum checkpoint interval through equation (1). In particular, if $\alpha = 0$ and $\beta = 1$, exclusive focus is given to execution time, and (1) computes the checkpoint interval that minimizes exclusively the execution time of the application. This is important for time-critical applications, which need to provide their results in a specific time frame, and therefore the performance is their major concern. Similarly, if $\alpha = 1$ and $\beta = 0$, exclusive focus is given to energy, and (1) computes the checkpoint interval that minimizes the energy consumption of the application. This is important for energy-critical applications, which need to operate with the minimum energy footprint. Finally, if $\alpha \neq 0$ and $\beta \neq 0$, then y^* is the value that achieves a balance among execution time and energy consumption, as expressed by the user through these parameters. This is important, as it allows the model to be adapted to the specific needs of an application.

4.4 Forecasting Toolbox

The main objective of the Forecasting Toolbox is to predict the future evolution of the three main quality attributes that are monitored by the SDK4ED platform, which are the Maintainability (in fact, TD), the Energy Consumption, and the Dependability of embedded software. The high-level overview of the Forecasting Toolbox is illustrated in Figure 10. The Forecasting Toolbox, as shown in Figure 10 provides three core functionalities, which are implemented as individual web services. A description of these services is provided below:

- TD Forecaster: The goal of this service is to forecast the TD Principal of a given embedded software product. More specifically, it predicts the future evolution of the total remediation effort (measured in minutes) that is required in order to fix all the code-level TD issues (e.g., code smells, bugs, code duplication, etc.) of an embedded software application, up to a desired point in time (selected by the user).
- Energy Forecaster: The goal of this service is to forecast the Energy Consumption of a given software product. More specifically, it predicts the future evolution of the total energy consumption (measured in Joules) of a given embedded software application, up to a future point specified by the user.
- Dependability Forecaster: This service is responsible for generating Security forecasts for a given software application. More specifically, it predicts the future evolution of the Security Index (see Sect. 4.3.1) of a given embedded software application, up to a desired point in time (selected by the user).

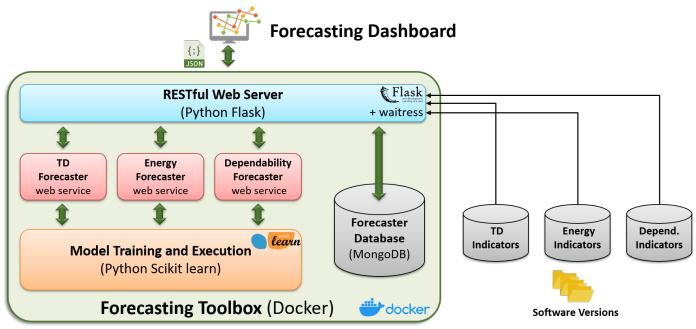


Fig. 10 The high-level overview of the Forecasting Toolbox

Towards building a solid basis for the Forecasting Toolbox realization, the suitability of various dedicated time series was investigated (Tsoukalas et al. 2019), whereas advanced machine-learning (ML) models have been introduced as part of the business logic of the toolbox (Tsoukalas et al. 2023), following the overall concept described in Tsoukalas et al. (Tsoukalas et al. 2020). For short-term forecasting, the time series models have demonstrated satisfactory forecasting performance up to 8 commits ahead, with the ARIMA model demonstrating the best results (Tsoukalas et al. 2019). For mid- and long-term forecasting, more advanced ML models were examined, including the Linear Regression, the Support Vector Regression, and the Random Forest (Tsoukalas et al. 2020, 2023). Our analysis led to the observation that non-linear ML models (such as Random Forest) are capable of capturing the future evolution of the three quality attributes supported by the SDK4ED platform (i.e., maintainability, energy consumption, and dependability) up to 40 commits ahead.

Advanced feature selection techniques were employed in order to select the best subset of indicators to be used for building less complex, but equally accurate forecasting models of the three quality attributes of choice. Several indicators were selected including *code smells* for Maintainability, *Cache Misses* for Energy Consumption, and *null pointer dereferences* for Dependability, among others. These indicators are produced by the three individual toolboxes, namely the Maintainability, the Energy, and the Dependability toolboxes, and constitute the required input for the corresponding forecasting models for providing their forecasts (see Figure 10).

To sum up, the Forecasting Toolbox integrates within its business logic various novel time series and ML forecasting models, built based on the results (i.e., monitored indicators) of the three core SDK4ED modules (see Figure 2). Depending on the forecasting horizon and the targeted quality attribute, it allows for the remote invocation of the most appropriate ones in order to provide meaningful forecasts to the front-end of the SDK4ED dashboard.

4.5 Decision Support Toolbox

The final toolbox of the SDK4ED Platform is the Decision Support Toolbox. Its goal is to support developers' decision making by ranking and sorting the

alternative suggestions from the rest of the SDK4ED Toolboxes. Through the Decision Support Toolbox output, the developer eventually chooses from the list of suggestions that the SDK4ED Platform makes to improve TD, energy, and dependability of the target application.

Each of the main three toolboxes analysis is individual, meaning that it does take into consideration other qualities. However, the final user of the SDK4ED Platform needs to evaluate all the suggestions (i.e., proposed refactorings) universally. Therefore, the proposed refactorings should include information about their impact on the rest of the aspects (energy, technical debt, and/or dependability). This information builds the 'design space' of the decision making and its production cannot be considered as trivial, as the three qualities are extremely dissimilar (particularly taking energy into consideration). As a result, it is very difficult to provide fine-grained estimations of a refactoring's impact without actually applying it and making a new analysis. Another requirement is to provide a solution that is as project- and platform-agnostic as possible.

The Decision Support flow is depicted schematically in Figure 11. After gathering all suggested optimizations from the rest of the toolboxes, we use an empirical model in order to form the design space. More specifically, after querying each of the proposed refactorings to the Decision Support database, the returned design space is analyzed by a Multiple Criteria Decision Making (MCDM) algorithm (Guo and Zhao 2017) (Lamprakos et al. 2022).

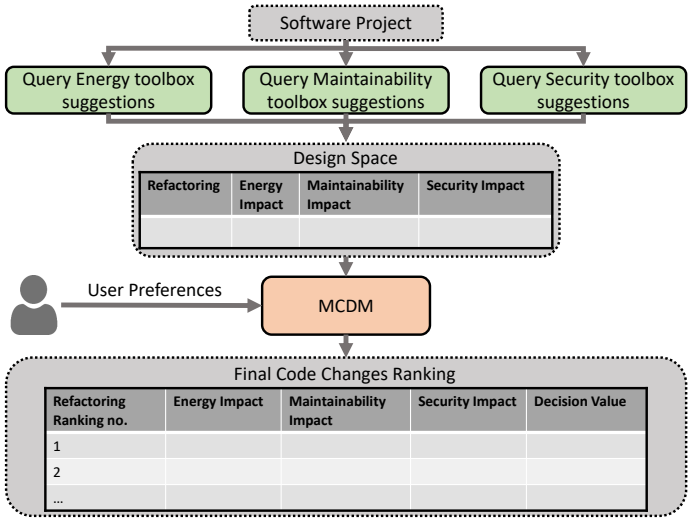


Fig. 11 Decision Support Toolbox flow

More specifically, as can be seen by Figure 11, for a given software project the refactorings that are proposed by the three toolboxes of the SDK4ED platform are initially retrieved and their impact on the three quality attributes of choice is determined based on a Look-up Table (LUT) that was devised in Lamprakos et al. (2022). The Decision Support Toolbox also takes as input the current values of the three quality attributes, namely maintainability, dependability, and energy

consumption, along with their projected values (as reported by the Forecasting Toolbox) and computes a 'future-to-current ratio' for each quality. These ratios represent the projected changes in quality metrics over time and are used to scale the Energy, Technical Debt, and Dependability components of each refactoring's impact. Subsequently, the user declares the relative importance of each one of the three quality attributes that are supported by the SDK4ED platform for the selected software project, and a trade-off analysis is performed based on a Multi-criteria Decision Making (MCDM) mechanism. The output of the Decision Support Toolbox is a list of those refactorings that better satisfy the requirements that were set by the user (i.e., the relative importance of the selected quality attributes of choice for the selected software project) and a detailed report of the impact that each refactoring has on each one of the three quality attributes that are supported by the SDK4ED platform. In that way, the user can have a better understanding of how each one of the suggested refactorings will affect the overall quality of the software project, allowing them to decide which refactorings to be applied to the selected software projects and which not.

5 Implementation of the SDK4ED Platform

The SDK4ED platform has been implemented in the form of a service-based platform based on Microservices Architecture. The reasoning behind our decision to opt for a service-based solution is the increased visibility that a service-based platform offers, the high accessibility by multiple users through local or wide area network, and the overall ease of use, as it enables the central installation of the platform avoiding in that way the tedious part of manually installing and configuring the tool for each user. The platform can be installed locally on the premises of an interested company and be accessible only through its private network or even over the Internet. For improving the end-user experience, a front-end, i.e., a dedicated user-friendly Graphical User Interface (GUI), the SDK4ED Dashboard, is also provided. In Figure 12, the Home Page of the SDK4ED Platform is illustrated.

The front-end of the SDK4ED platform, i.e., the SDK4ED Dashboard, was implemented utilizing cutting-edge technologies, in order to be up to date and ensure its longevity. In particular, it has been implemented in JavaScript, utilizing the React.js framework, which is one of the most popular frameworks for building cloud-based applications, in conjunction with MD-Bootstrap, ASP.NET Core, and PostgreSQL. Through the SDK4ED Dashboard, the user may define a new software project that they would like to analyze by providing its Git URL (which can be from any popular online Git Repositories, including GitHub, GitLab, and Bitbucket), perform a central analysis or analyze the project using features from specific toolboxes that are more interesting to them, and view the results of the analysis and potential recommendations in a visual form, by navigating to the pages of the toolboxes from the top menu (see Figure 12). Hence, the GUI acts as an interface between the user and the novel features of the SDK4ED platform that are provided by its toolboxes (see Section 4), which reside at the back-end of the platform.

The back-end of the SDK4ED platform consists of the five toolboxes, which are illustrated in Figure 2 and have been thoroughly described in Section 4. These toolboxes have been implemented as individual microservices, utilizing the Docker

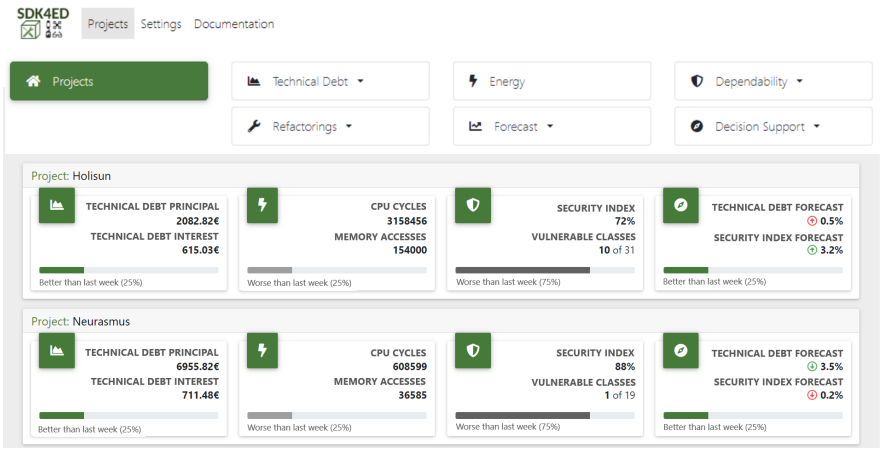


Fig. 12 The home page of the SDK4ED Dashboard.

Engine[8] for their deployment. In particular, each one of the toolboxes offers its novel features in the form of web services that are invokable through dedicated REST APIs, which are built as individual Docker Images that are deployed as independent Docker Containers. The reasoning behind the decision of building the toolboxes in the form of dockerized microservices is threefold. Firstly, from a development viewpoint, dockerized microservices enable the individual development of the toolboxes utilizing highly diverse programming languages, technologies, and frameworks, which can then be integrated directly into a unified platform, allowing different development teams with different backgrounds and expertise effectively collaborate. Secondly, from an exploitation viewpoint, each toolbox is a self-contained application that can be separately utilized or exploited. This enables an interested party to utilize only those components from the SDK4ED platform that they consider more valuable or interesting. Finally, the microservice approach enhances the reliability of the final solution, since the independent nature of the microservices enables the SDK4ED platform to continue its operation normally, even when one or more toolboxes are unavailable.

The novel features of the SDK4ED Platform can be utilized either through its dedicated GUI or by directly invoking the web services provided by its toolboxes. In order to facilitate potential users in setting up and utilizing the SDK4ED platform, a dedicated wiki page is available online[9].

Comment on the extensibility of the SDK4ED Platform: As already stated, the SDK4ED platform provides mechanisms (i.e., toolboxes) for analyzing and optimizing the quality of embedded software products, focusing specifically on three important quality attributes, namely maintainability, dependability, and energy efficiency, whereas it supports the analysis of applications written in three programming languages, namely Java, C, and C++, which are widely used in embedded systems. The decision to focus on these quality attributes and programming languages was based on the requirements that were elicited at the beginning

[8] https://www.docker.com/
[9] https://gitlab.seis.iti.gr/sdk4ed-wiki/wiki-home/-/wikis/home

of the SDK4ED project based both on the proposal document and the additional requirements that were expressed by the actual users of the platform, i.e., the use case providers, through a formal requirements elicitation approach that was carried out at the beginning of the project (Sas and Avgeriou 2020). In addition to this, the configurability of the platform, i.e., its ability to be configured in order for the user to (i) select specific analysis to be executed for a given application, and (ii) configure each analysis (toolbox) in order to better meet the specific needs of the application was a requirement that was also expressed by the industrial partners during the requirements elicitation phase and integrated into the final platform.

It is true that, apart from the three quality attributes and programming languages that are already covered by the SDK4ED platform, additional quality attributes (e.g., scalability) could be relevant to different types of embedded applications, whereas other programming languages (e.g., Python) are also utilized for their implementation. Therefore, it would be valuable for the SDK4ED platform to be customizable and extensible, allowing the easy integration of new toolboxes that provide analysis for other quality attributes and programming languages. However, since the extensibility and customizability of the platform was not a functional requirement defined during the requirements elicitation phase, no streamlined approach is currently provided for extending the platform, in a sense that no dedicated Application Programming Interface (API) or a formal integration approach is provided.

Despite this, the decision to adopt the Microservice-oriented Architectural Design Pattern for the design of the SDK4ED platform, significantly contributes towards its future extensibility. In particular, the back-end of the SDK4ED platform, instead of being a monolithic application, is a combination of standalone microservices implemented and deployed as independent Docker Containers. Each one of these microservices (i.e., toolboxes) is written in its own programming language and exposes its functionalities through its own RESTfull API (i.e., through GET/POST requests). Hence, should someone wishes to integrate a new toolbox (e.g., for analyzing a new quality attribute or provide support for another programming language), they would have to follow the following steps and satisfy their corresponding requirements:

1. **Step 1 - Microservice Implementation:** The toolbox needs to be implemented as an individual microservice. The developers are free to implement their toolbox in any programming language they prefer and with any framework(s) they desire. The only requirement is to enable the toolbox to internally perform a dedicated analysis of the selected software project and generate a report with the results of their analysis in a machine-readable format (preferably JSON). The toolbox must be implemented as an independent Docker Image and deployed as a standalone Docker Container.
2. **Step 2 - RESTfull API Implementation:** The toolbox must expose its features through a web-based API, preferably a REST API. The developers are free to implement as many endpoints as they consider necessary. As a minimum, the toolbox must provide an endpoint for initiating the analysis of a given software project that resides on an online version control and hosting platform (e.g., GitHub, GitLab, BitBucket, etc.) and a second endpoint for retrieving the results of the latest analysis of a desired software project. The

existence of a database for storing the analysis results is highly-recommended, but not required.

3. **Step 3 - Integration with the SDK4ED Dashboard (GUI):** A new web page needs to be included in the SDK4ED Dashboard, which will communicate with the REST API of the newly added toolbox. The developers are free to customize the dedicated web page for their toolbox however they wish, adding any visual elements they consider necessary. As a minimum, the SDK4ED Dashboard must invoke the endpoint for triggering the analysis of a given software project and the endpoint for retrieving the analysis results in order to present them on the web page. Details on how to install and update the SDK4ED Dashboard can be found on the wiki page of the SDK4ED.

Hence, from the above description it is clear that in case that a third-party developer has a toolbox that performs software analysis similar to the analyses performed by the SDK4ED toolboxes and is already web-based, the integration of the third-party component to the SDK4ED framework is a matter of containerization of the toolbox and implementation of a dedicated web page on the SDK4ED Dashboard that communicates with the toolbox. Since most of the software analysis tools that are implemented nowadays are usually web-based (e.g., SonarQube), the integration of a wide range of tools is potentially feasible, without much effort, as it will be reduced to the (i) containerization of their toolbox, and (ii) the implementation of a dedicated web page on the SDK4ED GUI.

Although the extensibility/customizability of the SDK4ED platform is not seamless at the moment, it should be noted that in the future we are planning to streamline the process of integrating a new third-party toolbox to the SDK4ED platform through the implementation of a formal integration pipeline and the provision of relevant detailed guidelines and step-by-step-tutorials on the process.

6 Evaluation

6.1 Use Case Demonstration

The major challenge of the SDK4ED platform is to provide practical solutions that can actually help project managers and developers build high-quality embedded software applications. Hence, the usefulness and practicality of the SDK4ED platform were evaluated in a real-world setting through three use cases on real-world commercial embedded software applications that are actively developed and maintained by the industrial partners (i.e., pilot providers) of the SDK4ED project. These applications come from three different application domains, particularly from the automotive, airborne, and healthcare domains. For reasons of privacy, the exact names of the applications that were utilized for the purposes of the use cases, as well as those of the involved industries, are not disclosed.

With respect to the evaluation setup, a unified approach was adopted for each one of the use cases. The SDK4ED Platform was deployed to the premises of the involved industries by their developers, based on detailed instructions that are provided on the SDK4ED Wiki page[10]. To ensure the correct setup and configuration of the platform, assistance by the actual developers of the SDK4ED platform was

[10] https://gitlab.seis.iti.gr/sdk4ed-wiki/wiki-home/-/wikis/home

also provided when it was considered necessary. Since a common requirement that was expressed by all the pilot providers at the requirements definition phase of the project was the platform to be highly configurable, and particularly to allow the user to focus only on specific quality attributes (as not all of them are equally important for each company and/or application) (Sas and Avgeriou 2020), as described in Section 5, the final platform enables the users to select only a subset of the monitoring and optimization mechanisms to be executed. Hence, each pilot provider initially declared the quality attributes that are more important for their application so that the analysis could focus mainly on them, and the rest to be treated supplementarily. The pilot providers used actively the SDK4ED platform for the course of around 1 and $\frac{1}{2}$ years, and by the end of the project they provided feedback with respect to the usefulness and practicality of the SDK4ED platform. Their feedback and evaluation results were reported in dedicated project deliverables (SDK4ED 2019a,b,c,d).

It should be noted that in the present use cases emphasis is given on the qualitative analysis of the SDK4ED platform. The quantitative evaluation of the novel features of the SDK4ED platform is not enough for measuring its potential success, as, although a specific feature may be correct and accurate, it may not provide any practical value to the end user. In fact, the correctness, accuracy, and reliability of the various novel features that have been proposed as part of the SDK4ED project and have been integrated into the final produced platform have extensively been studied and reported in their own scientific publications[11]. Hence, this is out of the scope of the present paper, which attempts to complete for the first time the overall analysis of the SDK4ED platform, by providing a qualitative evaluation of its novel features, through actual utilization by real companies that develop embedded software.

6.1.1 Automotive Use Case

For the automotive use case, an embedded software application that is running on Android-based Augmented Reality (AR) glasses (mainly smart glasses, such as Epson Moverio, Microsoft Hololens, etc.) was utilized, which enables remote maintenance and support of automotives by bi-directional audio and video streaming. In particular, its purpose is to connect an engineer or technician who is operating on-site, with engineers at the support center, in order to assist them with interactive support. The wearer of the glasses can stream to a desktop/mobile device what she/he sees, whereas they can also receive audio guidance from the other side. Since the application is running on smart glasses the technician has their hands free so that they can work on the maintenance activities in the meantime. Apart from audio/video communication, the application allows the end users to exchange media files, like drawings, manuals, schemas, and images with technical specifications. The application is a mid-size application that is written in Java programming language, as it runs on devices with Android OS.

The subject embedded software application faces a lot of challenges with respect to real-time operation, with the most critical being its performance, as the latency of the communication should be kept at a minimum level in order to ensure seamless and effective communication. Hence, its development is guided by this

[11] https://sdk4ed.eu/documents/

factor, and thus, its developers focus on source code changes and updates that maintain efficient real-time operation. However, the code changes that they make may affect the maintainability, and, in turn, the longevity of their application. Before the SDK4ED project, the company did not use any means of measuring the design-time quality of the subject application (and their applications in general). Hence, within the context of the project the company has been mainly interested in the technical debt monitoring and optimization features that are provided by the SDK4ED platform, in order to assess whether it can provide them with meaningful insights about the maintainability of their application, and in turn whether this information should be incorporated into their development process.

Maintainability: The subject application was analyzed with the TD monitoring and optimization features that are available by the SDK4ED platform, over the course of the 1 and $\frac{1}{2}$ years. In fact, four major versions of the subject applications were analyzed using the SDK4ED platform. The results of the analysis, as illustrated by the SDK4ED Dashboard, are depicted in Figure 13.

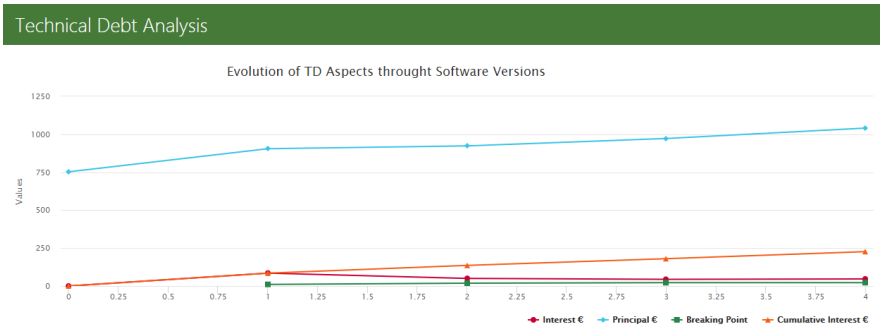


Fig. 13 The evolution of the Technical Debt aspects of the automotive embedded software application.

Figure 13 illustrates the evolution of four important TD aspects, namely TD principal, TD interest, Cumulative TD Interest, and the Breaking Point, over the four major releases of the application. As can be seen by Figure 13, the overall TD Principal is constantly increasing, starting from a value of 750 minutes (which translates into 487.5 Euros in maintenance costs) to a value of 984 minutes (which translates into 639.6 Euros in maintenance costs) in the latest version. As already stated, the TD Principal actually denotes the total time (and, in turn, money) that a company needs to spend in order to remove all the maintainability issues (i.e., TD liabilities) that reside in the source code. The same trend is observed in the Cumulative TD Interest, which increases from the value of 0 to a value of 241 in the latest version. In fact, according to the developers, the continuous increase of TD principal and the cumulative interest are expected, since the codebase of the application is becoming larger.

On the positive side, with respect to the TD Interest, which quantifies the additional costs that are introduced by not fixing the maintainability issues promptly (i.e., close to their introduction to the system), a declining trend is observed from the 2nd version and beyond, as can be seen by Figure 13. This can be attributed

to the fact that the development team (as they reported in (SDK4ED 2019c)) actually applied source code quality enhancements through code refactorings, in order to keep maintenance costs at low levels. This also justifies the slow increase observed in the Cumulative TD Interest, as well as the relatively large value of the Breaking Point. In fact, the Breaking Point indicates the version at which the application will be considered unmaintainable, i.e., the point in time at which the Cumulative TD Interest will become higher than the TD Principal. In the studied application, the Breaking Point was found to be 22. This means that, given the current conditions, in version 22, the cumulative interest will be higher than the corresponding principal, and the application will become unmaintainable. Nevertheless, since the project is currently in version 4 (based on the analysis), the breaking point lies well ahead in the future. However, the continuous monitoring practices on quality control and TD repayment strategies will push the breaking point even further in time.

The SDK4ED platform, apart from quantitative indicators of important TD aspects, also provides information that could be utilized for optimizing the maintainability of the application, by reporting potential TD items that need to be repaid (i.e., fixed), and potential components that may accumulate TD and need to be refactored. For instance, as can be seen by Figure 14, for the analyzed embedded software application, the SDK4ED Platform provides for each artifact (i.e., package or class), important information about their TD. For each artifact, the exact TD items are also reported (on user click), along with information on how to fix them. This information is very useful for the development team in order to decide where to focus their refactoring activities.

Show entries 10 Search Search

Artifact	TD-minutes	TD-currency	Bugs	Vulnerabilities	Duplications	Code Smells
com/holisun/arassistance	34	26	0	0	0	8
com/holisun/arassistance/SplashActivity	34	26	0	0	0	8
com/holisun/arassistance/adapters	38	29	0	1	0	7
com/holisun/arassistance/OnlineUsersAdapter	6	5	0	0	0	3
com/holisun/arassistance/ChatAdapter	22	17	0	0	0	4
com/holisun/arassistance/DocumentRecyclerViewAdapter	10	8	0	1	0	0
com/holisun/arassistance/models	24	18	0	0	0	6
com/holisun/arassistance/ChatModel	5	4	0	0	0	1
com/holisun/arassistance/OnlineUserModel	19	15	0	0	0	5
com/holisun/arassistance/services	115	88	0	1	0	18
Artifact	**TD-minutes**	**TD-currency**	**Bugs**	**Vulnerabilities**	**Duplications**	**Code Smells**

Showing 1 to 10 of 38 entries Previous 1 2 3 4 Next

Fig. 14 The TD indicators of the artifacts of the automotive embedded software

It should be noted that within the context of the present use case, an empirical study was performed (Tsintzira et al. 2019), in order to examine whether the

Table 5 The scores of the security properties of the latest version of the automotive embedded application

Property	Score
Resource handling	80%
Assignment	85%
Exception handling	80%
Misused functionality	72%
Synchronization	20%
Null pointer	85%

SDK4ED platform accurately captures TD aspects, i.e., whether the TD aspects reported by the SDK4ED platform are in line with those perceived by practitioners. To this end, several software components from the studied embedded software application were ranked based on their TD Principal, TD Interest, and TD Interest Probability, which were reported either by the SDK4ED platform or by the opinion of practitioners that work for the company. The two rankings were compared using the Spearman rank correlation coefficient, and a statistically significant positive correlation between these rankings was observed. This provides confidence that the TD apsects that are computed by the SDK4ED platform are in line with those expressed by the practitioners, and therefore can be reliable used for quantifying TD of embedded systems.

Although the Maintainability was the main focus of the present use case, the features of the SDK4ED platform for monitoring and optimizing the dependability and energy consumption of the selected embedded software application were also employed. This was done in order to ensure that the other critical quality attributes of the embedded application are also kept in a satisfactory level and no important issues can be detected. For reasons of brevity, and since they are not the main attributes of interest, a summary of their results is provided in this section. For the detailed results we refer the reader to the associated project report, which is publicly available (SDK4ED 2019c).

Dependability: With respect to software security, the Security Index of the studied application was found to be 72%, which is relatively high, indicating that no critical vulnerabilities can be found in the system. The Security Index was found to be around 72% across all the four studied versions of the application, whereas the scores of the Security Characteristics of Confidentiality, Availability, and Integrity were found to be above 70%, indicating that the code changes that were performed for either adding new features, or for reducing the maintenance costs did not affect its overall security level. The scores of the latest version of the low-level Security Properties that were computed by the model are presented in Table 5. As can be seen by Table 5, all the properties receive a high score (i.e., above 70%), with the only exception of the properties of Logging and Synchronization that were found to be 40% and 20% respectively.

However, after inspecting the detailed results the development team identified that the low score in the Logging property is caused by the fact that they do not utilize a formal logging library (like the Log4j), but a custom logging system, whereas the low score in the Synchronization issue was caused by the excessive use of Threads, which is, however, inevitable, due to the strong need of the application to ensure real-time communication with the lowest possible latency. None of those issues were considered critical, which is also in line with the model as

their low score did not affect the overall Security Index significantly. However, the development team, based on this feedback declared that in the future they will reconsider the adoption of a popular logging mechanism and will examine the security implications of the utilization of threads.

Energy Consumption: Finally, an energy analysis of the final version of the selected embedded software application was conducted using the Energy Toolbox, in order to ensure that the energy consumption of the final version of the application (after adding new features and reducing maintenance costs) is within acceptable levels. The energy analysis results, as reported by the Energy Toolbox, are presented in Table 6.

Table 6 The energy analysis of the latest version of the automotive embedded application

Memory (bytes)	CPU Load (%)	CPU Freq (kHz)	Context Switches (#)	Test Case Duration (sec)	System Calls (#)	Total Energy (Joules)
1733098	49.95	1374600	23604	5.93	278	12.39

In Table 6, several energy-related indicators are provided. The energy consumption of the final version of the application was found to be 12.39 Joules, which is well below the maximum acceptable level that was set by the development team, which is 20 Joules. Hence, although the energy consumption analysis was not the main focus of the company, being available on the platform, allowed the development team quickly determine whether the energy footprint of the final version of the application is within the acceptable levels.

6.1.2 Airborne Use Case

For the airborne use case, an embedded software application that is operating on Unmanned Aerial Vehicles (UAV) (i.e., drones) was utilized. The selected embedded software application is responsible for the piloting of a Vertical Take-Off Landing (VTOL) drone. More specifically, the application contains algorithms for remote and autonomous piloting of the drone, obstacle avoidance, navigation, and route planning. Algorithms for improving autonomy capacity are also provided. The application is running on a commercial off-the-self flying platform and has been implemented recently by an industrial partner of the SDK4ED project, which is a leading industry in aircraft production, defense, and aerospace. The application is a small-sized embedded software application, which has been implemented in C++ programming language.

The aforementioned embedded software application faces a lot of challenges with respect to real-time operation, with the most critical being energy consumption. The drone on which the application is running is operating on batteries and therefore is characterized by limited energy capacity. In fact, these batteries can support a flight of 30 minutes with only the engine running and no active payload consuming power. Any additional item like sensors or computing power will reduce drastically the flight time and thus the mission time. It is therefore of utmost importance that the embedded software is optimized, so as to use minimum energy.

Hence, the major quality attribute that the airborne industry would like to monitor and optimize through the SDK4ED platform is the energy consumption of the selected application, as it is critical for the duration and success of a drone mission. To this end, emphasis was given to the features provided by the Energy Toolbox of the SDK4ED platform. However, similarly to the other use cases, additional features were also used supplementarily, in order to ensure that no critical issues with respect to other critical quality attributes reside in the analyzed embedded application.

Energy Consumption: Initially, the Energy Consumption Estimation mechanism of the Energy Toolbox was utilized, which provides static estimations of the expected energy consumption of the analyzed embedded software applications on different hardware platforms. The results of the analysis of the selected airborne embedded software application are illustrated in Figure 15.

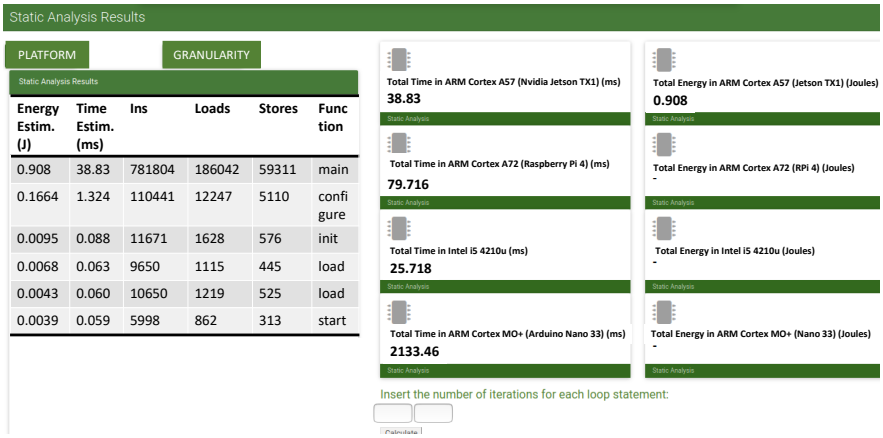


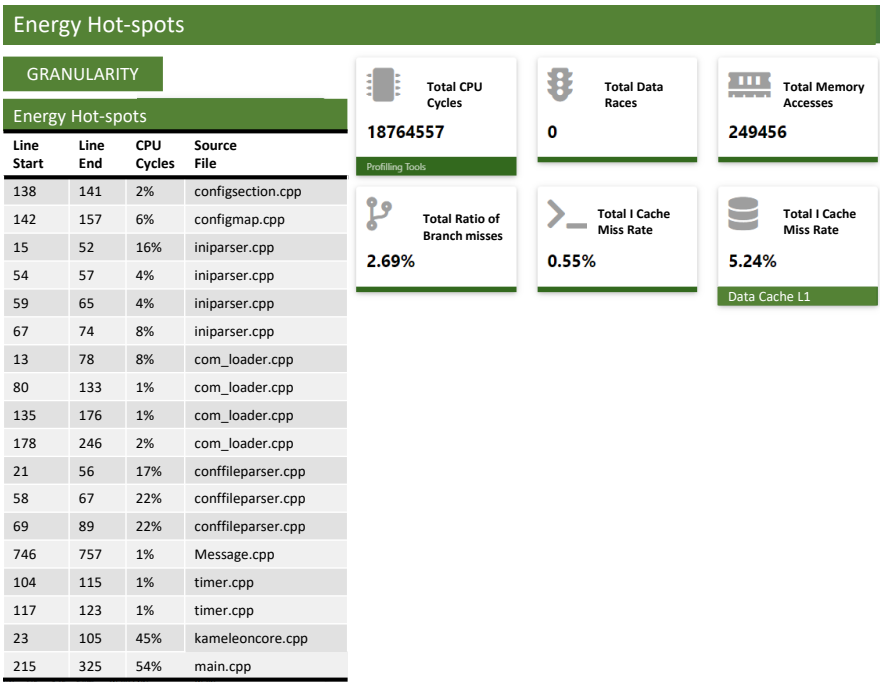
Fig. 15 The estimated energy consumption of the airborne embedded application on various hardware platforms

As can be seen by Figure 15, a detailed table is provided by the SDK4ED platform, which reports the estimated time and energy consumption of each function of the analyzed application, as well as of its loops (which are usually time and energy-consuming parts of the source code). In addition to this, the expected total execution time and energy consumption of the overall application for various hardware platforms are also reported. This allows developers to have a fast estimation of the expected performance and energy consumption of their application on various platforms, as well as to detect potential time-consuming and energy-hungry functions and loops, early enough in the development, even prior to having an executable version of the application, which is necessary for actual performance and energy measurements.

Figure 15 also shows the final energy estimation results for running the airborne embedded software application on different devices. Comparing the energy estimation of Nvidia Tegra TX1 with energy consumption retrieved from an integrated energy sensor in the actual device, we conclude that the Energy Toolbox estimation accuracy can be considered acceptable for the airborne use case. More

precisely, the actual energy consumption is around 1.4 Joules, while the Energy Toolbox estimates 0.908 Joules using only Static code analysis. Based on developers' feedback on these results, the SDK4ED Energy toolbox offers good assistance in the application development phase, as it provides a quick estimation of Energy Consumption without needing to implement the application on the actual hardware.

Apart from the static energy estimations, the Consumption Analysis mechanism of the Energy Toolbox was also employed, in order to measure the actual energy consumption of the application on the selected hardware platform, as well as for the identification of the energy hotspots, i.e., those code parts that consumed a lot of energy. Figure 16 presents the results of the Consumption Analysis component of the SDK4ED Energy toolbox for the analysis of the Airborne pilot use case.



Line Start	Line End	CPU Cycles	Source File
138	141	2%	configsection.cpp
142	157	6%	configmap.cpp
15	52	16%	iniparser.cpp
54	57	4%	iniparser.cpp
59	65	4%	iniparser.cpp
67	74	8%	iniparser.cpp
13	78	8%	com_loader.cpp
80	133	1%	com_loader.cpp
135	176	1%	com_loader.cpp
178	246	2%	com_loader.cpp
21	56	17%	conffileparser.cpp
58	67	22%	conffileparser.cpp
69	89	22%	conffileparser.cpp
746	757	1%	Message.cpp
104	115	1%	timer.cpp
117	123	1%	timer.cpp
23	105	45%	kameleoncore.cpp
215	325	54%	main.cpp

Fig. 16 The energy hotspots (i.e., functions and loops) of the airborne embedded application along with the reported energy indicators

The values of Energy indicators characterize the application's Energy Consumption, helping to further explore the characteristics of the program that consume more energy. The total number of Memory accesses is 249456. Most of the memory operations concern the initialization of the application. The data cache misses rate is 5.24%. These are mainly cold misses at the beginning of the application that do not lead to any cache-blocking optimization. In addition, the use case includes a small number of loops. According to the hotspot analysis, most of

the Energy is consumed in the main function and more specifically in a while loop, waiting for the user to give input.

The results are in line with the previously extracted static estimations. This shows that the static estimators can be utilized during the development for having some quick estimations of the expected energy consumption, whereas the dynamic estimations can be used afterwards, prior to the release of the application in order to verify the actual values.

Dependability: Reliability is also a critical parameter for drone applications. In order to enhance the reliability of the drone operation, the Checkpoint and Restart (CR) mechanism is utilized by the selected drone application. Since the drone application is an energy-critical application, and checkpoints are known to introduce overheads with respect to energy consumption, the Dependability Toolbox of the SDK4ED platform was utilized for selecting the optimum checkpoint interval, i.e., the interval that minimizes energy consumption. The results of the analysis are illustrated in Figure 17.

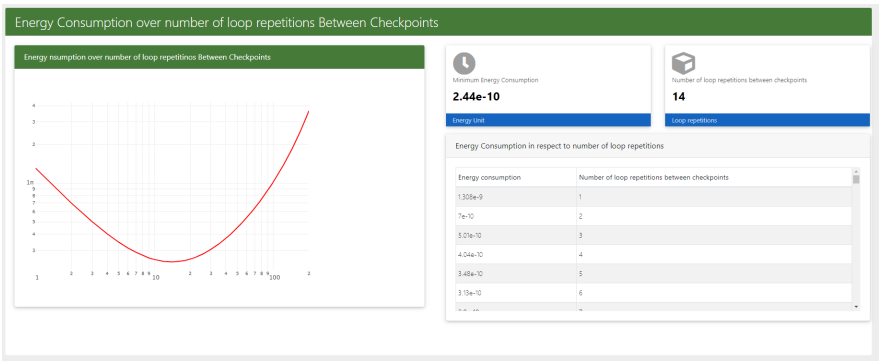


Fig. 17 The optimum checkpoint interval that minimizes the energy consumption of the airborne embedded software application

As can be seen by Figure 17, the checkpoint interval should be set to 14, in order to minimize the energy consumption of the application. This means that if we want to ensure that the energy consumption of the source code of the airborne use case is not affected by the checkpointing mechanism, we need to take a checkpoint every 14 iterations of the overall loop. The developers found this analysis useful and utilized this tool in order to set the checkpoint interval of the application. The utilization of the CR mechanisms with the selected checkpointing interval did not have an observable impact on the duration of the flight, indicating that the energy overhead of the CR mechanism is reduced to its minimum level.

With respect to the security level of the drone application, the Dependability Toolbox reported very high scores. In particular, the Security Index of the overall application was found to be 86%, which is very high. In addition to this, the security scores of the characteristics of Confidentiality, Integrity, and Availability were all found to be above 85%, indicating that the application highly satisfies these three critical security requirements. Finally, the security scores of the security properties that are reported by the security model, were found to be above 88%,

with the only exception of the Exception Handling property which received a low score of 22%. However, after a close inspection by the development team, the reported exception handling issues were not critical enough for causing harm to the system, which was also reflected by the security model, as it did not significantly affect the overall security index.

It should be noted that the development team of the airborne application is following a secure software development lifecycle (SSDLC) for the construction of drone applications, including the application used in the present use case, and therefore no important issues were expected to be found by the security monitors of the Dependability Toolbox. Indeed, the Dependability Toolbox assigned a really high security score to the analyzed drone application. This provides confidence for the reliability and accuracy of the proposed security monitors, as they were able to reflect the real case.

Maintainability: Finally, the TD Toolbox was utilized in order to compute the TD of the application and detect whether critical maintainability issues may reside in the source code of the drone application. The TD evolution analysis was out of interest for the airborne use case, since the application is recently developed, and therefore no sufficient number of versions/releases have been developed so far. The results of the TD analysis of the latest version of the airborne application are presented in Figure 18.

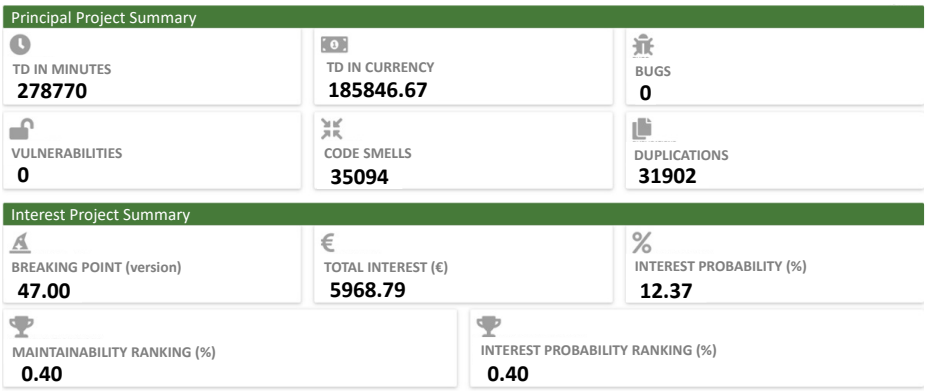


Fig. 18 The summary of the TD Principal and Interest of the airborne embedded software application

As can be seen by Figure 18, the TD principal was found to be approx. 193 days, which is only due to code smells and code duplications. No critical TD liabilities were detected in the application, which would require immediate care. Despite the fact that TD interest amount is quite large, the Breaking point is assessed at 47 versions from the current release, probably due to the very low amount of interest probability (approximately 12%). The scarce maintenance of the system renders TDM not the top priority for this pilot case.

A TD analysis of the artifacts of the analyzed drone application revealed that the vast majority of the artifacts have small TD Interest. Although no critical TD

liabilities were detected by the TD toolbox, specific suggestions for improvements were provided. In particular, through the artifact analysis, highly complex artifacts were identified and reported to the development team. In addition to this, design-level and code-level optimizations were also proposed. More specifically, the TD toolbox identified several code parts that could be extracted and implemented in the form of individual methods (i.e., exact method refactoring), as well as best on the TD New Code analysis (see Section 4.2.2), the TD Toolbox suggests the developers to "document Public APIs, in order to be used by customers" and use "Method names that comply to conventions", as they were the most frequent TD liabilities found in the system. The development team is planning to fix these issues steadily throughout the lifecycle of the applications since they were not observed to be critical that require immediate fix.

6.1.3 Healthcare Use Case

For the healthcare use case, we opted for an embedded software application that is running on implantable medical devices (IMD), developed by an industrial partner of the SDK4ED consortium. In particular, the selected software application is operating on an implantable neurosimulator that is used for seizure detection. The application includes functionality for receiving data from (ECoG/EEG) sensors periodically, performing FIR filtering, and deciding whether a seizure is detected or not in order to apply electrical stimulus via GPIO to suppress it. It is a small-sized application, due to the limited memory available in the implantable devices, which is written in C programming language.

Contrary to the previous use case, the healthcare application is a special case, since Implantable Medical Devices (IMDs) belong to a class of highly life-critical, resource-constrained, deeply embedded systems. These applications face significant challenges with respect to critical non-functional requirements, whereas any compromise may have devastating consequences to the safety of the end-user (health complications or even death). Energy efficiency is highly critical for IMD applications, as their long-term operation (for years or even a decade) should be ensured, due to the fact that battery replacement is not possible without surgery. The reliability and security of these applications are also of utmost importance, since the exploitation of a single vulnerability or an unexpected failure, may put in danger the safety of the patient. Finally, due to the high maintenance costs of healthcare applications in general, their maintainability is also a concern for the development teams.

Hence, the development team of the subject healthcare application was interested in monitoring and optimizing horizontally all three quality attributes that the SDK4ED platform supports, namely the energy efficiency, the dependability (i.e., security and reliability), and the maintainability of their application. To this end, they actively utilized all the toolboxes of the SDK4ED platform during the development of the application, including the decision-support mechanism that computes the impacts of the proposed refactorings on the targeted quality attributes. In fact, this was the only use case that declared the need for achieving a trade-off among all three quality attributes, which was important for us to judge the usefulness of the code refactoring impact calculator based on trade-off analysis. In the rest of this section, the most notable results of the application of the various toolboxes on the selected IMD application are presented.

Energy Consumption: The first version (v1.3) of the IMD application, analyzed by the SDK4ED Energy Toolbox, is an emulator of all the IMD components, which was running on a Linux PC. Figure 19 presents the resulted Energy indicators, while Figure 20 shows the estimated energy and the identified hotspots. These results show that the application has a relatively small number of Memory accesses (1546270). Most of the cache misses refer to the application initialization. The presented data races are caused by the use of I/O C libraries (printf, scanf) and the identified hotspots include locks waiting for the rest of the application's threads while one hotspot corresponds to a loop statement. It is worth mentioning that the toolbox proposes this loop as a candidate block for acceleration because due to its large instruction parallelism, which is combined with a few control operations while the most of the operations are memory accesses (not to the same address). However the small number of iterations would lead to more delays in data transmission, so acceleration was not applied.

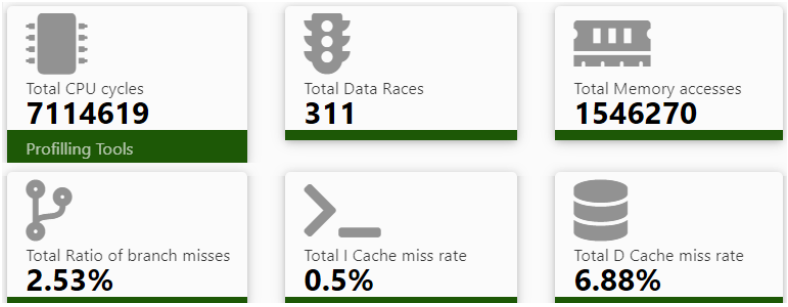


Fig. 19 The SDK4ED Energy Indicators of the healthcare embedded software application

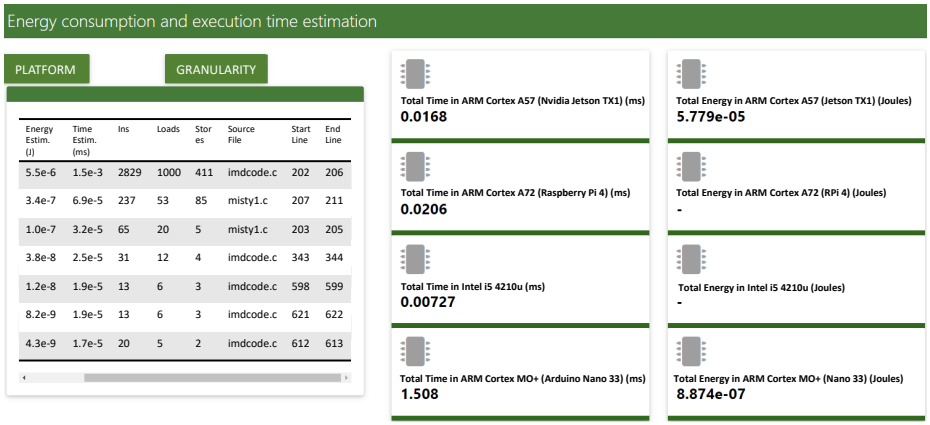


Fig. 20 The SDK4ED Energy Consumpion Estimation and Hotspots Identification results of the healthcare embedded software application

The next versions of the use case, which were running on the actual hardware devices, were analyzed using the static analysis energy estimation component of the SDK4ED Energy toolbox. As mentioned in Section 4.1, this component was designed to give an easy and fast way to estimate the energy. Also, using this component we are able of comparing the use of different platforms. According to the results presented in Figure 20, selecting the microprocessor ARM Cortex M0+ leads to a more energy-efficient solution (of course with a penalty on the execution time). The energy savings compared to using the more complex ARM Cortex A57 can reach up to 98%. The user of course, as mentioned in 4.1, can add more platforms by following the relevant guidelines.

The Energy Toolbox can also report the estimations for all the application versions by retrieving the previous analysis results from the database. Figure 21 shows the estimated energy for each application version. The estimated energy is always lower for using ARM Cortex M0+. An increase observed around the middle of the project's history is due to the addition of a new cryptographic function. The function was replaced in the next version by a special peripheral. As a result, the CPU energy was reduced by 95%. It is worth mentioning that the Energy toolbox only analyses software and thus it does not include the energy of hardware peripherals (Marantos et al. 2022a).

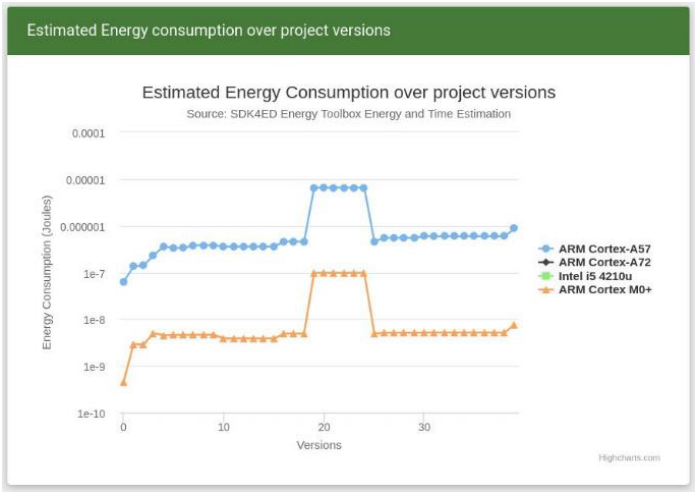


Fig. 21 The evolution of the energy consumption of the healthcare embedded software application over its versions

Maintainability: Three main TD facets are utilized in order to assess the maintainability of the healthcare application, namely the TD Principal, the TD Interest, and the TD Interest Probability. In the IMD application, the value of the TD Principal was characterized by significant variation across the various source code files of the application. The 'reader.cpp' file had the highest TD Principal, demanding $302 to fix 29 identified code smells. The TD Interest of the 'reader.cpp' file was found to be $9.68, which reflects the additional cost of maintaining this file, due to not fixing the identified issues at first place. Finally, the TD Interest

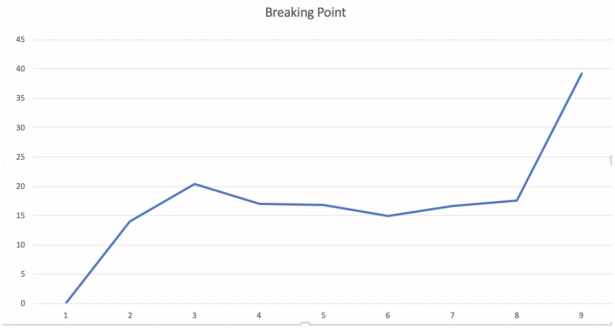


Fig. 22 Evolution of Breaking Point for the healthcare embedded software application

Probability of the aformentioned file was also found to be really high (i.e., 0.8), indicating that the file is changing frequently during the development of the IMD application, which renders it a potential source of TD accumulation. All these indicators are valuable for informing the development team about a specific source code file that poses a risk in the maintainability (and the associated maintenance costs) of the broader application, allowing them to make informed decisions on how and when to repay its TD.

The Maintainability toolbox of the SDK4ED platform can also report source code files, which are characterized by relatively high TD Principal, but low TD Interest and/or interest probability. These files should be assigned a lower priority for refactoring and fortification activities, since their low TD Interest and change frequency do not impose the same risk to the overall maintainability of the IMD application, compared to the risk imposed by frequently changing files, especially of files with high interest. Hence, the Maintainability Toolbox enables the developers to prioritize their TD repayment activities by focusing on more critical (from a TD viewpoint) files in a given time frame, and subsequently moving to the less critical ones. The evolution of the Breaking Point is of high interest for the IMD application (Fig. 22). The Breaking Point corresponds to the point in time at which the application becomes unmaintainable, i.e., the accumulated TD Interest of the application becomes higher than its TD Principal. As can be seen by Figure 22, the breaking point of the IMD application lies for most of the analyzed versions 20 versions ahead, while its value reaches 40 for the last version, indicating a rather healthy status from a maintainability viewpoint. Hence, based on the above analysis, although the IMD application contains TD liabilities (e.g., code smells), its overall quality can be considered high, not indicating the need for immediate TD repayment activities to be carried out. The development team is not expected to face unbearable maintenance costs in the projected future. This was very helpful for the development team of the IMD application, as it allowed them to focus on other critical aspects of their application, including energy and dependability, without losing time in fixing maintainability-related issues, as they were not found to impose any critical risk.

Finally, a separate TD analysis of the medical and security components of the IMD applications was performed (Siddiqi et al. 2021), in an attempt to examine whether the security-related features that are added in the application have an impact on the overall maintenance cost of the broader system. The analysis led

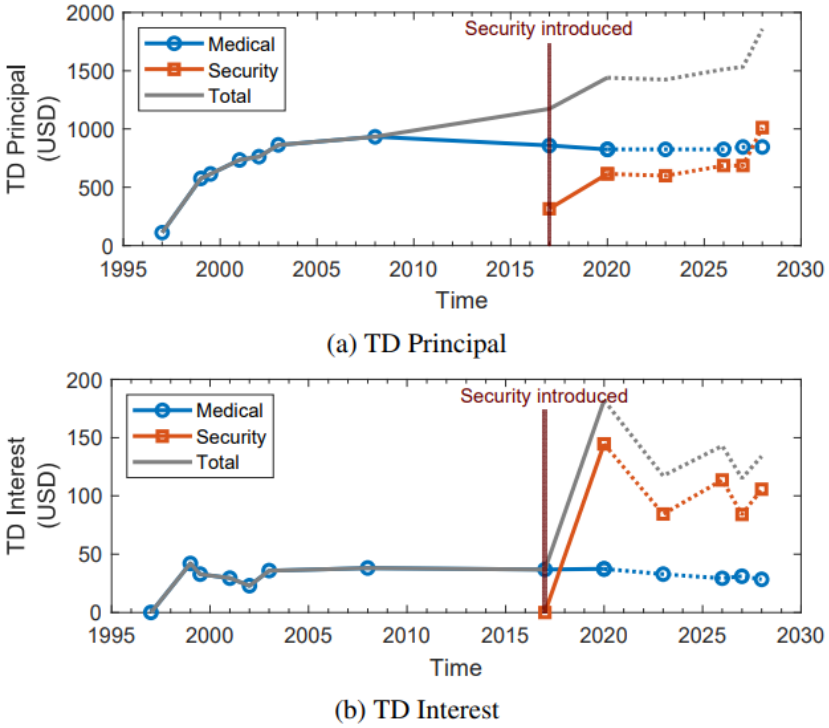


Fig. 23 The evolution of the TD Principal of the two main components (i.e., the medical and security components) of the healthcare embedded software application

to the observation that the TD Principal of the medical part of the IMD application remains stable over time, as opposed to the TD Principal of the security component, which follows an upward trend. Hence, the TD Principal of the overall IMD application follows the evolution of the TD Principal of the security component (see Fig. 23). This indicates that the inclusion of security features that are meant to strengthen the resilience of the IMD application against malicious attacks, are expected to significantly affect its maintainability, and therefore a sufficient trade-off between security and maintainability should be achieved. Such analysis initiated a research line within the company considering the security costs in the IMD domain, urging for more cost-efficient solutions at the software level.

Dependability: The IMD application was initially evaluated with the Dependability Toolbox of the SDK4ED platform, in order to assess its overall security level. In Figure 24, the results of the analysis that was performed by the security model of the Dependability Toolbox are illustrated. As can be seen by Figure 24, the *Security Index* of the IMD application is very high (i.e., 93%), and the scores of the three *Security Characteristics* of the model (i.e., *Confidentiality, Integrity, and Availability*) were found to be above 85%. With respect to the low-level *Security Properties* that are provided by the security model, almost all of them were found to have a really high score (above 94%). The only property that received a low score (i.e., 36%) is the *Dead Code* property, but this property is not equally

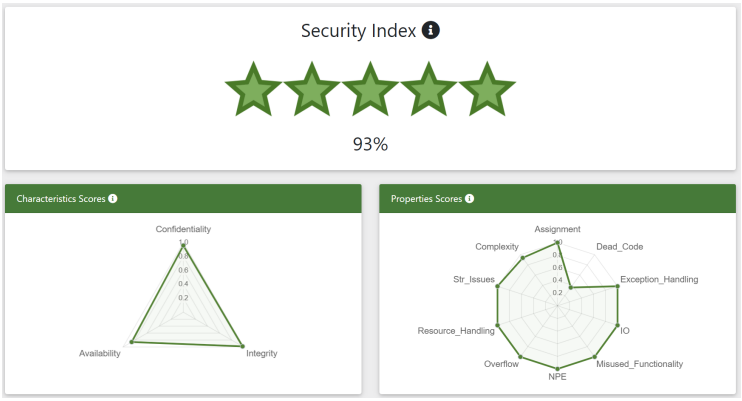


Fig. 24 The security assessment results of the healthcare embedded software application

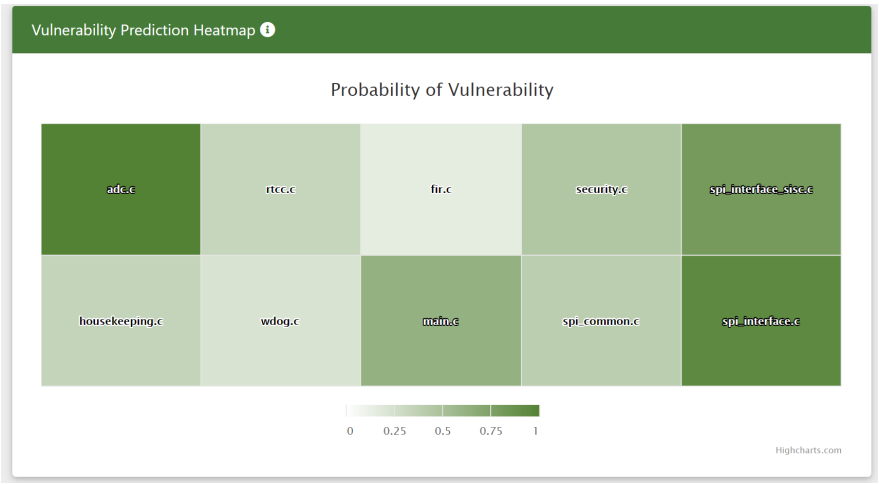


Fig. 25 Vulnerability Prediction Results for the healthcare application use case

critical to the other properties of the model, which is also reflected by the fact that the overall *Security Index* was not affected significantly.

In Figure 25, a heatmap is provided, which demonstrates the analysis results of the vulnerability prediction models of the Dependability Toolbox. In the presented heatmap, the rectangles correspond to the various source code files of the analyzed application and the color of the rectangle denotes how likely it is for the associated file to contain vulnerabilities. From Figure 25, it can be seen that the IMD application has three security hotspots, which are the abc.c, the spi_interface.c, and the spi_interface_sisc.c files, as they exhibit a high *vulnerability score* (i.e., above 0.8 in all three cases). Therefore, the development team can focus their security-related testing and fortification activities on these three source code files, increasing the chances of finding actual vulnerabilities.

The evolution of the security index over the nine major releases (i.e., versions) of the IMD application is also illustrated in Figure 26. As can be seen by this Figure 26, the Security Index is very high in all versions ranging between 93%

and 96%. It should be noted that prior to the release of each version, the source code of the application was evaluated utilizing the Security Assessment Model and the Vulnerability Prediction mechanisms of the Dependability Toolbox, and any detected critical security issues were fixed. During this process, two critical buffer overflow vulnerabilities were detected by the Dependability Toolbox, one before the third and another one before the seventh release of the application. The development team verified that those issues were actual vulnerabilities and corrected them before releasing their version on their Git repository. Therefore, the Dependability Toolbox enabled the prompt identification and elimination of vulnerabilities prior to the release of the application, which is a highly desirable aspect of security evaluation mechanisms, which acts as a success criterion in the literature (Mohammed et al. 2016).

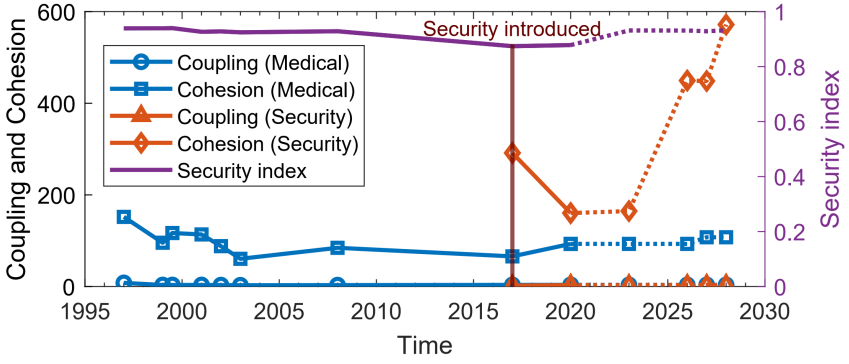


Fig. 26 The Security Index of the main releases of the healthcare embedded software application. Relevant TD indicators are also illustrated (Siddiqi et al. 2021).

Regarding the reliability of the IMD application, the Optimum Checkpoint Interval Recommendation (OCIR) mechanism was utilized, in order to select the optimum checkpoint interval for computational loops that reside in the source code of the IMD application. Figure 27 shows the results of the OCIR mechanism for the most computational loop of the healthcare use case. According to the OCIR mechanism, as shown in Figure 27, in order to minimize the execution time of the application a checkpoint should be established every 200 loop iterations, whereas a checkpoint should be generated every 30 loop iterations, if the goal is to minimize the energy consumption of the application.

Forecasting: The results of TD (i.e., Maintainability) forecasting for a forecasting horizon of 10 commits are illustrated in Figure 28. The green line corresponds to the actual evolution of the TD Principal of the IMD application until its latest commit, whereas the red line corresponds to the projected evolution of the TD Principal for the next 10 commits, as computed by the forecasting models. As shown in Figure 28, although the current value of the TD Principal of the IMD application is 5 days (which reflects the days that are required by the development team for fixing the underlying TD issues), this is expected to increase in 10 commits from now to the value of 8 days, which corresponds to an increase of 67%. his information helped the developers allocate the resources needed to quickly repay

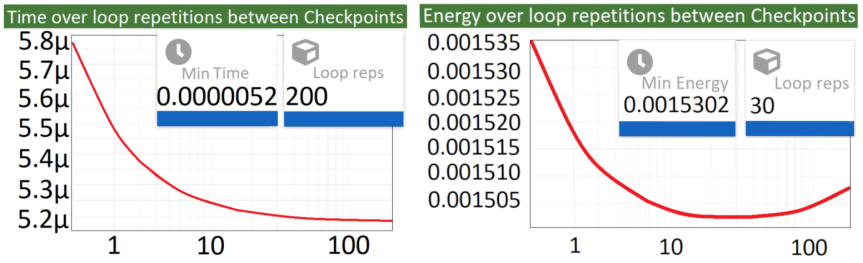


Fig. 27 Calculation of the optimum checkpoint interval that minimizes the execution time (left) and the energy consumption (right) of the healthcare embedded software application.

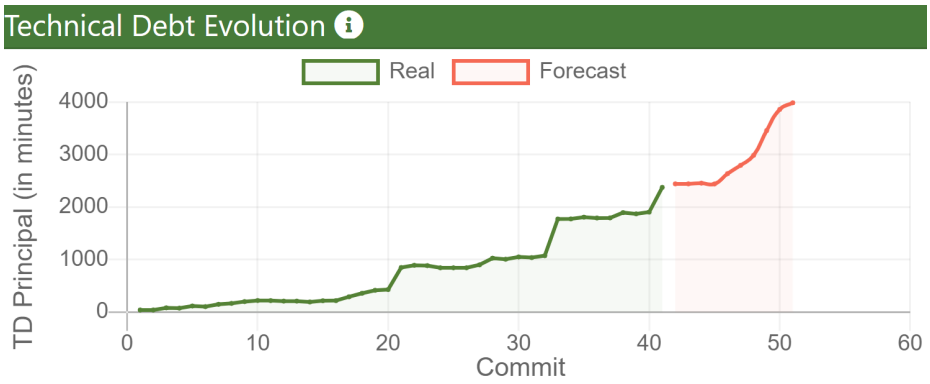


Fig. 28 TD Forecasting Results for the healthcare embedded software application

TD. In fact, these results are in line with the results of the TD analysis performed by the Maintainability Toolbox (and presented in Section 4.2), since, despite the fact that the TD Principal of the IMD application is expected to increase, the projected value of its TD is not considered to be concerning. With respect to the energy consumption and dependability, a slight increase is expected to be observed for the former, whereas no big changes are expected for the latter (i.e., the overall *Security Index* is expected to remain relatively stable).

Decision Support: The output of the SDK4ED Decision Support toolbox analysis for the Healthcare use case is depicted in Figure 29. A total value (right plot) is assigned to each of the refactorings proposed by the rest of the toolboxes. This value can be broken down into individual impacts (left plot) that the refactoring is expected to have, if implemented, on each of the three targeted quality attributes. Positive-signed values denote quality improvement. For instance, as shown in Figure 29. deciding to fix issues that belong to the "Resource Handling" category, will improve the Security and Energy Efficiency of the IMD application, whereas it is not expected to have any impact on its Technical Debt. Similarly, a decision to fix issues that belong to the "Overflow" category are expected to improve the security level of the IMD application, but it will have a negative impact to its maintainability and energy consumption, mainly due to the additional checks that need to be added to the source code, which lead to additional operations to be executed on the CPU and more complex source code. The user's preferences

drive the magnitude of each impact value. For example, if the user selects Energy as more important than TD, the colored bars on the left plot are expected to have bigger blue segments and smaller black segments than in the opposite scenario, where technical debt is declared as more significant one.

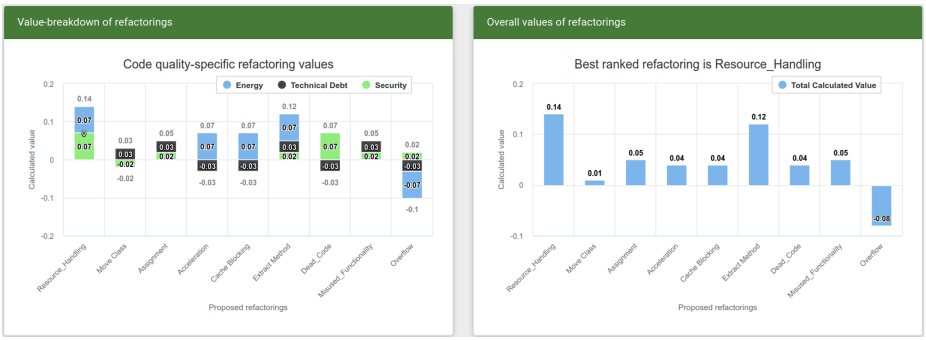


Fig. 29 The results of the Decision Support toolbox for the healthcare embedded software application

6.2 Industrial Study

Apart from the use cases that were presented in Section 6.1, whose main goal was to verify the usefulness and practicality of the SDK4ED platform in a real-world industrial setting, an empirical industrial study was also conducted, in order to verify whether the SDK4ED platform is relevant to the embedded software industry and can help in reducing the development cost of such applications. In fact, in an attempt to complement the feedback received by the industrial partners through the actual utilization of the platform via the three use cases (Section 6.1), questions with respect to the industrial relevance and usability of the platform (with emphasis on the TD toolbox, which measures the financial impacts) were also included.

More specifically, an embedded multiple case study was conducted, based on participants coming from the industrial partners of the SDK4ED consortium. In total, the study comprised 15 units of analysis (i.e., participants), coming from four companies, which are the three providers of the automotive, airborne, and healthcare use cases that were presented in detail in Section 6.1, and a fourth company that builds software applications for IoT systems, which is also part of the consortium. Similarly to Section 6.1, the companies are anonymized due to an NDA, but in Table 7, some information about them is provided.

The participants were given access to the platform for a 30-day trial period, during which they have been asked to involve the platform in their development routines (using the source code of their industrial projects), in the way that they perceive as most beneficial. An 1-day workshop was conducted at the beginning of this study, in order to present the platform to the participants and provide them with initial training. Upon the completion of the trial, we proceeded to data collection. Data collection comprised two methods, namely two survey sessions

Table 7 Participant companies demographics

ID	Application Domain	Country	Participants	Size
C1	Airborne	France	3	Large enterprise
C2	Internet of Things	Sweden	4	Small-medium Enterprise
C3	Automotive	Romania	6	Small-medium Enterprise
C4	Medical Applications	Netherlands	2	Small-medium Enterprise

and four focus groups. The purpose of the survey sessions and the focus groups was to evaluate the potential cost reduction that can be achieved by the platform and its industrial relevance, as well as the overall usefulness of the platform.

The surveys were based on online questionnaires that were shared with the participants who provided their responses in a 5 Likert-scale format. It should be noted that for evaluating the usability of the platform the relevant questionnaire was structured based on the System Usability Scale (SUS) (Brooke et al. 1996) approach. With respect to the focus groups, four focus groups were performed, one for each company. Each focus group was intended to last for 45 minutes (with each company - 3 hours in total); and was conducted using a teleconferencing platform.

For analyzing the results of the surveys and the focus groups, both *quantitative* and *qualitative analysis* were performed, based on the guidelines provided by Seaman (Seaman 1999). To obtain the *quantitative results*, we used the data obtained by the two surveys, by summing the scores assigned by all participants to a specific question, and used bar charts for their visualization. For usability, the total SUS score was provided, along with the most common scales for interpretation, in terms of acceptance, adjective, and grade. To obtain the *qualitative assessments*, the data from the focus groups were analyzed based on the quantitative content analysis (QCA) technique (Elo and Kyngäs 2008), which involved open coding, creating categories, and abstraction, and were visualized through alluvial diagrams. For more information about the overall qualitative and quantitative evaluation process, we refer the reader to (Ampatzoglou et al. 2022).

The results of the study led to some interesting observations. First of all, the results indicate that all the features provided by the SDK4ED platform for TD management, are expected to lead to development cost savings, especially with respect to future maintenance, as all of the relevant features received a score higher than 83%, with the *New Code Analysis Quality Control* to be the most promising one, having a sore close to 95%. With respect to the overall usability of the platform, the average score received by the participants was 76.8% and its average grade is B. Hence, the SDK4ED platform received a positive evaluation in terms of usability, rendering it acceptable for industrial usage. The focus groups revealed that, among the various features provided by the platform, the features that assist developers in writing cleaner new code are the most welcomed ones, as they allow them to keep the maintenance costs at specific levels. In addition to this, according to the participants, the monetization of the assessments is the most beneficial feature, as it enables them to understand the current and future development costs that they may face.

7 Discussion

In the present section, we attempt to summarize the main observations that we made with respect to the usefulness and practicality of the SDK4ED platform, by analyzing the use cases that were described in Section 6.1, as well as by considering the outcomes of the embedded industrial study presented in Section 6.2. It should be noted that for the derivation of the main observations that are presented in the present section, the evaluation reports that were delivered by the end of the SDK4ED project by the use case providers (reporting their experience with the SDK4ED platform along with their overall assessment (SDK4ED 2019a,b,c,d)), were also taken into account.

Application developers were able to easily monitor the quality attribute(s) of choice of their embedded software applications in a quantitative way frequently during the overall development cycle, thanks to the provided monitors.

In the automotive use case, the development team was able to monitor the evolution of the maintainability of their application, which was the main quality attribute of interest, over its four major releases. The maintainability was measured using the Technical Debt (TD) metaphor, and particularly the aspects of TD Principal, TD Interest, and TD Interest Probability. The monetization of their values enabled them to have an estimation of their maintenance costs, whereas the breaking point allowed them to understand whether their application is at risk of becoming unmaintainable in the future. The benefits of the monetization of the TD assessment were also highlighted by the participants of the industrial study that was presented in Section 6.2. In addition to this, the activities that were undertaken by the development team for reducing the maintenance costs between successive versions, were reflected with a drop in the overall computed TD interest. According to their evaluation reports, the SDK4ED platform was able to capture the TD concepts accurately. More specifically, as they report: "the indicators for principal and interest seem to be (at least) strongly correlated to our perception on principal", and "we are very eager to exploit the SDK4ED TD dashboard as part of our quality assurance processes, since it seems to be accurately reflecting our own perspective on software quality assessment." (SDK4ED 2019c).

Similarly, in the airborne use case, the developers were able to retrieve static estimations of the energy consumption of their application (which was the most critical attribute) during the design time, allowing them to gain a fast overview of the expected energy consumption and the expected energy-hungry hotspots (i.e., functions and loops). The runtime monitors enabled them to measure the actual energy of the application during its operation, and therefore to verify the accuracy of the static estimations. Notably, the development team reported in their evaluation document (SDK4ED 2019a) that "the indicators provide a good approximation of the reality".

Finally, in the healthcare use case in which all the three studied quality attributes were equally important and required monitoring, the Technical Debt indicators allowed the developers to monitor the evolution of their maintenance costs, both of the overall application and of its subparts (i.e., medical and security subcomponents), whereas the dependability toolbox enabled them to ensure that the security level of the various versions where high. The energy monitors (both static and dynamic) allowed them to monitor the energy consumption of the application

on different hardware platforms. The analysis of their application through the SDK4ED platform revealed that the security-related code of an IMD application is costlier with respect both to its development and maintenance, compared to the medical/core code of the application, and that it will dominate the cost in the future. More specifically, according to their reports (SDK4ED 2019b; Siddiqi et al. 2021), the SDK4ED platform allowed them to understand that "security-code TD amasses faster and will eventually overtake medical-code TD", and therefore "IMD economic viability will, thus, only be ensured if security-development efforts are allocated significant resources within the next decade".

Application developers were able to retrieve meaningful optimization suggestions from the SDK4ED Platform, which led them in improving the quality of their embedded software.

For instance, in the automotive use case, the development team identified critical TD liabilities that required their attention and took into account the reported code refactorings. According to the company, a dedicated quality manager was hired, who utilize the TD monitoring and optimization functionalities of the SDK4ED platform on a frequent basis, and assigned refactoring activities to the development team, based on the suggested optimizations. In the healthcare use case, the application of the security model of the dependability toolbox frequently during the development of the application led to the identification and elimination of two critical buffer overflow vulnerabilities prior to two major releases of the application. This allowed developers to eliminate critical security issues from their application at design-time, and maintain the security level of their application very high among all its versions (as shown in Figure 26). In the airborne use case, by utilizing the energy toolbox, the development team verified that they were able to detect energy-hungry functions and loops that reside in their application and that they were not aware of. They also verified that the most energy-hungry hotspots will be rewritten in order to reduce their energy footprint. In addition, the development team found really useful the optimum checkpoint interval recommendation mechanism of the dependability toolbox, as it allowed them to select the checkpoint interval that minimizes energy footprint.

The SDK4ED platform is highly configurable allowing its adaptation to custom user needs, as well as to highly diverse application domains.

As described in Section 5, the platform enables the execution of a central analysis of a given application, in which all the mechanisms of the SDK4ED platform are executed. However, the platform gives the opportunity to the end-user to perform custom analysis, i.e., to select only those features that are more interesting or relevant to their applications. For instance, in the automotive use case, primary emphasis was given on the maintainability of the application, in order to see the evolution of TD aspects of the application (e.g., TD Principal, TD Interest, etc.), as well as to detect TD liabilities that require repayment. In the airborne use case, specific emphasis was given on the energy consumption and reliability of the application, utilizing mainly the static and dynamic monitors of the Energy Toolbox, and the optimum checkpoint recommendation system of the Dependability toolbox. On the contrary, in the healthcare use case, since all three quality attributes were considered equally important by the development team, all the features of the SDK4ED platform were actively utilized during the development of the application.

Hence, the platform is highly configurable allowing end users to cherry peak those components that they consider relevant and interesting, enabling the analysis to adapt to custom user needs and application domains. Finally, its successful application on different use cases stemming from different domains and developed by different development teams further supports its ability to adapt to custom needs and domains. The high configurability and adaptability of the platform, and mainly the ability to focus on a desired set of quality attributes, was one of the main functional requirements that were expressed by all the industrial partners of the SDK4ED consortium, during the requirements definition phase of the platform (Sas and Avgeriou 2020).

At this point it should be stated that with the term highly configurable we refer to the ability of the SDK4ED platform to be properly configured (through proper modifications of its settings and/or parameters - without the need for source code interventions) in order for the performed analyses to be tailored to specific user needs, applications, and application domains. In particular, the SDK4ED platform enables the user to select specific types of analysis to be executed for a given application with proper configuration, as well as to declare which quality attributes are more important for a given application (among maintainability, dependability, and energy efficiency) in order to retrieve more tailored recommendations. Hence, the term highly configurable does not cover (or imply) the potential customizability and extensibility of the SDK4ED platform through a formal approach for the streamlined addition of third-party toolboxes, which could support the analysis of additional quality attributes or other programming languages. Although the later is highly interesting and a valuable feature, emphasis was given on the former, as it was expressed as one of the main desired requirements of the SDK4ED project during the Requirements Elicitation phase that was carried out at the beginning of the project with the industrial partners (i.e, use case providers) (Sas and Avgeriou 2020). The potential extensibility and customizability of the SDK4ED platform is an interesting direction for future work and a valuable feature that is planned to be included in future updates of the platform.

The SDK4ED platform is user-friendly, useful, and relevant to the embedded software industry, reducing development costs of embedded software.

All the use case providers, according to the evaluation reports that they delivered by the end of the project (SDK4ED 2019a,b,c,d), found the SDK4ED platform user-friendly and easy to use. More specifically, all of them stated that "the final platform is very intuitive and user friendly", whereas its "learning curve is reasonable and the tool can be adopted in a very easy way" (SDK4ED 2019d). They also found the mechanisms provided by the platform very relevant and useful and expressed their will to include the platform as part of their development process. In particular, the automotive use case provider has already incorporated the SDK4ED platform as part of their development and quality assurance process. They have hired a quality assurance (QA) manager who is in charge of TD, who regularly consults the SDK4ED tool in her daily routine, placing special emphasis on the suggestions provided by the tool. Her main responsibilities are: (i) monitor the TD, (ii) suggest code improvements, and (iii) reduce the number of code smells, and, in turn, the value of the overall TD of the applications under development. In the future, they are also planning to incorporate the dependability toolbox into their workflow. In the same sense, the healthcare and airborne

use case providers have expressed their plans to incorporate the energy estimators and the security checks (which are more important for their applications) that are provided by the Energy and Dependability Toolbox respectively in their quality assurances processes (SDK4ED 2019a,b).

Finally, the 15 participants of the industrial study that was presented in Section 6.2, evaluated positively the usability of the SDK4ED platform, assigning an average SUS score of 78.6%, which corresponds to an average rank of B. This indicates that they found the platform useful and relevant to their industrial needs. Also, specific features like the monetization of the assessments, the identification of hotspots, and the assistance in producing cleaner new code were marked as the most beneficial and relevant features of the platform, which provides further confidence for the industrial relevance of the solutions that are provided by the SDK4ED platform.

8 Conclusion

The development of embedded software applications is a difficult task mainly due to the limitations that are imposed by the hardware platforms on which they operate, as well as by the often-conflicting non-functional requirements (i.e., quality criteria) that they need to satisfy. The SDK4ED platform, which is the outcome of an EU-funded research project, aims at facilitating the development of high-quality embedded software, focusing mainly on the quality attributes of energy efficiency, dependability, and maintainability. The platform provides novel solutions for monitoring and optimizing these three quality attributes of embedded software individually. It also provides novel solutions for facilitating decision-making during the development of embedded software, through advanced forecasting techniques that predict the future evolution of the three quality attributes of interest, and a novel fuzzy multi-criteria decision-making technique for computing the impact of code refactorings on critical quality attributes, based on trade-off analysis among them.

In the present paper, we provided an overview of the SDK4ED platform, along with a description of the main novel features that it encapsulates. We also demonstrated the usefulness and practicality of the platform through three use cases based on real-world embedded software applications coming from the automotive, airborne, and healthcare domains. The results of an industrial study were also presented. According to the qualitative analysis, the SDK4ED platform is highly user-friendly and provides solutions that are useful and with industrial relevance. According to the use cases, the development teams were able to monitor effectively the quality attributes of choice and retrieve useful recommendations for quality improvement. All the involved industrial partners expressed their desire to include the SDK4ED platform (or at least parts of it) in their actual development and quality evaluation processes, whereas one of the industrial partners of the SDK4ED consortium, has already hired a dedicated quality assurance (QA) manager, who consults the SDK4ED platform in a frequent basis in order to monitor and optimize the quality of the embedded software applications that are produced by the company. Another interesting observation, which can be also considered a benefit, is that the high-configurability and adaptability of the platform enables it to be adapted to custom user needs and application domains. To the best of

our knowledge, this is the only quality evaluation platform that encapsulates novel monitoring and optimization mechanisms for multiple quality attributes, as well as that considers the interplay among different often-conflicting quality attributes of interest. In the future, we are planning to enhance and encourage the further extensibility of the SDK4ED platform with new third-party features and toolboxes, in order to support the evaluation of additional quality attributes. as well as the analysis of other programming languages. In particular, we foresee to provide a formal integration pipeline along with detailed guidelines and step-by-step tutorials on how a new third-party toolbox can be integrated to the SDK4ED platform.

9 Acknowledgments

This work is partially funded by the European Union's Horizon 2020 Research and Innovation Programme through SDK4ED project under Grant Agreement No. 780572.

Declarations

Authors' Contributions

Conceptualization: Miltiadis Siavvas, Dimitrios Tsoukalas, Charalampos Marantos, Dimitrios Soudris, Alexander Chatzigeorgiou, Apostolos Ampatzoglou, Erol Gelenbe, Dimitrios Tzovaras; Methodology: Miltiadis Siavvas, Dimitrios Tsoukalas, Alexandros Chatzigeorgiou, Dimitrios Soudris, Lazaros Papadopoulos; Formal analysis and investigation: Miltiadis Siavvas, Dimitrios Tsoukalas, Oliviu Matei, Christos Strydis, Paris Avgeriou, Muhammad Ali Siddiqi, Philippe Chrobocinski, Katarzyna Filus, Joanna Domańska; Writing - original draft preparation: Miltiadis Siavvas; Writing - review and editing: Miltiadis Siavvas, Dimitrios Tsoukalas, Alexander Chatzigeorgiou, Erol Gelenbe, Dimitrios Soudris, Dimitrios Tzovaras; Funding acquisition: Dionysios Kehagias; Resources: Dionysios Kehagias; Supervision: Miltiadis Siavvas, Alexander Chatzigeorgiou, Dimitrios Soudris, Erol Gelenbe, Dimitrios Tzovaras.

Funding

This work is partially funded by the European Union's Horizon 2020 Research and Innovation Programme through SDK4ED project under Grant Agreement No. 780572.

Competing Interests

The authors have no competing interests as defined by Springer, or other interests that might be perceived to influence the results and/or discussion reported in this paper.

References

Aggarwal K, Hindle A, Stroulia E (2015) Greenadvisor: A tool for analyzing the impact of software evolution on energy consumption. In: 2015 IEEE international conference on software maintenance and evolution (ICSME), IEEE, pp 311–320

Alshammari B, Fidge C, Corney D (2011) A Hierarchical Security Assessment Model for Object-Oriented Programs. 2011 11th International Conference on Quality Software (1):218–227, DOI 10.1109/QSIC.2011.31

Amanatidis T, Mittas N, Moschou A, Chatzigeorgiou A, Ampatzoglou A, Angelis L (2020) Evaluating the agreement among technical debt measurement tools: building an empirical benchmark of technical debt liabilities. Empirical Software Engineering 25:4161–4204

Ampatzoglou A, Ampatzoglou A, Chatzigeorgiou A, Avgeriou P (2015) The financial aspect of managing technical debt: A systematic literature review. Information and Software Technology 64:52–73, DOI https://doi.org/10.1016/j.infsof.2015.04.001, URL https://www.sciencedirect.com/science/article/pii/S0950584915000762

Ampatzoglou A, Michailidis A, Sarikyriakidis C, Ampatzoglou A, Chatzigeorgiou A, Avgeriou P (2018) A framework for managing interest in technical debt: An industrial validation. In: 2018 IEEE/ACM International Conference on Technical Debt (TechDebt), pp 115–124

Ampatzoglou A, Tsintzira AA, Arvanitou EM, Chatzigeorgiou A, Stamelos I, Moga A, Heb R, Matei O, Tsiridis N, Kehagias D (2019) Applying the single responsibility principle in industry: Modularity benefits and trade-offs. In: Proceedings of the 23rd International Conference on Evaluation and Assessment in Software Engineering, Association for Computing Machinery, New York, NY, USA, EASE '19, p 347–352, DOI 10.1145/3319008.3320125, URL https://doi.org/10.1145/3319008.3320125

Ampatzoglou A, Chatzigeorgiou A, Arvanitou EM, Bibi S (2022) Sdk4ed: A platform for technical debt management. Software: Practice and Experience

Ansar SA, Alka, Khan RA (2018) A phase-wise review of software security metrics. In: Networking Communication and Data Knowledge Engineering

Ardalani N, Lestourgeon C, Sankaralingam K, Zhu X (2015) Cross-architecture performance prediction (xapp) using cpu code to predict gpu performance. In: Proceedings of the 48th International Symposium on Microarchitecture, ACM, pp 725–737

Arora R (2017) ITALC : Interactive Tool for Application - Level Checkpointing. Proceedings of the Fourth International Workshop on HPC User Support Tools

Avgeriou PC, Taibi D, Ampatzoglou A, Arcelli Fontana F, Besker T, Chatzigeorgiou A, Lenarduzzi V, Martini A, Moschou A, Pigazzini I, Saarimaki N, Sas DD, de Toledo SS, Tsintzira AA (2021) An overview and comparison of technical debt measurement tools. IEEE Software 38(3):61–71, DOI 10.1109/MS.2020.3024958

Awan MA, Petters SM (2011) Enhanced race-to-halt: A leakage-aware energy management approach for dynamic priority systems. In: 2011 23rd Euromicro Conference on Real-Time Systems, IEEE, pp 92–101

Bazzaz M, Salehi M, Ejlali A (2013) An accurate instruction-level energy estimation model and tool for embedded systems. IEEE transactions on instrumentation and measurement 62(7):1927–1934

Besker T, Martini A, Bosch J (2019) Software developer productivity loss due to technical debt—a replication and extension study examining developers' development work. Journal of Systems and Software 156:41–61, DOI https://doi.org/10.1016/j.jss.2019.06.004, URL https://www.sciencedirect.com/science/article/pii/S0164121219301335

Binkert N, Beckmann B, Black G, Reinhardt SK, Saidi A, Basu A, Hestness J, Hower DR, Krishna T, Sardashti S, et al. (2011) The gem5 simulator. ACM SIGARCH computer architecture news 39(2):1–7

Brooke J, et al. (1996) Sus-a quick and dirty usability scale. Usability evaluation in industry 189(194):4–7

Brown N, Cai Y, Guo Y, Kazman R, Kim M, Kruchten P, Lim E, MacCormack A, Nord R, Ozkaya I, Sangwan R, Seaman C, Sullivan K, Zazworka N (2010) Managing technical debt in software-reliant systems. In: Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research, Association for Computing Machinery, New York, NY, USA, FoSER '10, p 47–52, DOI 10.1145/1882362.1882373, URL https://doi.org/10.1145/1882362.1882373

Catthoor F, Danckaert K, Brockmeyer E, Kulkarni K, Kjeldsberg PG, Van Achteren T, Omnes T (2002) Data access and storage management for embedded programmable processors.

Springer Science & Business Media
Charalampidou S, Ampatzoglou A, Chatzigeorgiou A, Gkortzis A, Avgeriou P (2017) Identifying extract method refactoring opportunities based on functional relevance. IEEE Transactions on Software Engineering 43(10):954–974, DOI 10.1109/TSE.2016.2645572
Chatzigeorgiou A, Ampatzoglou A, Ampatzoglou A, Amanatidis T (2015) Estimating the breaking point for technical debt. In: 2015 IEEE 7th International Workshop on Managing Technical Debt (MTD), IEEE Computer Society, Los Alamitos, CA, USA, pp 53–56, DOI 10.1109/MTD.2015.7332625, URL https://doi.ieeecomputersociety.org/10.1109/MTD.2015.7332625
Chowdhury I, Zulkernine M (2011) Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. Journal of Systems Architecture DOI 10.1016/j.sysarc.2010.06.003, URL http://www.mendeley.com/research/using-complexity-coupling-cohesion-metrics-early-indicators-vulnerabilities
Colombo RT, Pessôa MS, Guerra AC, Filho AB, Gomes CC (2012) Prioritization of software security intangible attributes. ACM SIGSOFT Software Engineering Notes 37(6):1, DOI 10.1145/2382756.2382781, URL http://dl.acm.org/citation.cfm?doid=2382756.2382781
Cunningham W (1993) The wycash portfolio management system. ACM SIGPLAN OOPS Messenger 4(2):29–30
Dam HK, Tran T, Pham T, Ng SW, Grundy J, Ghose A (2018) Automatic feature learning for predicting vulnerable software components. IEEE Transactions on Software Engineering 47(1):67–85
David H, Gorbatov E, Hanebutte UR, Khanna R, Le C (2010) Rapl: Memory power estimation and capping. In: Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design, pp 189–194
Dayanandan U, Kalimuthu V (2018) Software architectural quality assessment model for security analysis using fuzzy analytical hierarchy process (fahp) method. 3D Research 9(3):31, DOI 10.1007/s13319-018-0183-x, URL https://doi.org/10.1007/s13319-018-0183-x
DeMarco T (1986) Controlling Software Projects: Management, Measurement, and Estimates. Prentice Hall PTR, Upper Saddle River, NJ, USA
Digkas G, Chatzigeorgiou A, Ampatzoglou A, Avgeriou P (2022) Can clean new code reduce technical debt density? IEEE Transactions on Software Engineering 48(05):1705–1721, DOI 10.1109/TSE.2020.3032557
Eder K, Gallagher JP, Fagas G, Gammaitoni L, Paul D (2017) Energy-aware software engineering. ICT-energy concepts for energy efficiency and sustainability pp 103–127
Egwutuoha IP, Levy D, Selic B, Chen S (2013) A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems. Journal of Supercomputing 65(3):1302–1326, DOI 10.1007/s11227-013-0884-0
Elnozahy EN, Alvisi L, Wang YM, Johnson DB (2002) A Survey of Rollback-recovery Protocols in Message-passing Systems. ACM Comput Surveys 34(3):375–408, DOI 10.1145/568522.568525, URL http://doi.acm.org/10.1145/568522.568525
Elo S, Kyngäs H (2008) The qualitative content analysis process. Journal of advanced nursing 62(1):107–115
Euler L (1783) De serie lambertina plurimisque eius insignibus proprietatibus. Acta Academiae scientiarum imperialis petropolitanae pp 29–51
Filus K, Boryszko P, Domańska J, Siavvas M, Gelenbe E (2021a) Efficient feature selection for static analysis vulnerability prediction. Sensors 21(4):1133
Filus K, Siavvas M, Domańska J, Gelenbe E (2021b) The random neural network as a bonding model for software vulnerability prediction. In: Modelling, Analysis, and Simulation of Computer and Telecommunication Systems: 28th International Symposium, MASCOTS 2020, Nice, France, November 17–19, 2020, Revised Selected Papers 28, Springer, pp 102–116
Fontana FA, Roveda R, Zanoni M (2016) Technical debt indexes provided by tools: A preliminary discussion. In: 2016 IEEE 8th International Workshop on Managing Technical Debt (MTD), pp 28–31, DOI 10.1109/MTD.2016.11
Fowers J, Brown G, Cooke P, Stitt G (2012) A performance and energy comparison of fpgas, gpus, and multicores for sliding-window applications. In: Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays, pp 47–56
Fowler M (1999) Refactoring: Improving the Design of Existing Code. Addison-Wesley Longman Publishing Co., Inc., USA

Gelenbe E (1989) Random neural networks with negative and positive signals and product form solution. Neural computation 1(4):502–510

Gelenbe E, Siavvas M (2021) Minimizing energy and computation in long-running software. Applied Sciences 11(3):1169

Gelenbe E, Boryszko P, Siavvas M, Domanska J (2020) Optimum checkpoints for time and energy. In: 2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), IEEE, pp 1–8

Georgiou S, Rizou S, Spinellis D (2019) Software development lifecycle for energy efficiency: techniques and tools. ACM Computing Surveys (CSUR) 52(4):1–33

Guo S, Zhao H (2017) Fuzzy best-worst multi-criteria decision-making method and its applications. Knowledge-Based Systems 121:23–31

Hanif H, Maffeis S (2022) Vulberta: Simplified source code pre-training for vulnerability detection. In: 2022 International Joint Conference on Neural Networks (IJCNN), IEEE, pp 1–8

Holzmann GJ (2017) The Value of Doubt. IEEE Software 34(1):106–109, DOI 10.1109/MS.2017.19

Hönig T, Eibel C, Kapitza R, Schröder-Preikschat W (2012) Seep: exploiting symbolic execution for energy-aware programming. ACM SIGOPS Operating Systems Review 45(3):58–62

Hönig T, Janker H, Eibel C, Mihelic O, Kapitza R (2014) Proactive energy-aware programming with {PEEK}. In: 2014 Conference on Timely Results in Operating Systems ({TRIOS} 14)

Hursey J, Squyres JM, Mattox TI, Lumsdaine A (2007) The Design and Implementation of Checkpoint / Restart Process Fault Tolerance for Open MPI. Architecture

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

ISO/IEC (2013) ISO/IEC 27001:2013(en) Information technology — Security techniques — Information security management systems — Requirements. Tech. rep.

Kim S, Choi J, Ahmed ME, Nepal S, Kim H (2022) Vuldebert: A vulnerability detection system using bert. In: 2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp 69–74, DOI 10.1109/ISSREW55968.2022.00042

Lai ST (2010) An analyzer-based software security measurement model for enhancing software system security. Proceedings - 2010 2nd WRI World Congress on Software Engineering DOI 10.1109/WCSE.2010.104

Lambert JH (1758) Observationes variae in mathesin puram. Acta Helvetica 3(1):128–168

Lamprakos CP, Marantos C, Siavvas M, Papadopoulos L, Tsintzira AA, Ampatzoglou A, Chatzigeorgiou A, Kehagias D, Soudris D (2022) Translating quality-driven code change selection to an instance of multiple-criteria decision making. Information and Software Technology 145:106851

Lee S, Meredith JS, Vetter JS (2015) Compass: A framework for automated performance modeling and prediction. In: Proceedings of the 29th ACM on International Conference on Supercomputing, pp 405–414

Li Z, Avgeriou P, Liang P (2015) A systematic mapping study on technical debt and its management. Journal of Systems and Software 101:193–220, DOI https://doi.org/10.1016/j.jss.2014.12.027, URL https://www.sciencedirect.com/science/article/pii/S0164121214002854

Li Z, Zou D, Xu S, Ou X, Jin H, Wang S, Deng Z, Zhong Y (2018) Vuldeepecker: A deep learning-based system for vulnerability detection. arXiv preprint arXiv:180101681

Llamocca D, Carranza C, Pattichis M (2011) Separable fir filtering in fpga and gpu implementations: Energy, performance, and accuracy considerations. In: 2011 21st International Conference on Field Programmable Logic and Applications, IEEE, pp 363–368

Losada N, Martín MJ, Rodríguez G, Gonzalez P (2016) Portable application-level checkpointing for hybrid MPI-OpenMP applications. Procedia Computer Science 80:19–29, DOI 10.1016/j.procs.2016.05.294

Lowe-Power J, Ahmad AM, Akram A, Alian M, Amslinger R, Andreozzi M, Armejach A, Asmussen N, Beckmann B, Bharadwaj S, et al. (2020) The gem5 simulator: Version 20.0+. arXiv preprint arXiv:200703152

Manotas I, Pollock L, Clause J (2014) Seeds: A software engineer's energy-optimization decision support framework. In: Proceedings of the 36th International Conference on Software Engineering, pp 503–514

Manotas I, Bird C, Zhang R, Shepherd D, Jaspan C, Sadowski C, Pollock L, Clause J (2016) An empirical study of practitioners' perspectives on green software engineering. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), IEEE, pp 237–248

Marantos C, Salapas K, Papadopoulos L, Soudris D (2021) A flexible tool for estimating applications performance and energy consumption through static analysis. SN Computer Science 2(1):1–11

Marantos C, Papadopoulos L, Lamprakos CP, Salapas K, Soudris D (2022a) Bringing energy efficiency closer to application developers: An extensible software analysis framework. IEEE Transactions on Sustainable Computing

Marantos C, Papadopoulos L, Tsintzira AA, Ampatzoglou A, Chatzigeorgiou A, Soudris D (2022b) Decision support for gpu acceleration by predicting energy savings and programming effort. Sustainable Computing: Informatics and Systems 34:100631

Marantos C, Siavvas M, Tsoukalas D, Lamprakos CP, Papadopoulos L, Boryszko P, Filus K, Domańska J, Ampatzoglou A, Chatzigeorgiou A, et al. (2022c) Sdk4ed: One-click platform for energy-aware, maintainable and dependable applications. In: 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, pp 981–986

Medeiros N, Ivaki N, Costa P, Vieira M (2018) An approach for trustworthiness benchmarking using software metrics. In: 2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC), pp 84–93

Mohammed NM, Niazi M, Alshayeb M, Mahmood S (2016) Exploring Software Security Approaches in Software Development Lifecycle: A Systematic Mapping Study. Comp Stand & Interf DOI 10.1016/j.csi.2016.10.001, URL http://linkinghub.elsevier.com/retrieve/pii/S0920548916301155

Moody A, Bronevetsky G, Mohror K, d Supinski BR (2010) Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System. In: 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, pp 1–11, DOI 10.1109/SC.2010.18

Morrison P, Moye D, Pandita R, Williams L (2018) Mapping the field of software life cycle security metrics. Information and Software Technology 102(May):146–159, DOI 10.1016/j.infsof.2018.05.011, URL https://doi.org/10.1016/j.infsof.2018.05.011

Noureddine A, Rouvoy R, Seinturier L (2015) Monitoring energy hotspots in software. Automated Software Engineering 22(3):291–332, DOI 10.1007/s10515-014-0171-1, URL http://dx.doi.org/10.1007/s10515-014-0171-1

Pinto G, Castor F (2017) Energy efficiency: a new concern for application software developers. Communications of the ACM 60(12):68–75

Reddy R, Petrov P (2010) Cache partitioning for energy-efficient and interference-free embedded multitasking. ACM Transactions on Embedded Computing Systems (TECS) 9(3):1–35

Rios N, Oliveira Spínola R, Mendonça M, Seaman C (2019) Supporting analysis of technical debt causes and effects with cross-company probabilistic cause-effect diagrams. In: 2019 IEEE/ACM International Conference on Technical Debt (TechDebt), pp 3–12, DOI 10.1109/TechDebt.2019.00009

Rodríguez G, Martín MJ, González P, Touriño J, Doallo R (2010) CPPC: a compiler-assisted tool for portable checkpointing of message-passing applications. Concurrency and Computation: Practice and Experience 22(6):749–766, DOI 10.1002/cpe.1541, URL http://dx.doi.org/10.1002/cpe.1541

Saaty TL (2008) Decision making with the analytic hierarchy process. International Journal of Services Sciences

Sas D, Avgeriou P (2020) Quality attribute trade-offs in the embedded systems industry: an exploratory case study. Software Quality Journal 28(2):505–534

Scandariato R, Walden J, Hovsepyan A, Joosen W (2014) Predicting vulnerable software components via text mining. IEEE Transactions on Software Engineering 40(10):993–1006, DOI 10.1109/TSE.2014.2340398

SDK4ED (2019a) D7.2 - Airborne Use Case Deployment. Tech. rep.

SDK4ED (2019b) D7.3 - Healthcare Use Case Deployment. Tech. rep.

SDK4ED (2019c) D7.4 - Automotive Use Case Deployment. Tech. rep.

SDK4ED (2019d) D7.5 - Empirical Study Results. Tech. rep.

Seaman C, Guo Y (2011) Chapter 2 - measuring and monitoring technical debt. Advances in Computers, vol 82, Elsevier, pp 25–46, DOI https://doi.org/10.1016/B978-0-12-385512-1.00002-5, URL

https://www.sciencedirect.com/science/article/pii/B9780123855121000025
Seaman CB (1999) Qualitative methods in empirical studies of software engineering. IEEE Transactions on software engineering 25(4):557–572
Sentilles S, Papatheocharous E, Ciccozzi F (2018) What do we know about software security evaluation? A preliminary study. In: 6th International Workshop on Quantitative Approaches to Software Quality
Shahzad F, Thies J, Wellein G (2018) CRAFT: A library for easier application-level Checkpoint/Restart and Automatic Fault Tolerance. IEEE Transactions on Parallel and Distributed Systems
Shin Y, Meneely A, Williams L, Osborne JA (2011) Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities. IEEE Transactions on Software Engineering 37(6):772–787, DOI 10.1109/TSE.2010.81
Siavvas M, Gelenbe E (2019a) Optimum checkpoints for programs with loops. Simulation Modelling Practice and Theory 97:101951
Siavvas M, Gelenbe E (2019b) Optimum interval for application-level checkpoints. In: 2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), IEEE, pp 145–150
Siavvas M, Kehagias D, Tzovaras D, Gelenbe E (2021) A hierarchical model for quantifying software security based on static analysis alerts and software metrics. Software Quality Journal 29(2):431–507
Siddiqi MA, Tsintzira AA, Digkas G, Siavvas MG, Strydis C (2021) Adding security to implantable medical devices: Can we afford it? In: EWSN, pp 67–78
Sommerville I (1995) Software engineering. Addison-Wesley
Suryanarayana G, Samarthyam G, Sharma T (2014) Refactoring for software design smells: managing technical debt. Morgan Kaufmann
Takizawa H, Koyama K, Sato K, Komatsu K, Kobayashi H (2011) CheCL: Transparent checkpointing and process migration of OpenCL applications. Proceedings - 25th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2011 pp 864–876, DOI 10.1109/IPDPS.2011.85
Tsantalis N, Chaikalis T, Chatzigeorgiou A (2018) Ten years of jdeodorant: Lessons learned from the hunt for smells. In: 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 4–14, DOI 10.1109/SANER.2018.8330192
Tsintzira AA, Ampatzoglou A, Matei O, Ampatzoglou A, Chatzigeorgiou A, Heb R (2019) Technical debt quantification through metrics: an industrial validation. In: 15th China-Europe International Symposium on software engineering education
Tsoukalas D, Jankovic M, Siavvas M, Kehagias D, Chatzigeorgiou A, Tzovaras D (2019) On the applicability of time series models for technical debt forecasting. In: 15th China-Europe International Symposium on Software Engineering Education (CEISEE) (in press), pp 1–10, DOI 10.13140/RG.2.2.33152.79367
Tsoukalas D, Kehagias D, Siavvas M, Chatzigeorgiou A (2020) Technical Debt Forecasting: An empirical study on open-source repositories. In: Journal of Systems and Software, vol 170, p 110777, DOI https://doi.org/10.1016/j.jss.2020.110777, URL http://www.sciencedirect.com/science/article/pii/S0164121220301904
Tsoukalas D, Siavvas M, Kehagias D, Ampatzoglou A, Chatzigeorgiou A (2023) A practical approach for technical debt prioritization based on class-level forecasting. Journal of Software: Evolution and Process p e2564
Wagner S, Goeb A, Heinemann L, Kläs M, Lampasona C, Lochmann K, Mayr A, Plösch R, Seidl A, Streit J, Trendowicz A (2015) Operationalised product quality models and assessment: The Quamoco approach. Information and Software Technology 62:101–123, DOI 10.1016/j.infsof.2015.02.009, URL http://www.sciencedirect.com/science/article/pii/S0950584915000452
Walden J, Doyle M, Welch GA, Whelan M (2009) Security of open source web applications. 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM 2009 DOI 10.1109/ESEM.2009.5314215
Wang S, Zhong G, Mitra T (2017) Cgpredict: Embedded gpu performance estimation from single-threaded applications. ACM Transactions on Embedded Computing Systems (TECS) 16(5s):146
Wang W, Mishra P, Ranka S (2011) Dynamic cache reconfiguration and partitioning for energy optimization in real-time multi-core systems. In: 2011 48th ACM/EDAC/IEEE Design

Automation Conference (DAC), IEEE, pp 948–953
Xu H, Heijmans J, Visser J (2013) A practical model for rating software security. Proceedings - 7th International Conference on Software Security and Reliability Companion, SERE-C 2013 DOI 10.1109/SERE-C.2013.11
Zafar S, Mehboob M, Naveed A, Malik B (2015) Security quality model: an extension of Dromey's model. Software Quality Journal 23(1), DOI 10.1007/s11219-013-9223-1
Zagane M, Abdi MK, Alenezi M (2020) Deep learning for software vulnerabilities detection using code metrics. IEEE Access 8
Zheng X, John LK, Gerstlauer A (2016) Accurate phase-level cross-platform power and performance estimation. In: Proceedings of the 53rd Annual Design Automation Conference
Zheng X, Vikalo H, Song S, John LK, Gerstlauer A (2017) Sampling-based binary-level cross-platform performance estimation. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, IEEE, pp 1709–1714
Zhou Y, Liu S, Siow J, Du X, Liu Y (2019) Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. arXiv preprint arXiv:190903496