Investigating Trade-offs between Portability, Performance and Maintainability in Exascale Systems

Elvira-Maria Arvanitou, Apostolos Ampatzoglou, Nikolaos Nikolaidis, Aggeliki-Agathi Tzintzira, Areti Ampatzoglou, Alexander Chatzigeorgiou

Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece
e.arvanitou@uom.edu.com, a.ampatzoglou@uom.edu.gr, it14189@uom.edu.gr, angeliki.agathi.tsintzira@gmail.com, ampatzoglou@gmail.com, achat@uom.gr

***Abstract*— Due to the rapid advancements in the hardware architectures of High-Performance Computing infrastructures, new challenges have arisen in the development of scientific software applications. In particular, software that runs on Exascale machines, needs to be highly portable, highly parallelizable and at the same time maintainable, since software for HPC evolves constantly over time. By taking into account that an overall optimization of all the aforementioned qualities is not realistic, in this study, we explore the possible trade-offs, when optimizing the run-time qualities of the software (i.e., performance and portability) through state-of-practice techniques in Exascale software development, in expense of code maintainability, as expressed by technical debt. To achieve this goal, we have performed a case study, in which the effect of run-time optimizations on technical debt has been measured. The results suggest that run-time optimizations tend to reduce TD principal, whereas the effect on interest is not consistent. The results are discussed in detail in this paper from the point of view of both researchers and practitioners.**

***Keywords*— technical debt; portability; performance**

I. Introduction

Exascale computing refers to applications capable of making a quintillion calculations per second. Such applications are usually developed for scientific purposes (such as physics, biology, etc.) that require simulations relying upon large volumes of data [5]. Due to the rapid evolution of the hardware of Exascale systems, developers cannot have a complete view of its internal structure [11]. Therefore, the ***programming*** of future supercomputing architectures will become significantly more ***challenging***, in the sense that future architectures will become more ***parallel*** and applications will have to be able to exploit the parallelism at different levels. Additionally, architectures will become more ***heterogeneous*** involving different type of processor cores and accelerators (such as GPUs and FPGA boards). Thus, developers do not only have to be able to exploit the capabilities of these more complex hardware architectures, but software applications need to remain (performance) portable. These challenges are vividly explained in the Future and Emerging Technologies (FET) calls for HPC, suggesting that there is a need for programming methods that will enhance portability and performance. For further supporting the importance of these qualities in Exascale applications, Carver et al. [9] suggested that ***functional correctness***, ***performance***, ***portability***, and ***maintainability*** are considered by developers as the most important for Exascale systems.

By considering the fact that functional correctness is not negotiable, the rest qualities need to be safeguarded. Therefore, ***optimizations*** for improving the levels of the corresponding quality attributes (e.g., performance, portability, maintainability) are required. This problem is wicked in the sense that it is often hard to find solutions that balance and optimize a variety of quality attributes, since various ***trade-offs*** appear: for instance, the use of polymorphism improves the extendibility (a sub-characteristic of maintainability) of the system, but incurs a significant performance penalty. Trade-offs occur because almost every architectural decision has the potential to positively affect some quality attributes and negatively affect others. Therefore, it is vital to understand the nature of a trade-off, to achieve the right balance between quality attributes [4], rendering the decision to apply the optimization a fully informed one.

The goal of this study focuses on the aforementioned qualities (*namely*: portability, performance and maintainability), and explores if there are trade-offs between them, when optimizing Exascale systems. To achieve this goal, we performed a case study on 6 Exascale projects, which employ state-of-practice tools for enhancing portability (*SkePU* [14]) and performance (*StarPU* [12]) and explore the effect of using these tools on maintainability. Software maintainability is assessed through an emerging notion on the software development community, termed Technical Debt (TD). TD has been introduced [3] to monetize the financial costs that arise, along maintenance: TD refers to the shortcuts taken along development (e.g., in terms of shorter delivery time) that may have negative impact on software qualities, e.g., maintainability. The TD metaphor relies on two basic concepts: ***TD principal*** (i.e., the effort required to refactor the software, so as to improve its quality) and ***TD interest*** (i.e., the extra effort needed along software maintenance, due to the existence of TD principal). In this study, we quantify these concepts, based on the FITTED framework—proposed by Ampatzoglou et al. [1], and validated in an industrial setting [26]. We note that details on SkePU [14], StarPU [12], and TD quantification [1] are not discussed in this paper, due to space limitations, and can be accessed in the aforementioned original studies.

II. Related Work

Literature on software quality mostly focuses on the optimization of separate qualities, such as maintainability, usability, security, etc. [4]. However, during the development process, the effort to optimize one software quality attribute might negatively affect another. Thus, the struggle to achieve a higher software quality level is subject to many trade-offs. Since the amount of research in this domain is very large, this section includes only an indicative sample of this corpus of studies, due to space limitations. Buyens et al. [6] analyze the trade-offs in three cases between security and maintainability. Security is measured by two metrics: the number of violations and the estimation of the attackers' effort. On the other hand, maintainability is measured by two coupling metrics. The results suggest that it is more effective to apply transformations jointly, and indicate the existence of trade-offs between the qualities of security and maintainability.

DOI 10.1109/SEAA51224.2020.00020

Additionally, Feitosa et al. [15] focus on the existence of: (a) quality trade-offs in critical embedded systems (CES) by analyzing their implemented architecture through evolution; and (b) different trade-offs between critical embedded systems and systems from other domains. The results suggest that quality trade-offs are usually in favor of critical qualities, and in expense of non-critical ones. In a similar fashion, Papadopoulos et al. [20] analyze the trade-offs between design-time and run-time qualities in the field of embedded systems. The results have empirically validated the existence of trade-offs between run- and design-time qualities.

III. Case Study Design

To explore the relation between: (a) the application of performance / portability optimizations; and (b) their effect on maintainability, we performed a case study on six Exascale projects. In this section we describe the study design, according to the guidelines of Runeson et al. [22]. The reason for conducting a case study was that we aimed at investigating real-world projects that apply performance and portability optimizations (as they are performed in practice), without controlling their consequences on maintainability.

Objective and Research Questions. The goal of this study, described using the Goal-Question-Metric (GQM) formulation is: "to *analyze* portability and performance optimizations for *the purpose of* understanding possible unintentional trade-offs *with respect to* software maintainability *from the point of view of* software engineers, *in the context of* Exascale software development". Although research in the field of Technical Debt Management (TDM) has been very active during the last years, the accumulation and consequences of TD through the development of Exascale systems has not been investigated in the literature [25]. We take into consideration the fact that the use of optimization approaches, such as SkePU or StarPU, result either to the ***modification*** (***refactoring***) of existing files, or to the addition of ***new code***. The reason differentiating the two types of files is to check if the new code that is introduced for applying the optimization is different from the refactored code (due to the application of the optimization). The differences of TD between ***new*** and ***refactored code*** have been discussed by Arvanitou et al. [2]. Based on the above, we have extracted two research questions (RQs). In each one we focus on the two main concepts of TD, namely: principal and interest. Despite the fact that intuitively TD principal is expected to co-evolve with interest (i.e., the higher the principal, the higher the interest that it is produced), in some cases (such as reuse), they appear to be not correlated [16].

RQ$_1$: *What is the effect of performance and portability optimization on software maintainability, in the refactored parts of the source-code?*

To answer this research question, we compare the levels of TD principal and TD interest of the files that are modified along the optimization. The results on performance (SkePU) and portability (StarPU) are treated separately, since they obey to different transformation rules. Through this question, we aim at investigating if the modifications of the source-code, due to the application of tools that improve performance or portability lead to more or less maintainable code.

RQ$_2$: *What is the effect of performance and portability optimization on software maintainability, in the new parts of the source-code?*

To answer this research question, we compare the levels of TD principal and TD interest of the files that are introduced (e.g., libraries) due the optimization process. Similarly to RQ$_1$, the results on performance (SkePU) and portability (StarPU) are treated separately. Through this research question, we aim at investigating if the files that are introduced along of the transformation exhibit higher or lower maintainability compared to the average existing files.

Case Selection and Unit of Analysis. According to Runeson et al. [22], our study is characterized as an embedded multiple case study, as we investigate multiple units of analysis (i.e., files) extracted from various cases (i.e., Exascale projects). Despite the plethora of available Exascale as open-source, we have selected to limit our case selection process to a convenience sample consisting of projects that: (a) we are aware of the commit in which a SkePU or StarPU transformation has been performed; and (b) we are aware that in the aforementioned commit, limited other changes (apart from the quality optimization) have been performed.

TABLE I. Projects considered in the case study

Project	Language	Tool	#files		Provider
			(b)	(a)	
Co2Capture	Fortran	SkePU	25	35	CERTH
Metalwalls	Fortran / C++	SkePU StarPU	42 41	42 42	CNRS
Pastix	Fortran / C	StarPU	10 0	10 0	INRIA
QR-mumps	Fortran / C	StarPU	91	10 2	JULICH
Rodinia	C / C++	StarPU	4	14	LIU
ParseC	C / C++	SkePU	36	65	LIU

Due to the aforementioned limitations, we were not able to blindly search for open-source projects, but we had to refer to specific Exascale application providers. In particular, we have used six consortium-owned projects (see Table I). Apart from the project name, we report the programming language, the tool used for the optimization, the number of files before (*b*) and after (*a*) the quality optimization, and the partner that has provided the tool and applied the optimization.

Data Collection and Pre-processing. For every project we have analyzed two versions: *before optimization* and *after optimization*, and we recorded several variables. We note that the selection of these variables (as well as the argumentation of how they related to TD concepts) is presented in detail by Ampatzoglou et al. [1]: (a) ***Number of Code Smells***—NCS (*TD Principal*); (b) ***Number of Functions***—NOF (*TD Interest*); (c) ***Complexity***—CC (*TD Interest*); (d) ***Lines of Code***—LoC (*TD Interest*); (e) ***Comments Ratio***—CR (*TD Interest*); (f) ***Fan-Out***—FO (*TD Interest*); and (g) ***Lack of Cohesion of Lines***—LCOL (*TD Interest*).

Next, by comparing the file names and sizes (in terms of KBs and LoC) in the before and after versions, we characterized each file as: `NEW`, `REFACTORED`, or `UNCHANGED`. Then, we have performed the following data transformations:

- for each `REFACTORED` file, for every metric, we calculate the difference between the before and after version. To ensure the uniform interpretation of the difference variable, the order of the subtraction ensures that negative differences correspond to negative effect of the transformation; whereas positive differences to positive effect. Thus, for CR, we have calculated `DIFF` as `AFTER-BEFORE`, whereas for the rest `BEFORE-AFTER`.
- for each `NEW` file, for every metric, we first calculate the mean value of the metric in the before version. Next, we

calculate the difference of the metric between the after version and the mean value in the before version. Similarly, regarding CR, we have calculated `DIFF` as `AFTER-MEAN`, whereas for the rest as `MEAN-AFTER`.

Next, by acknowledging the need to synthesize the six TD interest proxies in one variable, we have relied on the FITTED framework [1], and we calculated the unified ***TD Interest Proxy***. Therefore, the final dataset of the study contains the following variables: [***V1***] Filename; [***V2***] Used Tool for Optimization (`SkePU` or `StarPU`); [***V3***] File Type (`NEW`, `REFACTORED`, or `UNCHANGED`); [***V4***] TD Principal Proxy ($DIFF_{NCS}$); and [***V5***] TD Interest Proxy.

Data Analysis. As a first step for our data analysis we have performed a descriptive statistical analysis on the two datasets (*before* and *after optimization*) for all raw metrics (see aforementioned bullet list), for each project. Next, to answer each research question, we have performed statistical hypothesis testing to investigate if variables [V4] and [V5] differ from `zero`, i.e., to explore if the mean effect of the optimization differs statistically significant from having no effect (`DIFF=0.0`). To perform hypothesis testing with a single variable, we applied the one-sample t-test, using as `testing variables` [V4] and [V5], whereas as `test value 0`. More specifically, for RQ_1, we filter the dataset using [V3], selecting only files that are `REFACTORED`, whereas for RQ_2, we retained only files that are `NEW`. Finally, while reporting both research questions, we split the dataset, based on [V2].

IV. Results

We present the results of the case study organized by research question. Based on a descriptive analysis (omitted due to space limitations), we can observe that the cases in which the *before version* is better is 47%, and the cases that *after version* excels is 53%. To investigate: (*a*) if the aforementioned means present statistically significant differences; (*b*) if the same differences appear for refactored and new code in isolation; and (*c*) the differences on interest when all metrics are synthesized; we present next an in-detail analysis.

A. Effect of SkePU / StarPU Transformations (Refactoring)

In Table III, we present the results of the one value t-test on the mean difference of the variable before and after the optimization, from zero. Both differences are calculated first at a file level, and then a grand average (on all files, without a per project assessment) is calculated.

TABLE II. Effect of Transformation on TD for Refactored Code

Optimization	TD Concept	Mean	t-value	sig.
SkePU	TD Principal	19.8%	2.591	*0.014*
	TD Interest	2.8%	0.989	0.336
StarPU	TD Principal	16.6%	3.256	*0.001*
	TD Interest	-5.2%	-2.397	*0.019*
* *Across projects the range of difference values for principal and interest is [-390%, 200%] and [-230%, 160%]*				

The results suggest that all mean differences (apart from TD interest for StarPU) are positive, i.e., the value of TD principal or TD interest has decreased (i.e., improved) due to the optimization. This result is statistically significant in all cases, apart from TD Interest for SkePU. However, the differences (see column Mean) in absolute values appear to be small for TD Principal, and marginal for TD Interest. To dig further into the aforementioned cases, in terms of the frequencies of refactored files, in Fig. 1, we present bar charts on frequency of files that have been positively or negatively affected. The left part of Fig. 1 refers to the SkePU transformation, whereas the right part to the effect of the StarPU transformation. By contrasting the results of Table II and Fig. 1, we can observe that in terms of frequency, the transformations appear to have a negative effect in at least half of files for both SkePU and StarPU. Given that for TD Principal (in the StarPU case) the mean score of a set of values (that comprises 60% of negative numbers and 40% of positive numbers) is positive, we can deduce that in absolute values, the positive numbers are higher than the negative ones. This observation leads to the conclusion that the positive effect of StarPU on TD (when it appears) is higher (in magnitude), compared to the cases of negative effect. This observation also applies to all other cases.

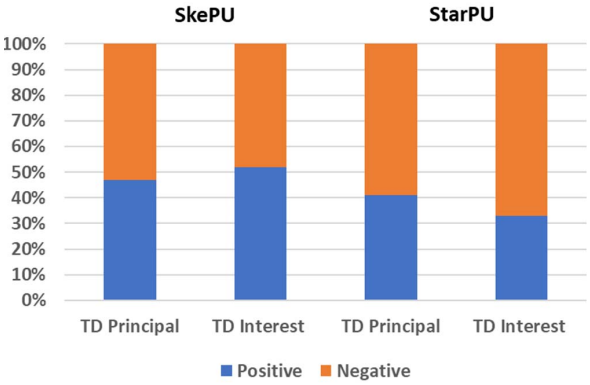

Fig. 1. Frequency of positive and negative effect

B. Effect of SkePU / StarPU Transformations in New Files

Similarly to RQ_1, in Table III, we present the results of the one sample t-test, that assesses the TD Principal and Interest, in new files introduced while performing SkePU / StarPU transformations, compared to the rest files.

TABLE III. Effect of Transformation on TD for New Code

Optimization	TD Concept	Mean	t-value	sig.
SkePU	TD Principal	34.7%	2.466	*0.018*
	TD Interest	103.7 %	3.229	*0.003*
StarPU	TD Principal	35.0%	3.403	*0.001*
	TD Interest	31.6%	2.030	*0.047*
* *Across projects the range of difference values for principal and interest is [-120%, 99%] and [-99%, 226%]*				

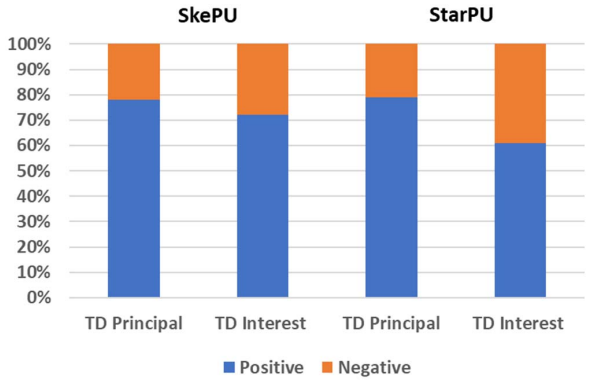

Fig. 2. Frequency of positive and negative effect

From the results of Table III, we can observe that all mean differences are positive (i.e., the transformations improve TD

Principal and Interest according to the mean values), and that all differences are statistically significant. In Fig. 2 we present the frequency of new files having a higher or lower TD Principal and Interest compared to the rest of the system while performing SkePU and StarPU transformations. In contrast to RQ_1 (effect on the refactored code), the effect of new code seems lower, in the sense that the mean values and the frequency investigations comply. Based on Fig. 2, approx. 80% of the files that are introduced along with the ***SkePU*** transformation have lower TD Principal and Interest, compared to the average values of the system. A similar observation can be made for new files of ***StarPU*** transformations, but on a lower rate, especially for TD Interest.

V. Discussion

In this section we summarize the main findings by interpreting the results, comparing them to existing literature, and providing implications to researchers and practitioners. As a first step towards an effective discussion, in Table IV, we summarize the main findings of our case study.

TABLE IV. Summary of Results

Optimization		TD Concept	Frequency	Absolute Effect
SkePU	Refactored	TD Principal	Equal	Strong for Positive
		TD Interest	Equal	Strong for Positive
	New	TD Principal	**Positive**	Strong
		TD Interest	**Positive**	Strong
StarPU	Refactored	TD Principal	**Negative**	Strong for Positive
		TD Interest	**Negative**	Strong
	New	TD Principal	**Positive**	Strong
		TD Interest	**Positive**	Limited

Interpretation of Results. Regarding SkePU we can claim that no important trade-offs are observed, in the sense that portability improvements offered by SkePU are not substantially hurting the maintainability of the system in terms of TD. More specifically, the results suggest that the new code that is introduced is of better quality compared to existing one, and for the refactored code the effects are balanced (approx. half of the files are positively affected and the other half are negatively affected). On top of this, it seems that positively affected files are more intensively affected, making the overall assessment positive (even statistically significant). Nevertheless, there is still room for improvement, esp. regarding the TD Interest of refactored files. An example of how SkePU transformations improves interest is presented in Fig. 3.

The top part of Fig. 3 corresponds to the source code of array dot product implementation with SkePU, whereas the lower part without SkePU. Both implementations have the same CR, FO, NOF, and LCoL (zero—number of cohesive pair of lines is higher or equal, compared to non-cohesive ones). The cyclomatic complexity of the SkePU implementation is 1, whereas for the non-SkePU implementation CC equals 2; also, the LoC of the non-SkePU implementation is higher by 1 line. Therefore, the TD interest of the non-SkePU solution is higher than the SkePU implementation. On the other hand, regarding StarPU, trade-offs are more evident: the use of StarPU guarantees the performance of the system, but it seems to hurt the maintainability of refactored files (more than 60% of them). However, similarly to before, the new files that are added seem to have better levels of TD Principal and Interest compared to the rest of the code. Therefore, in this kind of transformation there is again room for improvement, placing special emphasis to refactored code and the produced interest. The existence of trade-offs between run-time optimizations and maintainability is an expected outcome, since literature has reported similar findings [4][6][15]. Similarly, the findings of the study comply with existing literature on the correlation between TD Principal and TD Interest, since in all four cases, the effect of the transformation on both concepts of TD was uniform. Nevertheless, it is important to stress out that TD Interest seems more difficult to handle and more vulnerable to trade-offs compared to TD Principal. A tentative explanation on this is the fact that TD Interest is calculated as collection of usually conflicting quality properties (e.g., coupling vs. cohesion; size vs. complexity) in contrast to Principal.

```
float prod(float a, float b) {
        return a * b;
}

Vector<float> vector_prod(Vector<float> &v1, Vector
<float> &v2) {
        auto vsum = Map<2>(prod);
        Vector<float> result(v1.size());
        return vsum(result, v1, v2);
}
================================================
float prod(float a, float b) {
        return a * b;
}

Vector<float> vector_prod(Vector<float> &v1, Vec-
tor<float> &v2) {
        Vector<float> result;
        for (int i=0; i<v1.size(); i++) {
          result.push_back(prod(v1[i], v2[i]));
        }
        return result;
}
```

Fig. 3. SkePU Transformation - TD Interest Illusatrtion

Implications for Practitioners. Based on the findings we suggest practitioners to consider the effect of run-time quality optimizations on maintainability. By considering that the improvement of run-time qualities is non-negotiable in HPC applications, we highlight the most frequent pitfalls (while applying SkePU or StarPU) so that practitioners have them in mind and avoid them in future transformations. In Table V, we list the most frequent types of TD Items that are related to the SkePU / StarPU transformations. We list all the rules that are introduced in the new code and the refactored files. With red cell shading we denote the rules that appear in the top-10 most frequently violated rules of each project, whereas top-20 most frequently violated rules are denoted with yellow cell shading. The least frequently violated rules are denoted with green cell shading. The criterion for one smell to be included in the table was its occurrence in at least 3 projects. Based on the findings of Table V, we can encourage practitioners to try to avoid the "*Magic Number*", "*Missing Curly Braces*", and rule violations when applying StarPU and SkePU transformations in C/C++ code, and the "*Float Compare*", "*Check Code Return*", and the "*Exit Loop*" rule violations, when working with Fortran code.

Implications for Researchers. Regarding researchers, several future work opportunities can be highlighted. The most interesting future direction that we plan to pursue is to perform explanatory studies that would unveil the reasons for which the smells presented in Table V are introduced. This could be achieved through longitudinal studies and single-project analysis. Possible factors that might influence this effect are: (a) the overall frequency of these rules; (b) the culture of teams or the application domain; or (c) the specifics of the transfor-

mations of the two tools. Additionally, we encourage researchers studying the structural implications of StarPU and SkePU, since they both affect interest.

TABLE V. CODE SMELLS FREQUENCY

Code Smell	Project					
c:ClassName						
c:CommentedCode						
c:FileHeader						
c:FunctionCognitiveComplexity						
c:FunctionComplexity						
c:FunctionName						
c/cxx:MagicNumber						
c/cxx:MissingCurlyBraces						
c/cxx:MissingIncludeFile						
c:ReservedNames						
c/cxx:StringLiteralDuplicated						
c/cxx:TabCharacter						
c:TooLongLine						
c/cxx:TooManyParameters						
c:TooManyStatementsPerLine						
c/cxx:UndocumentedApi						
common-c:DuplicatedBlocks						
common-c:InsufficientCommentDensity						
common-c:InsufficientLineCoverage						
F-rules:COM.DATA.FloatCompare						
F-rules:COM.FLOW.CheckCodeReturn						
F-rules:COM.FLOW.ExitLoop						

VI. THREATS TO VALIDITY

The results are subject to ***generalization*** threats pertaining both to the analyzed projects and the selected optimization tools. In other words, we cannot argue that StarPU and SkePU optimizations will reduce TD principal in every Exascale software project, as only six projects have been employed. Similarly, it cannot be claimed that any other kind of performance or portability optimization, beyond those applied by SkePU and StarPU will lead to consistent results. Further research is required to validate these findings and to delve deeper into the reasons that optimizations affect software qualities. In terms of ***construct validity*** threats, we have to stress that to assess the impact on TD principal and interest, a specific tool (SonarQube) and selected structural metrics have been used. SonarQube assesses mainly the so-called code and design debt and pays less emphasis on inefficiencies at the higher levels of a software system (i.e. architecture). However, due to the nature of the applied performance and portability optimizations we would not anticipate changes beyond the code and design level of the impacted software. For TD interest, we have relied on proxies of interest, as the concept of interest (i.e., additional maintenance effort due to the presence of TD) is hard to quantify. However, the used interest proxies are widely acknowledged as indicators of maintainability. Finally, with respect to the ***reliability*** of the findings, the described methodology outlines all steps followed to conduct the case study, and a replication package is provided online.

VII. CONCLUSIONS

The continuous advancements in High-Performance Computing infrastructures have resulted in methods and tools supporting performance and portability optimizations in Exascale applications, such as SkePU and StarPU. At the same time HPC software is becoming increasingly complex and is also subject to continuous evolution, calling for increased maintainability. To shed light into potential tradeoffs between performance/portability optimizations and software quality, as captured by the popular Technical Debt metaphor, we have performed an empirical study on six Exascale applications. The results reveal that in the majority of cases SkePU is not hurting the maintainability of the system, whereas StarPU seems to have a negative effect on the maintainability of refactored code. Nevertheless, the majority of issues introduced are common; therefore, the creation of a strategy to optimize the refactoring seems feasible.

ACKNOWLEDGMENT

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 801015 - EXA2PRO

REFERENCES

[1] Ar. Ampatzoglou, A. Ampatzoglou, P. Avgeriou, and A. Chatzigeorgiou, "A Financial Approach for Managing Interest in Technical Debt", International Symposium on Business Modeling and Software Design (BMSD'15), Milan, Italy, 6 – 8 July 2015.

[2] E. M. Arvanitou, A. Ampatzoglou, S. Bibi, A. Chatzigeorgiou, and I. Stamelos. "Monitoring Technical Debt in an Industrial Setting", 23rd Proceedings of the Evaluation and Assessment on Software Engineering (EASE '19), Copenhagen, Denmark, 14 – 17 April 2019.

[3] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing Technical Debt in Software Engineering", Dagstuhl Reports, 2016.

[4] S. Barney, K. Petersen, M. Svahnberg, A. Aurum, and H. Barney, "Software quality trade-offs: A systematic map", Information and Software Technology. 54 (7), pp. 651–662, July 2012.

[5] C. K. Birdsall and A. B. Langdon, "Plasma Physics via Computer Simulation", Adam Hilger Series on Plasma Physics. New York, 1991

[6] K. Buyens, R. Scandariato, and W. Joosen, "Measuring the interplay of security principles in software architectures", 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM'09), Lake Buena Vista, FL, USA, 15-16 October 2009.

[7] J. C. Carver, R. P. Kendall, S. E. Squires and D. E. Post, "Software Development Environments for Scientific and Engineering Software: A Series of Case Studies," 29th International Conference on Software Engineering (ICSE'07), Minneapolis, MN, 20-26 May 2007.

[8] G. Da Costa, et. al., "Exascale Machines Require New Programming Paradigms and Runtimes", Supercomputing Frontiers and Innovations: an International Journal, 2 (2), 2015.

[9] U. Dastgeer, and C. Kessler, "Flexible Runtime Support for Efficient Skeleton Programming on Heterogeneous GPU-based Systems", Advances in Parallel Computing, 22 (7), pp. 159 – 166, August 2011.

[10] A. Ernstsson, L. Li, and C. Kessler, "SkePU 2: Flexible and Type-Safe Skeleton Programming for Heterogeneous Parallel Systems", International Journal of Parallel Programming, 46, pp. 62–80, 2017.

[11] D. Feitosa, A. Ampatzoglou, P. Avgeriou, and E. Y. Nakagawa, "Investigating Quality Trade-offs in Open Source Critical Embedded Systems", 11th International Conference on Quality of Software Architectures (QoSA '15), Montréal, Canada, May 2015.

[12] D. Feitosa, A. Ampatzoglou, A. Gkortzis, S. Bibi, and A. Chatzigeorgiou, "Code Reuse in Practice: Benefiting or Harming Technical Debt", Journal of Systems and Software, Elsevier, 2020.

[13] L. Papadopoulos, C. Marantos, G. Digkas, A. Ampatzoglou, A. Chatzigeorgiou, and D. Soudris, "Interrelations between Sotware uality Metrics, Performance and Energy Consumption in Embedded Applications", 21st International Workshop on Software and Compilers for Embedded Systems (SCOPES '18), Germany, May 2018.

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

[15] D. Soudris, et al., "EXA2PRO programming environment:Architecture and Applications", 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, 2018.

[16] A. A. Tsintzira, A. Ampatzoglou, O. Matei, A. Ampatzoglou, A. Chatzigeorgiou, and R. Heb, "Technical Debt Quantification through Metrics: An Industrial Validation", 15th China-Europe International Symposium on Software Engineering Education, Portugal, 2019.