SDK4ED: One-click platform for Energy-aware, Maintainable and Dependable Applications

Charalampos Marantos*, Miltiadis Siavvas†, Dimitrios Tsoukalas†, Christos P. Lamprakos*,
Lazaros Papadopoulos*, Paweł Boryszko§, Katarzyna Filus§, Joanna Domańska§,
Apostolos Ampatzoglou‡, Alexander Chatzigeorgiou‡, Erol Gelenbe§, Dionysios Kehagias†, Dimitrios Soudris*
* School of Electrical and Computer Engineering, National Technical University of Athens, Greece
† Centre for Research and Technology Hellas, Thessaloniki, Greece
‡ Department of Applied Informatics, University of Macedonia, Greece
§ Institute of Theoretical & Applied Computer Science, IITIS-PAN, Gliwice, Poland

***Abstract*—Developing modern secure and low-energy applications in a short time imposes new challenges and creates the need of designing new software tools to assist developers in all phases of application development. The design of such tools cannot be considered a trivial task, as they should be able to provide optimization of multiple quality requirements. In this paper, we introduce the SDK4ED platform, which incorporates advanced methods and tools for measuring and optimizing maintainability, dependability and energy. The presented solution offers a complete tool-flow for providing indicators and optimization methods with emphasis on embedded software. Effective forecasting models and decision-making solutions are also implemented to improve the quality of the software, respecting the constraints imposed on maintenance standards, energy consumption limits and security vulnerabilities. The use of the SDK4ED platform is demonstrated in a healthcare embedded application.**

***Index Terms*—Software quality, Energy consumption, Dependability, Development Toolkit**

I. Introduction

Computing devices are now everywhere, installed in many environments such as industrial, sanitary, and residential buildings. As these applications continue to evolve, developers face the challenges of meeting very different requirements such as energy saving, software maintainability, and security standards. Designing high quality software for such systems is not an easy task, due to the high complexity of modern applications. Demanding calculations and data processing require a large amount of energy. To make things worse, the maintenance costs caused by poor design quality (often result of a rapid development process) increase the complexity even more. Finally, security is an aspect of major concern, as an individual vulnerability, caused by a common programming error, can have severe consequences.

Several approaches have been proposed over the years to quantify and optimize individual software quality aspects [1] [2] [3]. However, existing work varies significantly, depending on the quality aspect and the level of abstraction at which it is addressed. On the one hand, there are tools that suggest best practices for software maintainability based on empirical studies and, on the other hand, practitioners suggest

This work was funded by the EU's Horizon 2020 Research and Innovation Programme through SDK4ED project under Grant Agreement No. 780572.

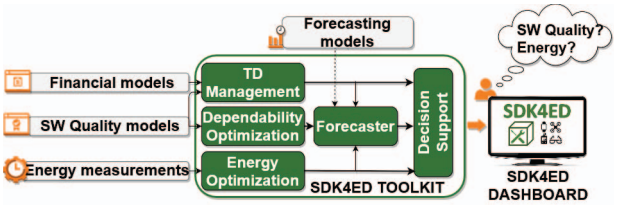


Fig. 1: Overview of the SDK4ED framework

reducing energy by making source code transformations for using specialized DSPs, accelerators, memory management optimizations, etc. These optimizations are usually customized and closely related to the targeted applications. Another important thing is that optimizing one quality aspect can have a positive or a negative impact on another, for example energy optimizations can make code less maintainable. None of the existing solutions has combined different attributes such as energy and maintainability in a unified way offering a trade-off analysis and a decision-support between conflicting criteria.

This work aims to give a solution to the aforementioned challenges by designing and combining cutting-edge technologies in one place. During three years of research, subfeatures and new tools-components, related to the optimization of individual quality characteristics were designed and presented in a large number of research articles (https://sdk4ed.eu/documents). This paper aims to present the unified and fully functional platform, which is the final product of the SDK4ED EU Project, offering a flexible, easy-to-use software analysis toolkit to assist designers in developing Energy-aware, Maintainable and Dependable embedded applications.

The rest of the manuscript is structured as follows. Section II provides the overview of the SDK4ED Platform, along with detailed description of its individual components. In Section III the SDK4ED Platform is demonstrated in one pilot use-case and finally, conclusions are summarized in Section IV.

II. SDK4ED Integrated Platform

A. Overview

Figure 1 presents a high-level overview of the SDK4ED Platform. The SDK4ED platform consists of five basic units,

three autonomous and two that operate based on the first three:

- Energy Optimization: Estimates and minimizes the Energy of a given application on specified target devices.
- Technical Debt Management: Monitors and optimizes the Maintainability through the notion of Technical Debt.
- Dependability Optimization: Monitors and optimizes the Security and Reliability of applications.
- Forecaster: Predicts the future evolution of the three aforementioned quality attributes of interest.
- Decision Support: Combines the input from the other modules and performs a trade-off analysis to determine impacts of refactorings and provide recommendations.

More details are provided in the the rest of the Section. Each of the main toolboxes consists of two parts. The first provides indicators to help developers identify the characteristics that have the greatest impact on each quality attribute. In addition, this part identifies code blocks (classes, functions, statements) that are candidate places for relevant optimizations. The second part is responsible for providing refactorings suggestions in order to improve the targeted quality attribute.

B. Energy Toolbox

1) Energy Monitoring: Energy indicators are monitored using dynamic instrumentation tools (Valgrind and Perf). The selected indicators include *CPU cycles, Branch misses, Number of memory accesses, Ratio of D-Cache/I-Cache misses* and *Data races*. To identify the energy hotspots, the toolbox analyses the CLANG Abstract Syntax tree (AST) and the profilling results from the previous step to find the code blocks in which the number of CPU cycles exceeds a limit ($> 1\%$).

To reduce the overhead of dynamic instrumentation, a static analysis mechanism is also integrated. This mechanism estimates the time and energy required for executing each application code block on different targeted devices. Regarding the loop statements, a dynamic information (the number of iterations) is gathered as an input from the user to provide results for the entire application. As the SDK4ED Energy Toolbox aims to provide a cross-platform solution, the proposed approach uses general features at assembly level that model and characterize the application's performance and energy consumption: Estimated *throughput* (by LLVM-mca), number of *instructions*, number of *loads*, *stores*, *operation* instructions in class 1 (*add, sub, shift and mul*) and class 2 (*conv, arrays, div*) and *operation-load-store* instructions *order*. The dataset includes synthetic loops that perform random matrix operations. A k-means clustering pre-processing is performed to avoid using similar data-points that may cause overfitting and an Orthogonal Matching Pursuit regressor was employed, after comparing the accuracy of alternative models. Results on Rodinia benchmark applications executed on ARM Cortex A57 are considered acceptable (R2 score = 0.92) [4].

2) Energy Optimizations: The SDK4ED Energy Toolbox suggests 3 categories of optimizations.

- Data flow-related optimizations: This type of optimizations aim to improve the memory hierarchy utilization and reduce memory accesses. Typical examples are the loop transformations that are proposed in the case that the hot-spot under analysis includes nested loops that have a ratio of cache misses that is beyond a threshold.
- Concurrency-related optimizations: As modern embedded systems typically integrate multiple cores, developers are facing challenges imposed by concurrency. A typical example is the deadlock bug which causes application stalls. The solution in this case is the proper use of locking mechanisms to avoid deadlocks, when a large number of Data races are monitored for a hotspot.
- Energy Gains by Acceleration: A massive improvement can be achieved by using the accelerators of modern heterogeneous embedded architectures. The role of the Energy Toolbox is to predict the potential energy gains by acceleration assisting developers to decide if it is worth developing GPU code for a hotspot. This component is based on a dynamic instrumentation approach. Classification models use features already presented in the literature [5] for estimating speedup of using General Purpose GPUs (GPGPUs) after investigating which of them are also related to energy gain (correlation analysis). After comparing different models we employ an Ensemble Voting Method reaching a final level of 85% classification accuracy. For building the dataset we combined an equal number of synthetic benchmarks and real-world applications (i.e. taken from benchmark suites).

The Energy Toolbox is a docker API service (Python Flask), the estimation models were built in Scikit Learn and a MongoDB database is used for saving the results.

C. Maintainability Toolbox

1) Maintainability Monitoring: The SDK4ED maintainability toolbox leverages the power of the successful *Technical Debt* (TD) metaphor to capture and represent inefficiencies in the analyzed software. Technical Debt refers to a universal software development phenomenon where design constructs are expedient in the short term but set up a technical context that can make future changes more costly or impossible [1]. In particular, the SDK4ED TD toolbox calculates the key concepts of principal, interest and interest probability [6].

For assessing the effort required to resolve the identified inefficiencies (principal) the toolbox relies on the most widely used tool , namely SonarQube. SonarQube identifies five types of problems, namely bugs, vulnerabilities, code smells, (lack of) test coverage, and duplications. Furthermore, the SDK4ED toolbox identifies Long Methods, violations of the modularity principle in the design of packages and architectural smells (such as cyclic and unstable dependencies). The TD toolbox incorporates a method for assessing TD interest which reflects the additional maintenance cost in future maintenance because of the presence of TD in the current software version. Particular emphasis is placed on the analysis of *New Code* that is to be committed to the codebase, since ensuring the high quality of new code can gradually lead to software quality improvement.

As TD management requires the continuous monitoring of the underlying issues over time, the SDK4ED toolbox provides

various views for studying its evolution. Other views, such as the TD Analysis panel offer insight into key indicators such as the TD in minutes and currency, the ranking of the analyzed project against other projects in the database of analyzed systems, the number of issues and the novel indicator of Breaking Point. Breaking Point equals the number of software versions ahead in which the accumulated additional maintenance costs (interest) are expected to exceed any savings gained by not repaying TD (principal) [7].

2) Maintainability Optimizations: Each identified TD issue represents an opportunity for improving the internal quality of software thereby repaying TD. The user is presented with lists of refactoring opportunities . Among them, the toolbox yields major maintainability optimizations consisting in "Extract Long Method" and "Move Class" refactoring suggestions. Since the number of opportunities may become large and intractable even for medimum-sized systems, refactoring suggestions are ranked according to their urgency. For each recommended code-level optimization the user is referred to examples of the underlying problem and ways to address it. For design-level refactoring suggestions, appropriate visualizations are employed, such as a heatmap illustrating methods in need of Extract Method refactoring and tree structures illustrating the proposed package organization.

D. Dependability Toolbox

1) Dependability Monitoring: With respect to Software Security, the Dependability Toolbox provides a novel hierarchical Security Assessment Model (SAM) [8] for measuring the internal security level of software products. The proposed model employs static analysis in order to detect security weaknesses (i.e., potential vulnerabilities) that reside in the source code of the analyzed software, and aggregates them through several levels of normalization and aggregation in order to produce a high-level score, i.e., the *Security Index*, which reflects the security level of the analyzed software. The proposed model is based on popular international standards, including the ISO/IEC 25010 and the ISO/IEC 27001.

Vulnerability Prediction Models (VPMs) are also provided with the purpose to identify security hotspots, i.e., software components (e.g., classes) that are likely to contain vulnerabilities. The toolbox provides VPMs that are based on text mining, software metrics, and deep learning, which have demonstrated sufficient predictive performance. The output of the VPMs is a *vulnerability flag* (a binary value), which indicates whether the analyzed component is potentially vulnerable or not, and a *vulnerability score* (a continuous value), which indicates how likely it is for the given component to contain a vulnerability.

With respect to Software Reliability, the Dependability Toolbox focuses on the popular Checkpoint and Restart (CR) mechanism. However, since the CR mechanism and particularly the selection of the checkpoint interval (i.e., frequency of checkpointing) is known to affect important runtime attributes, the toolbox provides mathematical models to estimate the performance and energy consumption of a given program with and without the adoption of the CR mechanism. Based on these mathematical models, both numerical and analytical [2] solutions are provided for determining the Optimum Checkpoint Interval, i.e., the interval that strikes a good balance among reliability, performance, and energy consumption.

2) Dependability Optimizations:

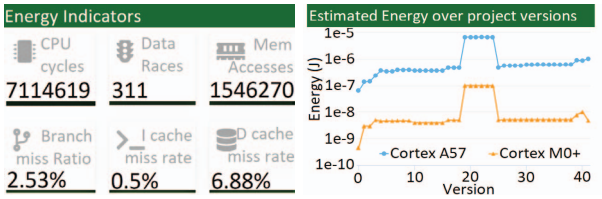
- Correction of Security Weaknesses: The Security Assessment Model, apart from the high-level security indicators, it also provides a detailed report of the security weaknesses (i.e., static analysis alerts) that it detected. For each static analysis alert, important information is provided by the toolbox, including its type, the exact line of code on which it resides, as well as external links with additional information and examples on how to fix them.
- Idendification of Security Hotspots: The VPMs of the Dependability Toolbox identify security hotspots, which are demonstrated to the user in the form of a table and a heatmap, allowing the developers to pinpoint those components that are critical from a security viewpoint. Hence, fortification activities can be better prioritized, by allocating limited test resources to high risk areas.
- Optimum Checkpoint Interval Recommendation: The computational- and energy-hungry hotspots (in fact, loops) that are detected by the Energy Toolbox, are analyzed with the mathematical models [2], and their optimum checkpoint interval is computed and recommended to the user. The user can then decide which hotspots to checkpoint and what interval to set, in order to achieve the highest possible energy and time savings.

E. Forecasting Toolbox

The Forecaster Toolbox focuses on predicting the future evolution of the three quality attributes of interest, namely Maintainability, Energy Consumption, and Dependability of software products. For this purpose, it integrates within its business logic various forecasting models and, depending on the occasion, 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. This is achieved through the provision of time series and machine-learning (ML) models, built based on the results (i.e., monitored indicators) of the three core SDK4ED modules (see Fig. 1).

To limit the initial set of indicators to those that have significant effects on the targeted quality attributes, the Forecaster Toolbox employs feature selection techniques, resulting in the development of less complex, but equally accurate forecasting models. Examples of selected indicators (i.e., predictors) include *code smells* for Maintainability, *CPU cycles* for Energy, and *exception handling* for Dependability, among others. These indicators constitute the external data that the Forecasting Toolbox expects to receive as input from the three core SDK4ED Toolboxes for the execution of the models.

Depending on the forecasting horizon and the targeted quality attribute, the Forecaster Toolbox employs either time series or popular ML models. In particular, regarding the time series approach, the ARIMA model was found to provide accurate results for short-term forecasts up to 8 commits



(a) Indicators panel (b) History panel

Fig. 2: Energy toolbox results

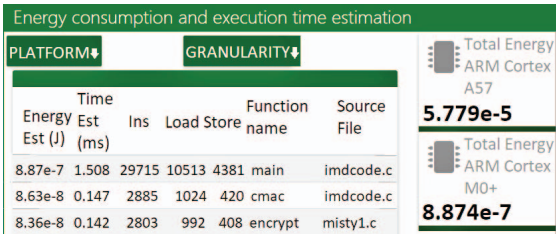


Fig. 3: Energy toolbox static analysis panel

ahead. Regarding mid-and long-term forecasts, various ML models such as the Linear Regression, Support Vector Regression, and Random Forest were investigated and compared [9]. The results indicated that non-linear models, such as the Random Forest, are capable to capture the future evolution trend of the three investigated quality attributes, providing reliable forecasts up to 40 commits ahead.

F. Decision-Support Toolbox

The main goal of the Decision Support Toolbox is to support decision making by enabling the development team to rank, sort and choose from alternative suggestions to improve TD, energy and dependability of the target software systems.

None of the individual toolboxes does take into consideration code qualities other than the one it optimizes. To take meaningful decisions, however, the user will need to evaluate all refactorings universally. Regardless of the source toolbox of a suggestion, the displayed information should include its impact on all aspects.From this point on, this information will be referred to as 'design space'. Production of the design space is based on approximate methods. Each toolbox deals with extremely dissimilar aspects making it hard to provide a fine-grained estimation of a suggestion's impact without actually applying it and repeating the analysis.Also, the approach should be as project- and platform-agnostic as possible.

In this component, after gathering all proposed refactorings from all the toolboxes, an empirically-derived, static model is queried for each of the suggestions. All matching entries are returned, forming the final design space of the problem which is solved using a state-of-the-art Multiple Criteria Decision Making (MCDM) algorithm [10].

G. Implementation

Toolboxes provide their functions as individual web services. They have been implemented in the form of Docker Images and have been deployed as independent Docker Containers. Docker provides flexibility in the sense that each toolbox is implemented using extremely different languages and technologies, encouraging agile software development. A central front panel provides an easy-to-use interface for invoking all functionalities through graphical elements. The Dashboard is built using React in conjunction with MD-Bootstrap, ASP.NET Core and PostgreSQL5. The software project for analysis is retrieved from a Git repository. The user can invoke the toolboxes individually or perform a Central Analysis (for all quality attributes). Each toolbox has a page to inspect the produced results.

III. Evaluation - Demonstration

This Section presents the SDK4ED platform analysis results on an embedded healthcare application developed by Neurasmus (IMD). 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.

1) Energy: The first version of the IMD application was an emulator of the entire system, running on a linux PC (v1.3). Figure 2a depicts the results of Energy indicators. We have a relatively small number of Memory accesses (1546270), while the cache misses refer to the beginning of the application (initialization). A large number of data races is detected in the use of I/O C libraries (printf, scanf). The identified hotspots correspond to locks waiting for the rest of the threads and one hotspot is a loop statement. The toolbox proposes this loop as a candidate block for acceleration because there is a large instruction parallelism combined with a few control operations, while a significant part of operations are memory operations (not referring to the same address). However this loop has a small number of iterations and thus acceleration was not applied because we would have more delays on data transmission. In the next versions of the use-case (implemented on the hardware devices), static analysis energy estimation was used (Figure 3). This component provides an easy and fast way to estimate the energy and to compare the use of different platforms. Based on this analysis, we observe that selecting the microprocessor ARM Cortex M0+ is more energy efficient but with a penalty on the response time. Using this microcontroller can achieve energy savings up to 98% compared to using ARM Cortex A57. The Energy toolbox provides the option of adding more platforms (relevant guidelines are provided in Wiki page). A presentation of the estimations for all application revisions is also supported by the toolbox (Figure 2b). According to these results, the estimated energy is always lower when using ARM Cortex M0+, while there is an increase about in the middle of the project's history. In these versions, a cryptographic function was added. After measuring the energy overhead it was replaced by using a special peripheral and the CPU energy was reduced by 95%. Note that energy toolbox only analyses the software part of the application.

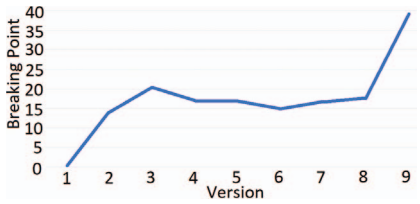


Fig. 4: Evolution of Breaking Point for IMD application

2) Maintainability: The results on the maintainability of the IMD application are presented on the basis of the three major TD aspects, namely principal, interest and interest probability. The principal varies largely across files. The highest principal was identified in file 'reader.cpp' requiring $302 to fix 29 identified code smells. The same file was found to suffer from an interest equal to $9.68, corresponding to the additional maintenance costs for developing each new version of the file. At the same time the interest probability is quite high (0.8 - a value of 1 would imply that the file is being changed in every revision of the system). Such values can alarm the development team about an imminent risk of costly maintenance. As an example of identified opportunities for improvement, it is recommended to add unit tests so as to cover 92 more lines of code need to reach the minimum threshold of 65% lines coverage. This issue requires substantial effort to be resolved (estimated at approximately 3 hours by SonarQube).

On the other hand, the SDK4ED TD toolbox identified files which have relatively high TD Principal but with limited TD Interest or interest probability. Such files should be assigned a low priority for TD repayment. Of particular interest is the evolution of the Breaking Point for the IMD application (Fig. 4) revealing a rather healthy overall status. The Breaking Point, i.e. the time point at which the accumulated additional maintenance costs will exceed TD Principal, lies for most of the analyzed versions 20 versions ahead, while its value reaches 40 for the last version. Based on these findings we can conjecture that while the IMD application suffers from the presence of some code-level inefficiencies, its overall design quality do not call for immediate repayment of TD: Even if the problems remain in the code, it is expected that the team will not face significant overhead in maintenance costs.

Finally, through a dedicated component , we have performed a separate TD analysis of the medical and the security components of IMD. The results suggested that the TD of the medical part is almost stable across the years of its evolution. At the timepoint of introducing additional security mechanisms TD has increased significantly; thereof, the TD of the complete system follows the evolution of the TD of the security component (see Fig. 5). This finding suggests an evident trade-off between guaranteeing security at the system level and code maintainability. Such analysis initiated a research line within Neurasmus considering the security costs in the IMD domain, urging for more cost-efficient solutions at the software level.

3) Dependability: The results of the security analysis, shown in Figure 6, indicate that the application is highly

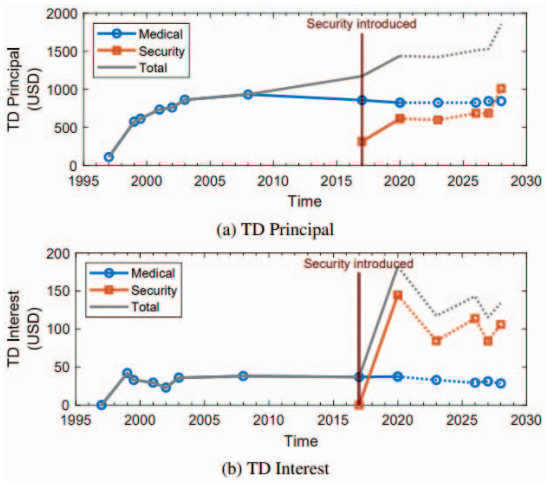


Fig. 5: Evolution TD in the two components of IMD

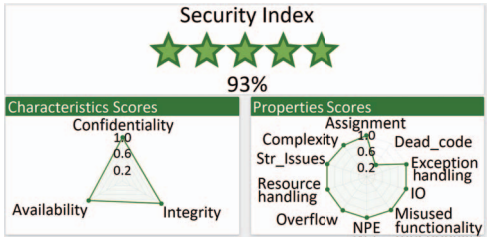


Fig. 6: Security Assessment Results

secure. Particularly, the *Security Index* was found to be 93%, which is a very high score, whereas the *Security Characteristics* of *Confidentiality, Integrity, and Availability*, were found to have a score above 86%. In addition, high security scores (above 94%) are assigned to almost all the low-level *Security Properties* (which corresponds to groups of security weaknesses), with the only exception of *Dead Code*, which received a low score (36%); however, this property is less critical compared to the others. It should be noted, that two important buffer overflow vulnerabilities were found and fixed using the security model through the course of the project.

In Figure 7, the results of the Vulnerability Prediction Models are illustrated in the form of a heatmap, where each rectangle corresponds to a specific source file, whereas the color denotes the probability of the corresponding source code file to contain vulnerabilities. More specifically, the greener the rectangle, the higher the probability of its associated file to contain vulnerabilities. As can be seen by Figure 7, three files are highly likely to contain vulnerabilities, namely the abc.c, the spi_interface.c, and the spi_interface_sisc.c, as they have a *vulnerability score* of 0.99, 0.95, and 0.81 respectively. This allowed the developers to pinpoint those source files that require more care from a security viewpoint.

Finally, with respect to reliability enhancement, Figure 8 shows an example of an actual computational loop retrieved

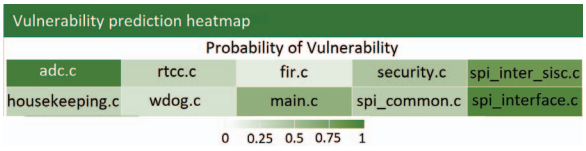
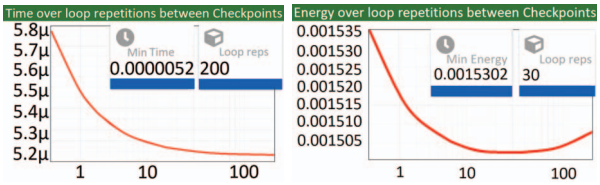


Fig. 7: Vulnerability Prediction Results



(a) Execution Time (b) Energy Consumption

Fig. 8: Optimum Checkpoint Interval

from the IMD application. Figure 8a suggests that a checkpoint should be generated every 200 loop repetitions in order to minimize the execution time of the application. On the contrary, it should be genereated every 30 repetitions in order to optimize its energy consumption as shown in Figure 8b.

4) Forecasting: Figure 9 illustrates the forecasting results for the Maintainability (i.e. TD) of the IMD application, having selected 10 commits as the forecasting horizon. The main screen of the TD Forecasting panel depicts an interactive plot that shows the past TD evolution (in green) followed by the forecasted evolution (in red). Tables with the detailed forecasted values are also provided. Based on these results, the TD of the IMD application is expected to increase during the next commits. While the current effort to repay TD is almost 5 days, in 10 commits from now it will have increased by 67% (∼8 days). This information helped the developers allocate the resources needed to quickly repay TD. Equally useful information were obtained also from the Energy and Dependability panels, where, with respect to the former the forecasts showed a slight increase during the next commits, while no big changes were expected with respect to the latter.

5) Decision Support: The output of the SDK4ED Decision Support toolbox is shown in Figure 10. Each refactoring has a total value (orange color) which can be broken down in individual impacts that the refactoring is expected to have on each of the three qualities. Positive-signed values denote improvement. The magnitude of each impact value is calibrated by the user's preferences: E.g. if energy is declared as more important than TD, the colored bars on the left would have bigger blue segments and smaller black segments.

IV. Conclusions

This work introduces SDK4ED, a software analysis framework that combines Energy, Maintainability and Dependability optimizations. A detailed description of each component is provided, along with the interaction between each other, to provide suggestions that help developers design embedded software. Using a healthcare embedded application as a use case, the proposed solution is demonstrated, emphasizing the integration of individual tools and the presentation of the analysis results through a unified platform.

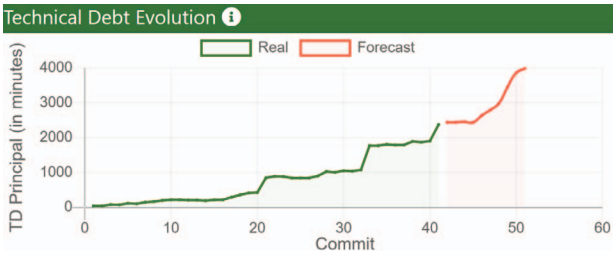


Fig. 9: TD Forecasting Results

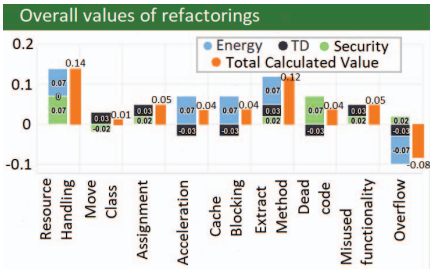


Fig. 10: Decision Support Results

References

[1] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing technical debt in software engineering (dagstuhl seminar 16162)," in *Dagstuhl Reports*, vol. 6, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[2] E. Gelenbe and M. Siavvas, "Minimizing energy and computation in long-running software," *Applied Sciences*, vol. 11, no. 3, p. 1169, 2021.

[3] S. Georgiou, S. Rizou, and D. Spinellis, "Software development lifecycle for energy efficiency: techniques and tools," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–33, 2019.

[4] C. Marantos, K. Salapas, L. Papadopoulos, and D. Soudris, "A flexible tool for estimating applications performance and energy consumption through static analysis," *SN Computer Science*, vol. 2, no. 1, pp. 1–11, 2021.

[5] I. Baldini, S. J. Fink, and E. Altman, "Predicting gpu performance from cpu runs using machine learning," in *2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing*, pp. 254–261, IEEE, 2014.

[6] A. Ampatzoglou, A. Michailidis, C. Sarikyriakidis, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "A framework for managing interest in technical debt: An industrial validation," in *Proceedings of the 2018 International Conference on Technical Debt*, TechDebt '18, (New York, NY, USA), p. 115–124, Association for Computing Machinery, 2018.

[7] A. Chatzigeorgiou, A. Ampatzoglou, A. Ampatzoglou, and T. Amanatidis, "Estimating the breaking point for technical debt," in *2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)*, pp. 53–56, 2015.

[8] M. Siavvas, D. Kehagias, D. Tzovaras, and E. Gelenbe, "A hierarchical model for quantifying software security based on static analysis alerts and software metrics," *Software Quality Journal*, vol. 29, no. 2, pp. 431–507, 2021.

[9] D. Tsoukalas, D. Kehagias, M. Siavvas, and A. Chatzigeorgiou, "Technical Debt Forecasting: An empirical study on open-source repositories," in *Journal of Systems and Software*, vol. 170, p. 110777, 2020.

[10] S. Guo and H. Zhao, "Fuzzy best-worst multi-criteria decision-making method and its applications," *Knowledge-Based Systems*, vol. 121, pp. 23–31, 2017.