SDK4ED: A Platform for Technical Debt Management

Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Elvira Maria Arvanitou, Stamatia Bibi
Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece
Department of Electrical and Computer Engineering, University of Western Macedonia, Kozani, Greece
a.ampatzoglou@uom.edu.gr, achat@uom.edu.gr, e.arvanitou@uom.edu.gr, sbibi@uowm.gr

Technical Debt Management is of paramount importance for the software industry, since maintenance is the costlier activity in the software development lifecycle. In this paper we present the SDK4ED platform that enables efficient technical debt management (i.e., measurement, evolution analysis, prevention, etc.) at the code level, and evaluate its capabilities in an industrial setting. The SDK4ED platform is the outcome of a 3-year project, including several software industries. Since, the research rigor of the approaches that reside in SDK4ED have already been validated, in this work we focus: (a) on the presentation of the platform per se; (b) the evaluation of its industrial relevance; (c) the usability of the platform; as well as (d) the financial implications of its usage.

1. Introduction

Technical Debt (TD) is a software engineering metaphor capturing the costs and benefits of producing immature software artifacts [3]. The metaphor resembles the process of releasing products of sub-optimal maintainability to going into debt. The amount that the company saves (internally) from the sub-optimal software development is termed as TD Principal; whereas the extra costs that arise along maintenance, due to the poor quality of the product is termed TD Interest [1]. The generation of TD Interest is not certain; in the sense that its occurrence is related to the probability of software modules to undergo maintenance, termed as TD Interest Probability [13]. By considering that zero-TD is not feasible in practice [7], it is of paramount importance to keep the amount of TD low only in design hotspots (i.e., modules with high interest probability) and prevent the increase of TD along software evolution. The process of safeguarding the levels of TD in a software system is termed Technical Debt Management (TDM). According to Li et al. [10], efficient TDM includes 8 activities:

- Identification—identify the artifacts (e.g., classes) that suffer from TD
- Quantification—quantify (in monetary terms) TD principal, interest, and interest probability
- Prioritization—rank the identified TD items, based on their urgency to be resolved
- Prevention—prevent the future accumulation of TD
- Monitoring—continuously perform TDM activities, along project evolution
- Repayment—resolve the problems of TD items (e.g., by refactoring)
- Representation / Documentation—visualize and persistently store all the data from TDM
- Communication of TD—offer a view of TDM to all stakeholders

Between 2018 and 2020, the SDK4ED consortium[1] proposed various novel approaches for TDM (leading to more than 20 publications–see Section 2.2), and implemented them as part of the SDK4ED platform (see Section 2.3). To the best of our knowledge SDK4ED is the first platform that comprehensively supports the complete range of TDM activities at the source code level[2]. We note that with the term "comprehensively" we do not imply that all possible ways to perform an activity are covered (e.g., we do not apply portfolio management for TD prioritization). A detailed comparison with other existing TDM solutions is provided in Section 2.2. In Figure 1, we provide a bird's eye view on how SDK4ED synthesizes the aforementioned TDM activities in a process that can be executed through

[1] https://sdk4ed.eu/

[2] We note that various other types of TD exist (e.g., requirements, build, version, etc.), but are not treated by the SDK4ED project

the SDK4ED platform. First, while coding, a continuous ***TD Prevention*** activity takes places, attempting to limit the amount of TD that is introduced in the system. However, since this process cannot be 100% efficient and zero TD is unrealistic [7], there will be a need for ***TD Identification***. In this step, a long list of ranked artifacts that suffer from TD will be generated. The presentation of TD items will be accompanied by the outcomes of ***TD Quantification*** and ***TD Prioritization***. This process will be performed along evolution offering continuous ***TD Monitoring***. For selected (by the engineers) artifacts, opportunities for ***TD Repayment*** are presented, so as to reduce their TD Principal so as to reduce their TD Principal, as well as the accumulation of future TD Interest. All the aforementioned activities are visualized in dashboards and a full history of all monitored indices are persistently stored, in order to achieve ***TD Representation and Documentation.*** Finally, we note that the platform is web-based so that all company stakeholders (e.g., architects, developers, managers, etc.) can have access to the data, being provided with different views based on their roles, enabling ***TD Communication***. At the current development stage, the platform supports the aforementioned activities for the `Java`, `C`, and `C++` programming languages.

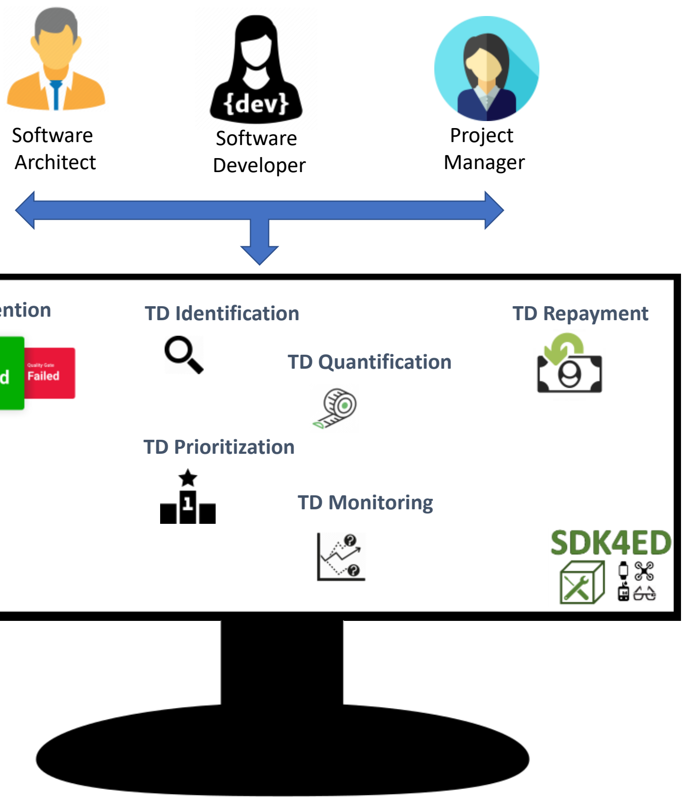


Figure 1. SDK4ED in a Nutshell

In this experience report, we present the process of building the SDK4ED platform, the platform itself, as well as the experience of industrial stakeholders when using the platform. The novelty of this work compared to the various studies published in the course of the SDK4ED project is that this is the first work that presents the SDK4ED as a whole, validating three important aspects that support its application in practice: (a) industrial relevance; (b) usability, and (c) economic benefits that it can bring to software companies. In Section 2, we provide background information that facilitate the understanding of this work, i.e., the process for requirements elicitation, as well as the requirements of the platform itself, an overview of the research approaches on which the SDK4ED approaches rely upon, and a walkthrough on the platform. In Section 3, we present the case study design, whose results we report and discuss in Section 4. In Section 5 we present the limitations/threats to validity; whereas in Section 6, we conclude the paper.

2. Setting the Scene

2.1 Industrial Requirements and Platform Capabilities

In this section, we present the process that we have followed for eliciting the requirements to specify the SDK4ED platform. Along requirements elicitation, we have used a number of complementary requirements analysis techniques [17]: first, we overviewed the literature and consulted with the companies involved in the project to develop baseline mockups on the platform, as well as its main features. Based on the literature, we decided to provide support for the 8 TDM activities presented in Section 1. Among them, ***documenting***, ***representing***, and ***communicating*** TDM were satisfied by our decision to provide a visual web-based dashboard that would be accessible by all stakeholders (with different roles), and use persistent storage for documentation. Thus, the following questions had to be answered, so as to support TD quantification, identification, prioritization, reduction, and prevention:

- What would be the preferred way to quantify TD?
- At what level (class or system) shall TD be identified?
- What information would be more interesting for the developer, when prioritizing?
- How would practitioners prefer to reduce TD?
- Are these responses affected by the role of the stakeholder?

To answer the aforementioned questions in a way that the SDK4ED platform is as relevant to the software industry as possible, we sought answers through a survey with 60 software engineers working for 11 software industries (located in 9 countries) [2]. This survey was broader than the consortium of the project, so as to boost generalizability. More details on the design of that study can be found in the original paper [2]. The main findings of the survey are visualized in Figure 2, and outlined below:

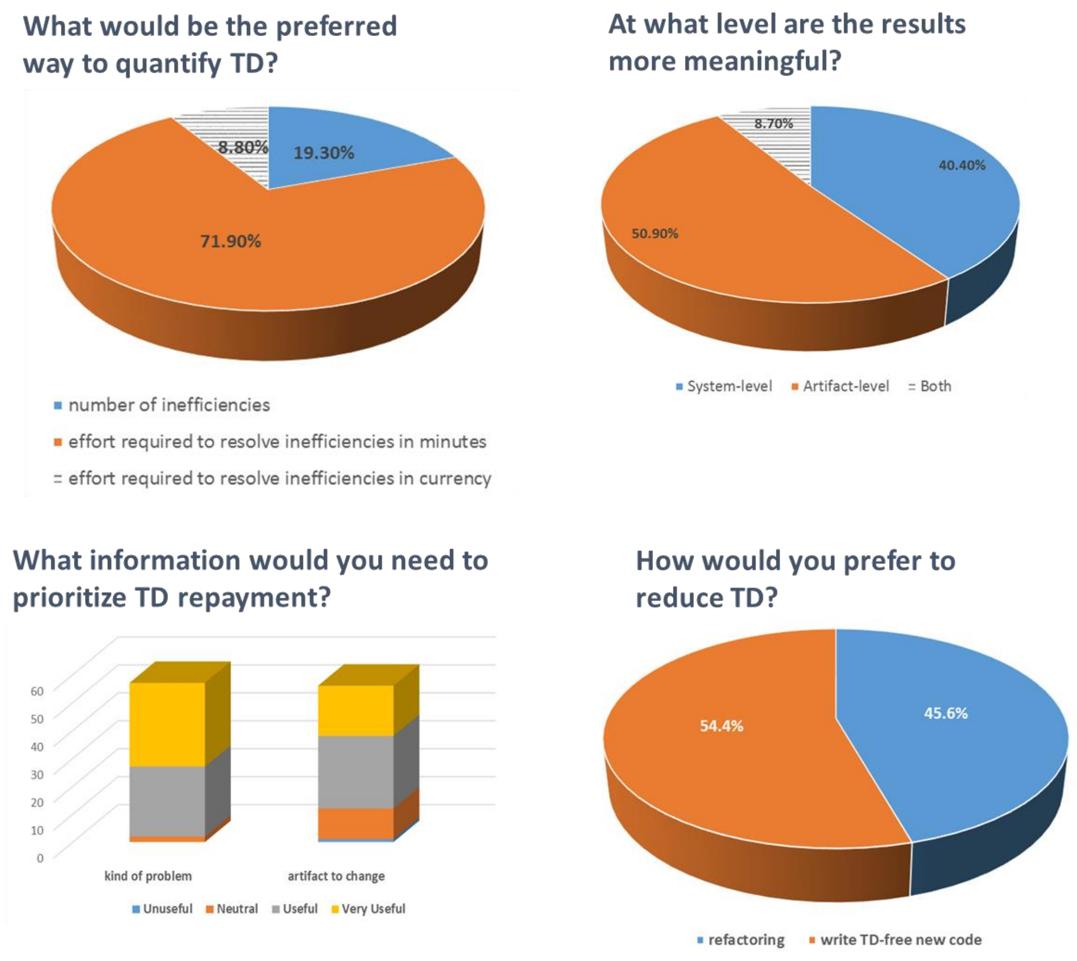


Figure 2. Requirements Analysis Overview

TD Quantification: Stakeholders prefer to see effort-related information of inefficiencies[3] and improperly designed / developed artifacts. This preference suggests that the TD metaphor (relating poor software development to effort) is useful in practice. ***Level of Detail for Reporting***: A marginal majority of stakeholders support the reporting at the artifact level, and not system-wide. Artifact-level reporting can be more easily related to actionable results in the sense that it points to parts of the system that need redesign. System-wide reporting provides a high-level view of quality, which does not lead to specific actions. Due to the importance that practitioner gave to both options; we have selected to implement both. ***TD Prioritization Parameters***: The question on TD prioritization was providing two options to the users: to prioritize based on the "kind of problem" (i.e., the specific rule of SonarQube violated in a TD item) or the characteristics of the "artifact to change" (i.e., the characteristic of the TD item per se). Stakeholders believe that they shall start repaying TD from artifacts that suffer from specific problems (i.e., artifacts to change in the bar chart), and not based on the artifact (e.g., the probability of the artifact to produce TD interest). ***TD Reduction Strategies***: Writing TD-free new code, or developing TD-free new artifacts seems as a more promising strategy for reducing TD, compared to refactoring existing code. However, both practices seem to be applicable in practice. ***Stakeholders' View***: The main differences of the managers' view compared to the study corpus are that they are more interested: (a) in monetary views and (b) system-wide evaluations, compared to the rest stakeholders. The first observation confirms the belief of the TD community that the TD metaphor can bridge the gap between technical and managerial stakeholders.

Based on the previously presented findings, we have selected to support the remaining 5 TDM activities, as presented in Table 1. The way that these features have been implemented resulted from the main research outcomes of the SDK4ED project (either by novel research or by reusing existing tools/methods). A detailed description of how these research goals have been satisfied is discussed in Section 2.2.

Table 1. SDK4ED Features

TDM Activity	Feature	Description
Identification Quantification	System-Level Analysis Artifact-Level Analysis	Quantification will need to be performed at artifact and system level. Identification by definition can only be performed at artifact level. The metrics that we need to measure are: TD Principal, TD Interest, and Interest Probability.
Prevention	New Code Analysis Quality Control	Prevention will need to rely on methods that would be able to continuously control the quality of the new code, and prevent low quality (high TD code) to be introduced in the system.
Monitoring	Evolution Analysis	Monitoring will be performed by recording the evolution of metrics scores related to TD (see Quantification).
Prioritizing	Commonly Violated Quality Rules	Prioritization of TD items will need to be made based on a method that will be related to the kind of problem, but also on the TD item that suffers.
Repayment	Refactoring Support	Repayment will be performed through code quality improvement techniques. The specific methods for code quality improvement (the number of available approaches is vast) will need to be decided.

[3] The term "inefficiencies" refers to the TD issues identified by SonarQube

2.2 Research on the Implementation of SDK4ED Features

In this section, we present the research approaches that we have implemented in the SDK4ED platform so as to support the features described in Table 1. We note that whenever possible, we preferred to reuse existing methods/tools instead of inventing new ones.

System and Artifact Level Analysis: As part of TD Quantification, we selected to measure the three main pillars of the TD metaphor (Principal, Interest, and Interest Probability) [13]. For *TD Principal quantification* a vast majority of tools exist [14] (e.g., SonarQube, CAST, Squore, etc.); however, recent studies suggested that their results are not in agreement [18][19]. In SDK4ED we have opted to use SonarQube, since: (a) according to Avgeriou et al. [14] is the most commonly used tool in industry and academia; (b) one of the main criticisms that it receives is that it neglects design and architectural problems, but in this study, we focus on code TD; and (c) it is open-source, avoiding the dependence of SDK4ED on closed-source software, limiting its availability. SonarQube quantifies TD Principal by identifying code smells (as the corresponding Quality Model requirements—SQALE method [20]) and calculating their remediation time. For identifying the existence of code smells, SonarQube version 7.9 (for Java) relies on 562 rules (e.g., "*Method overrides should not change contracts*", "*Package declaration should match source file directory*", "*Parameters should be passed in the correct order*", "*Unused labels should be re-moved*").

For quantifying *TD Interest*, we used the FITTED framework, as it has been proposed [1][21][22] and empirically validated [21][23] in our previous work. The validation was performed in an industrial setting and contrasted the scores of TD Interest with the perception of software engineers. The results suggested a rank correlation of 0.73 for TD Interest. We only recap the basic notions of the FITTED framework here and refer to the aforementioned works for further details. Assuming that a system has an *actual* implementation, and a *hypothetical* optimal implementation (in terms of maintainability—i.e., ease to maintain), maintaining the optimal system would require less effort than maintaining the actual system. Despite the fact that a system can by no means be characterized as globally optimal, based solely on the optimization of some structural characteristics, there is a plethora of studies aiming at software optimization, guided by the application of software refactoring [24][25]. Adding a new feature "A" to the optimal system would need a certain effort, noted as $\text{Effort}_{(\text{optimum})}$, whereas adding the same feature to the actual system necessitates a larger effort, noted as $\text{Effort}_{(\text{actual})}$. *The difference between these two efforts represents the TD Interest that is accumulated during this maintenance activity.* According to FITTED [21], maintenance effort is inversely related to the maintainability of the system. Although the relation between effort and maintainability is not necessarily (or by definition) linear, several studies model maintenance effort (through regression modeling) as a polynomial of maintainability indicators [26][27], achieving satisfactory prediction accuracy. Given the aforementioned relation, the maintenance effort for the optimal system (which is unknown), 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 (we call this ratio *Maintainability Level*). Finally, based on its definition, TD Interest can be calculated using the difference between the actual and the optimal effort, as follows. In the final version of the equation, $\text{Effort}_{(\text{actual})}$ is calculated as the average maintenance load (lines of code changed, per version), whereas *Maintainability Level* is calculated based on well-known maintainability metrics (e.g., CC, MPC, DIT, etc.)

$$\boxed{\begin{aligned} TD\ interest &= \Delta Effort = Effort(actual) - Effort(optimum) \\ &= Effort(actual) - Effort(actual) \times (MaintainabilityLevel) \\ &= Effort(actual) \times (1 - MaintainabilityLevel) \end{aligned}}$$

Finally, we quantify *TD Interest Probability* by considering the number of commits in which an artifact has changed and the probability of an artifact to change due to ripple effects.

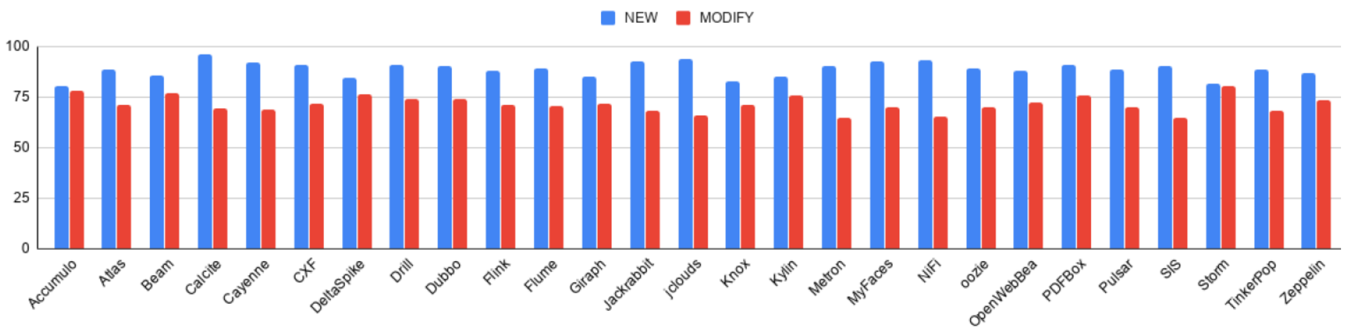


Figure 3. TD reduction caused by new or modified methods

New Code Analysis Control: Through this feature the user is able to inspect the TD Principal that is inserted in every commit, and check if it positively or negatively contributed to system TD. To normalize the contribution of the new code on the TD of the complete system, we use the notion of TD Density contribution [8]. The feature provides quality gate support, in the sense that the developer can first check the TD contribution of the new code, and if the contribution is positive, he/she can commit to the main branch of the project (a form of a quality gate). Through a large-scale case study [8], we have empirically validated that writing clean code is a safer option for reducing TD (see Figure 3).

Evolution Analysis: Regarding evolution analysis, we provide two contributions. The first is related to identifying a general trend in TD accumulation, by inspecting the values of TD Principal and TD Interest along software evolution. This can provide a bird's eye view on if developers consider TD accumulation, as well as pinpoint to timepoints at which TD changes takes place [9]. The second view, relies on FITTED, and in particular the concept of the *breaking point* [22] (see Figure 4). The breaking point is a timestamp in the future evolution of the software, that when surpassed, the accumulated TD Interest becomes higher than the TD Principal. Thus, if the anticipated software evolution is before the breaking point TD repayment is not urgent. For more details, the interested reader can be directed to our previous work [21][22]. Through this feature, we also present the evolution of the breaking point metric (in future versions).

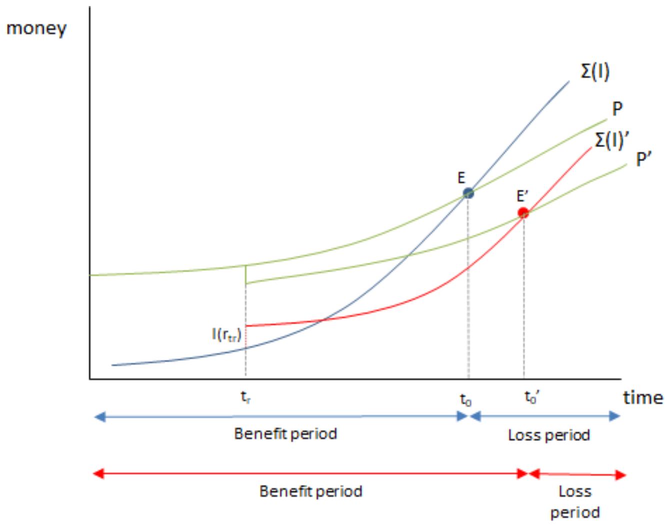


Figure 4. Definition of Breaking Point

Commonly Violated Quality Rules: Through this feature the user will be able to inspect the rules violated in codebase. The reported rule violations are the ones from SonarQube, since we selected to quantify TD Principal, based on this tool. The rules are prioritized by the smell interest probability—the probability that the developer will perform maintenance on at least one artifact that suffers from the corresponding smell in the upcoming revision of the code [5]. This prioritization is expected to guide management and technical stakeholders on which type of smells the company should pay attention to. To calculate smell interest probability, we calculate a joint probability of events. Specifically, as an event we consider the action of maintaining a module that suffers from a specific smell. This event holds a specific probability to occur. The probability that at least one of the modules suffering from the same smell will change (i.e., the interest probability of the smell), is calculated as the joint probability of any maintenance event to occur. The calculation of smell interest probability (vertical axis), contrasted with TDI interest probability (horizontal axis) is presented in Figure 5. For example, for `Smell-2`, we can observe that its occurrence frequency is 3/n, since it appears in three modules (i.e., 2, 3 and n) and the mean change proneness of the modules it appears in is: (`cp2` + `cp3` + `cpn`) / 3. We acknowledge that this approach is not related to the cost of fixing the debt—e.g., prioritize smells or arti-facts based on TD Principal, TD Interest, or TD Interest Probability (as suggested for some literature). However, SDK4ED allows this sorting based on the "Artifact-Level Analysis Feature".

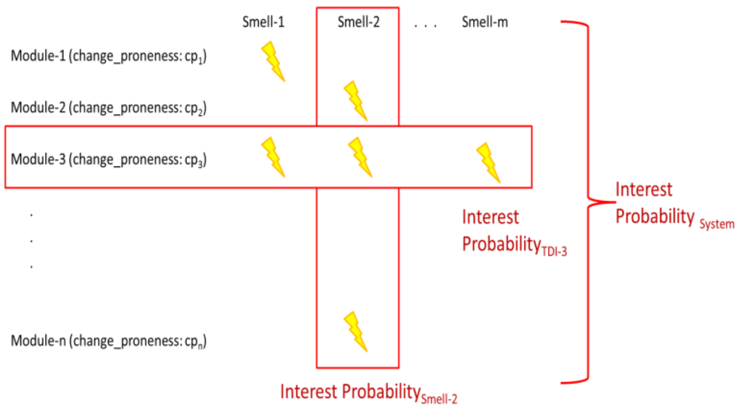
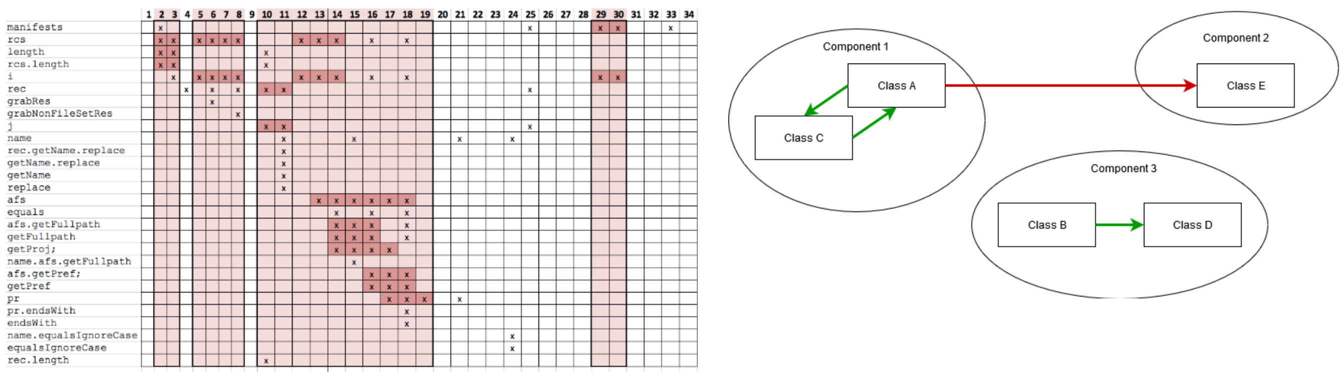

Smell-1
Smell-2
. . .
Smell-m
Module-1 (change_proneness: cp_1)
Module-2 (change_proneness: cp_2)
Module-3 (change_proneness: cp_3)
Module-n (change_proneness: cp_n)
Interest Probability$_{TDI-3}$
Interest Probability $_{System}$
Interest Probability$_{Smell-2}$

Figure 5. Smell Interest Probability

Refactoring Support: Through this feature the user will get a list of identified TD items. At the code level the refactoring support again comes from SonarQube; whereas at the design level the software engineer is guided to apply two object-oriented refactoring (Move Class and Extract Method). The application of refactoring is the most established way for repaying TD, reducing both TD Principal and TD Interest. Both design level refactoring suggestions are driven by the application of the Single Responsibility Principle, i.e., artifacts that are conceptually similar (e.g., classes, instructions) should be placed in the same container (e.g., package, method). The used methods are SEMI for Extract Method refactoring [6] and DeRecGEA [28] for Move Class refactoring.



Cohesive lines of code (since they use at least one common variable) are denoted with pink shading

Green edges denote cohesion, and red ones coupling

Figure 6. Cohesion Definition in SEMI and DeRecGEA

The rationale on how modularity of artifacts is defined in each study is visualized in Figure 6: (a) within the method as coherent we consider two lines of code that access at least one common variable (parameter, attribute, or local variable). Based on this definition, we develop groups of consecutive coherent lines of code that could be extracted; (b) within a component (e.g., a software package) as desired class connection—cohesion (green lines) we consider the links between classes that belong to the same component; whereas as undesired—coupling (red lines) the links between classes that belong to different components. Modularity is a synthesized measure of coupling and cohesion, using a genetic algorithm we propose class movements among packages, so as to optimize modularity. More details can be found in the original studies. We note that SEMI has been validated under a common benchmark, against other extract method refactoring methods and has been ranked as the most accurate one for long methods [6]; whereas DeRecGEA has only been validated in a preliminary manner.

2.3 SDK4ED Platform Walkthrough

In this section we provide a detailed walkthrough of the SDK4ED platform[4]. In particular, we navigate the reader along the front-end of the SDK4ED dashboard. The front-end of the platform communicates with the back-end, allowing the easy invocation of the main functionalities (i.e., web services) that the toolbox provides and the visualization of the produced results. In Figure 7, we present the main dashboard and pin-point to the main menus that are related to TDM: "*Technical Debt*" and "*Refactoring*". The "*Technical Debt*" menu provides two options "*TD Analysis*" and "*TD New Code*" (see Figure 8).

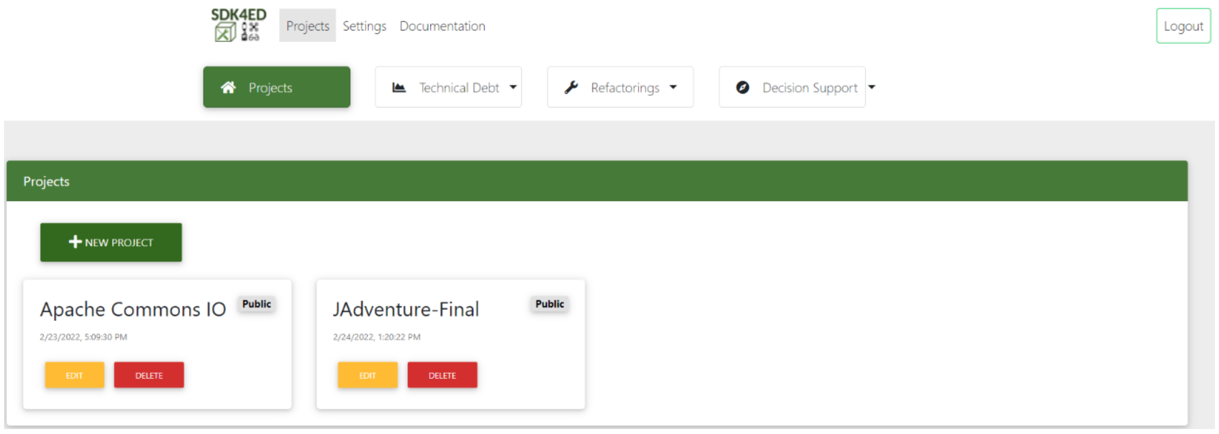


Figure 7. SDK4ED TDM Dashboard

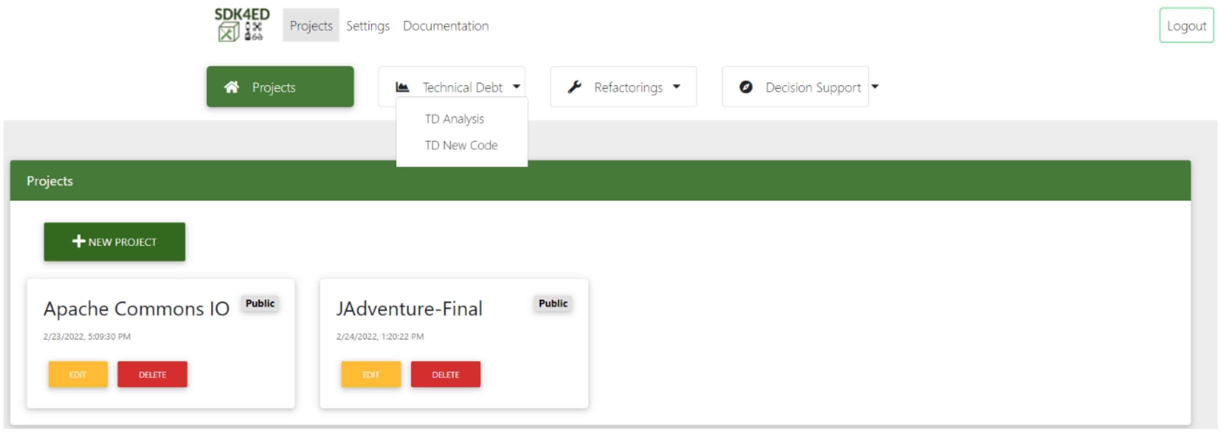


Figure 8. Load Existing Analysis or Create a New Analysis

Supposing that the user selects an existing project (white boxes) and then the "*TD Analysis*" option from the drop-down menu, he/she is navigated to the "*TD Analysis Panel*". The "*TD Analysis Panel*" shows the assessment results of the selected software application that are parsed and presented to the user through different types of visualization. In this panel the user can check the results of the "***Evolu-***

[4] http://sdk4ed-td.se.uom.gr:3000 The platform can be assessed through a free registration (for review purposes: sdk-demo@gmail.com / sdk147)

tion Analysis” (Figure 8), the “***System-Level Analysis***” (Figure 9), as well as the “***Artifact-Level Analysis***” features (Figure 10). If the user wants to start an analysis for a new project, he/she uses the “*New Project*” button in the top of the page, and is guided by the corresponding wizard.

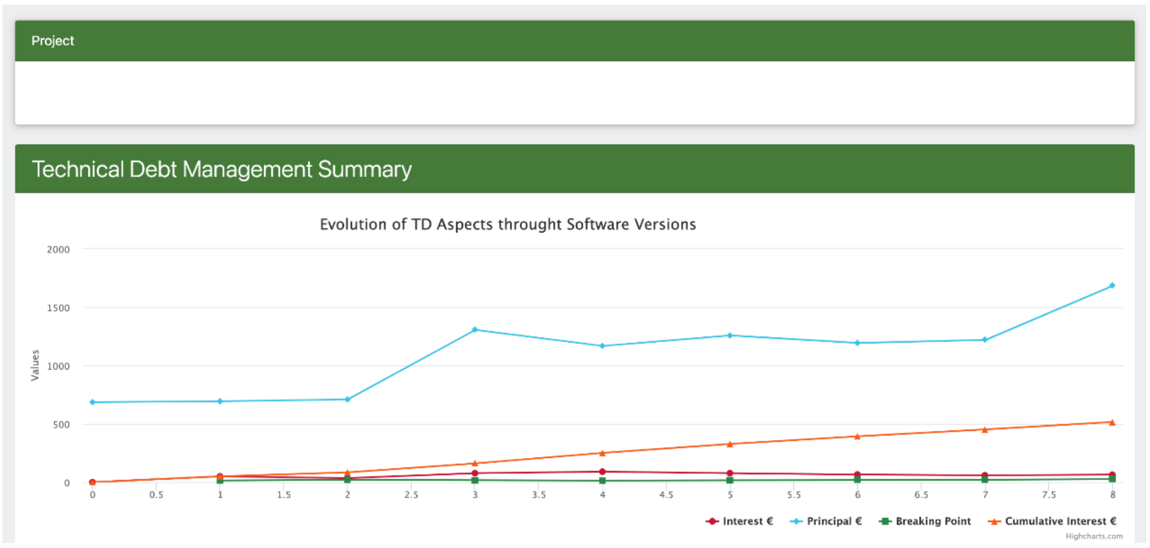
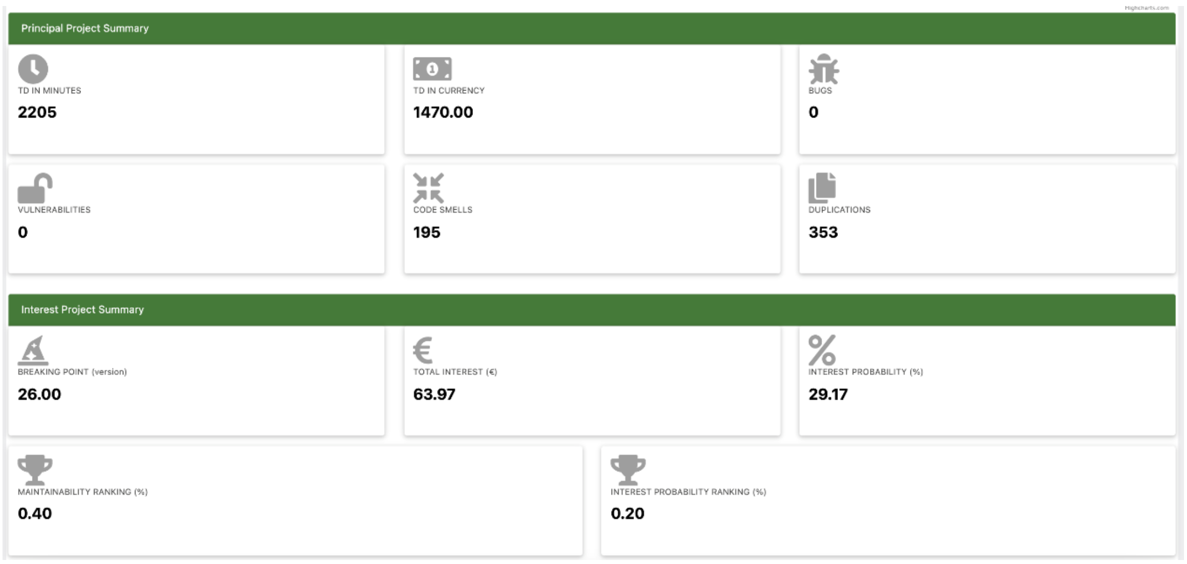


Figure 8. TD Evolution Feature



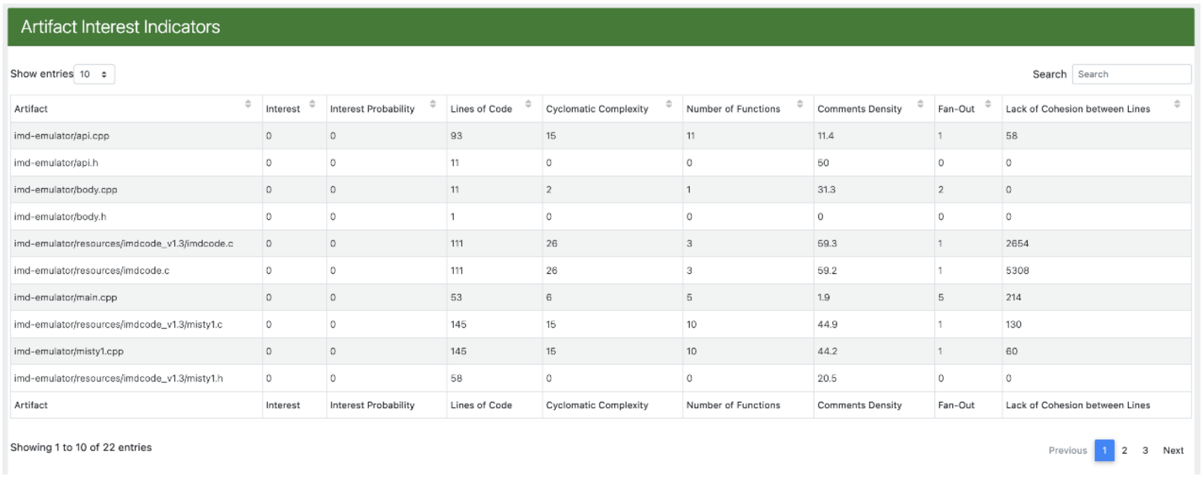
Principal Project Summary

TD IN MINUTES **2205**	TD IN CURRENCY **1470.00**	BUGS **0**
VULNERABILITIES **0**	CODE SMELLS **195**	DUPLICATIONS **353**

Interest Project Summary

BREAKING POINT (version) **26.00**	TOTAL INTEREST (€) **63.97**	INTEREST PROBABILITY (%) **29.17**
MAINTAINABILITY RANKING (%) **0.40**	INTEREST PROBABILITY RANKING (%) **0.20**	

Figure 9. System-Level Analysis Feature



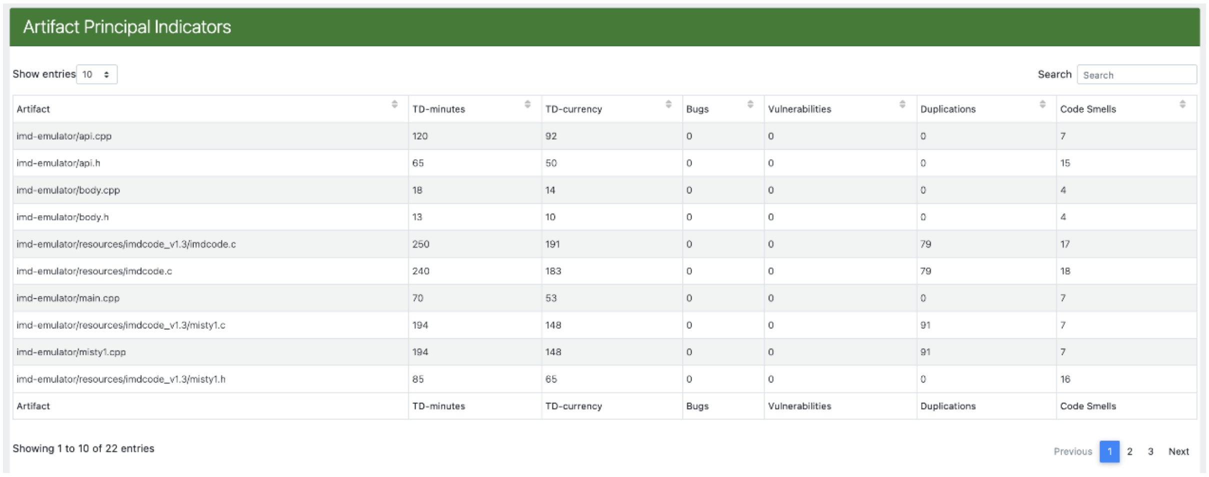
Artifact Interest Indicators

Show entries 10 Search

Artifact	Interest	Interest Probability	Lines of Code	Cyclomatic Complexity	Number of Functions	Comments Density	Fan-Out	Lack of Cohesion between Lines
imd-emulator/api.cpp	0	0	93	15	11	11.4	1	58
imd-emulator/api.h	0	0	11	0	0	50	0	0
imd-emulator/body.cpp	0	0	11	2	1	31.3	2	0
imd-emulator/body.h	0	0	1	0	0	0	0	0
imd-emulator/resources/imdcode_v1.3/imdcode.c	0	0	111	26	3	59.3	1	2654
imd-emulator/resources/imdcode.c	0	0	111	26	3	59.2	1	5308
imd-emulator/main.cpp	0	0	53	6	5	1.9	5	214
imd-emulator/resources/imdcode_v1.3/misty1.c	0	0	145	15	10	44.9	1	130
imd-emulator/misty1.cpp	0	0	145	15	10	44.2	1	60
imd-emulator/resources/imdcode_v1.3/misty1.h	0	0	58	0	0	20.5	0	0
Artifact	Interest	Interest Probability	Lines of Code	Cyclomatic Complexity	Number of Functions	Comments Density	Fan-Out	Lack of Cohesion between Lines

Showing 1 to 10 of 22 entries

Previous 1 2 3 Next



Artifact Principal Indicators

Show entries 10

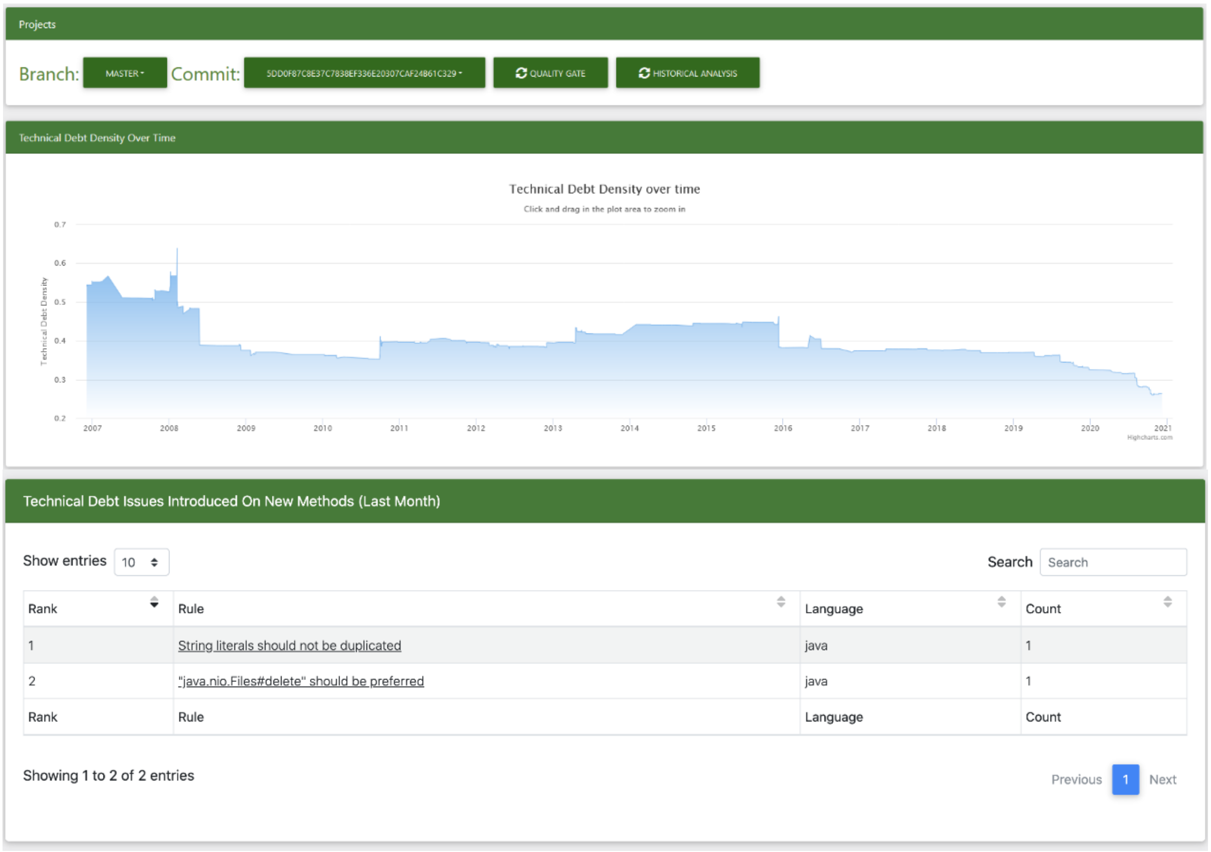
Search

Artifact	TD-minutes	TD-currency	Bugs	Vulnerabilities	Duplications	Code Smells
imd-emulator/api.cpp	120	92	0	0	0	7
imd-emulator/api.h	65	50	0	0	0	15
imd-emulator/body.cpp	18	14	0	0	0	4
imd-emulator/body.h	13	10	0	0	0	4
imd-emulator/resources/imdcode_v1.3/imdcode.c	250	191	0	0	79	17
imd-emulator/resources/imdcode.c	240	183	0	0	79	18
imd-emulator/main.cpp	70	53	0	0	0	7
imd-emulator/resources/imdcode_v1.3/misty1.c	194	148	0	0	91	7
imd-emulator/misty1.cpp	194	148	0	0	91	7
imd-emulator/resources/imdcode_v1.3/misty1.h	85	65	0	0	0	16
Artifact	TD-minutes	TD-currency	Bugs	Vulnerabilities	Duplications	Code Smells

Showing 1 to 10 of 22 entries

Figure 10. Artifact-Level Analysis Feature

If the user selects the "*TD New Code*" option from the drop-down list he/she is navigated to the "*TD New Code*" panel. The "*TD New Code Panel*" corresponds to the "***New Code Analysis Quality Control***" feature (see Figure 11). Through this panel, the user has the option to select a project and perform a historical analysis until the last commit. Moreover, after the historical analysis, the user could select a branch and a specific commit, and then perform a quality gate analysis. Upon the quality gate analysis, a mark (dot) appears in the last commit in the branch. If the dot falls within the light blue area, then the commit is acceptable for the main branch (lower TD density compared to average TD density of the system). If not, the developer must improve the new code that he/she committed, and retry the process. To aid in this process, the panel presented in the lower part of Figure 11 presents the technical debt issues introduced on new methods during the last month.


Projects
Branch:
MASTER
Commit:
5DD0F87C8E37C7838EF336E20307CAF24861C329
QUALITY GATE
HISTORICAL ANALYSIS
Technical Debt Density Over Time
Technical Debt Density over time
Click and drag in the plot area to zoom in
Technical Debt Density
0.7
0.6
0.5
0.4
0.3
0.2
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
Highcharts.com
Technical Debt Issues Introduced On New Methods (Last Month)

Show entries 10

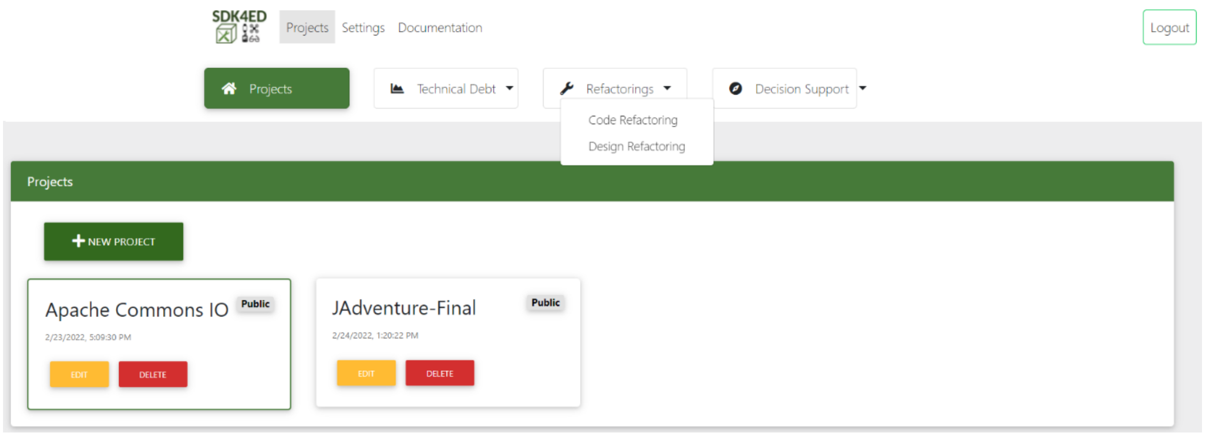
Search

Rank	Rule	Language	Count
1	String literals should not be duplicated	java	1
2	"java.nio.Files#delete" should be preferred	java	1
Rank	Rule	Language	Count

Showing 1 to 2 of 2 entries

Figure 11. New Code Analysis Quality Control Feature

The "*Refactoring*" menu comes with two options (see Figure 12): "*Code Refactoring*" and "*Design Refactoring*". Suppose that the user selects the "Code Refactoring" option from the drop-down menu.

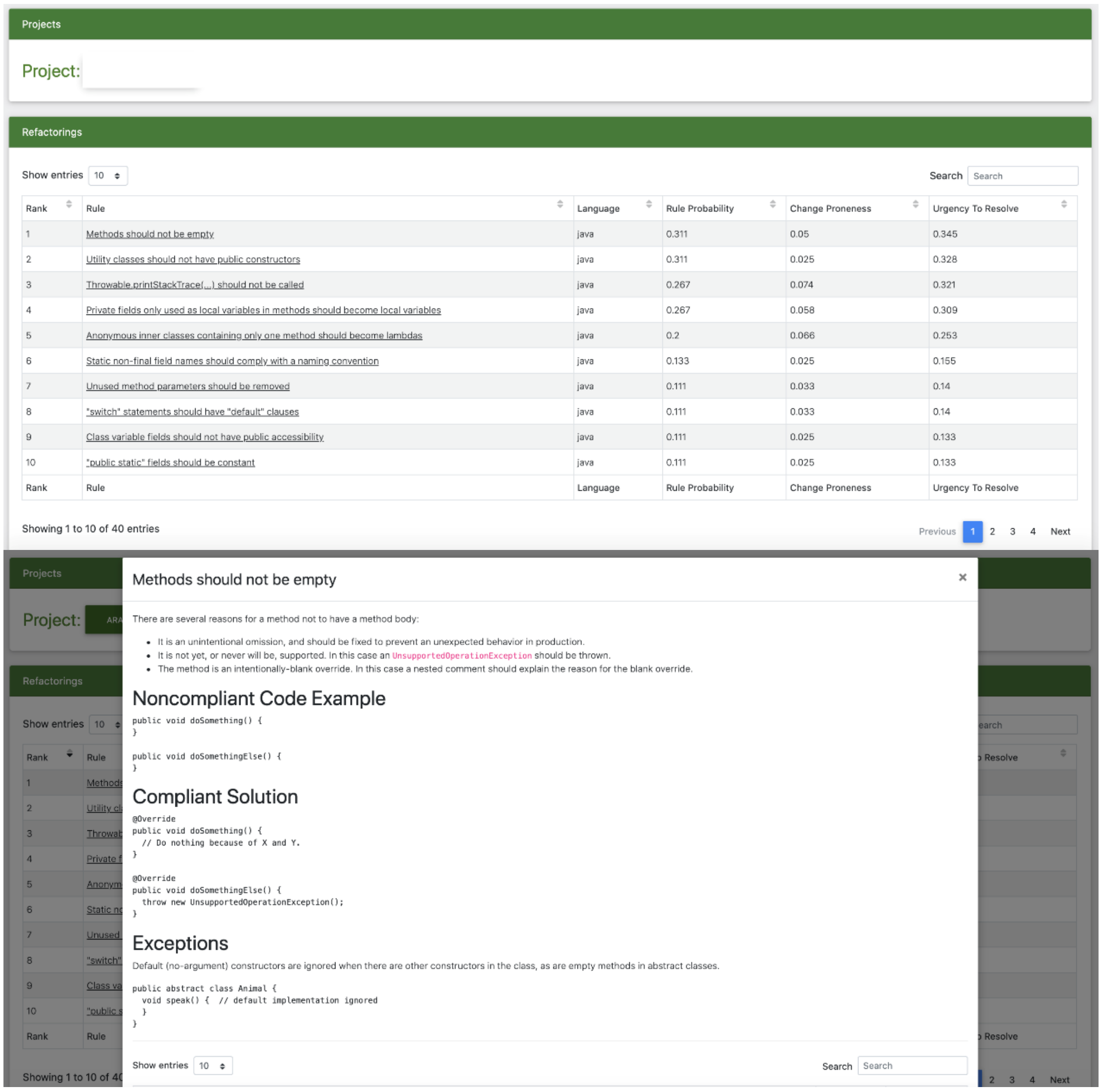


Apache Commons IO Public
2/23/2022, 5:09:30 PM

JAdventure-Final Public
2/24/2022, 1:20:22 PM

Figure 12. Perform TD Reduction Analysis

The user is navigated to the "*Code Refactoring Panel*" (see Figure 13). The resulted (violated) rules are ranked based on the urgency to be solved—smell interest probability. Additionally, we provide the probability of this rule to occur and the probability of artifacts that contain them to change. If the user selects a specific rule, a new window opens with a brief description of the rule and examples of how to solve this code smell (mid-part of Figure 13). Last, in the new window, the user can see the files that violate this rule, the line of code exhibiting the violation and the effort in minutes to solve the corresponding issue (bottom-part of Figure 13). This panel corresponds to the "***Common Quality Rules Violated***" feature.



Rank	Rule	Language	Rule Probability	Change Proneness	Urgency To Resolve
1	Methods should not be empty	java	0.311	0.05	0.345
2	Utility classes should not have public constructors	java	0.311	0.025	0.328
3	Throwable.printStackTrace(...) should not be called	java	0.267	0.074	0.321
4	Private fields only used as local variables in methods should become local variables	java	0.267	0.058	0.309
5	Anonymous inner classes containing only one method should become lambdas	java	0.2	0.066	0.253
6	Static non-final field names should comply with a naming convention	java	0.133	0.025	0.155
7	Unused method parameters should be removed	java	0.111	0.033	0.14
8	"switch" statements should have "default" clauses	java	0.111	0.033	0.14
9	Class variable fields should not have public accessibility	java	0.111	0.025	0.133
10	"public static" fields should be constant	java	0.111	0.025	0.133
Rank	Rule	Language	Rule Probability	Change Proneness	Urgency To Resolve

Showing 1 to 10 of 40 entries

Previous 1 2 3 4 Next

Methods should not be empty

There are several reasons for a method not to have a method body:

- It is an unintentional omission, and should be fixed to prevent an unexpected behavior in production.
- It is not yet, or never will be, supported. In this case an `UnsupportedOperationException` should be thrown.
- The method is an intentionally-blank override. In this case a nested comment should explain the reason for the blank override.

Noncompliant Code Example

```
public void doSomething() {
}

public void doSomethingElse() {
}
```

Compliant Solution

```
@Override
public void doSomething() {
  // Do nothing because of X and Y.
}

@Override
public void doSomethingElse() {
  throw new UnsupportedOperationException();
}
```

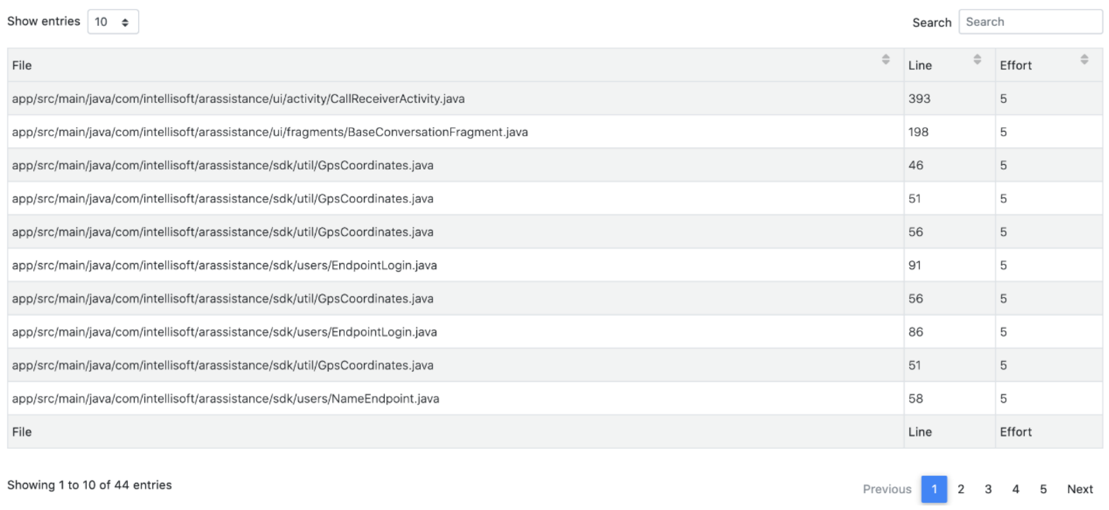
Exceptions

Default (no-argument) constructors are ignored when there are other constructors in the class, as are empty methods in abstract classes.

```
public abstract class Animal {
  void speak() {  // default implementation ignored
  }
}
```

Show entries 10

Search



Show entries 10

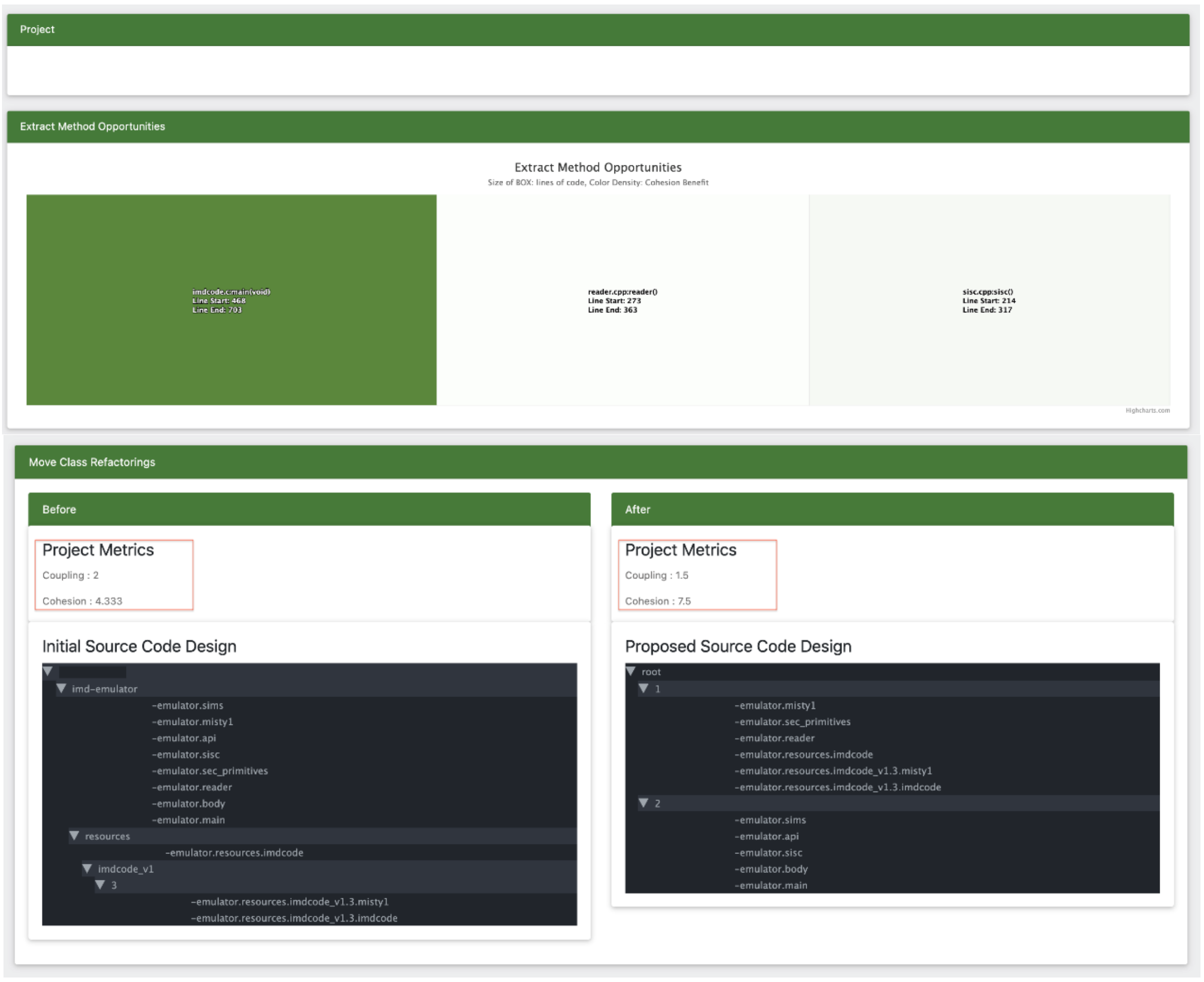
Search

File	Line	Effort
app/src/main/java/com/intellisoft/arassistance/ui/activity/CallReceiverActivity.java	393	5
app/src/main/java/com/intellisoft/arassistance/ui/fragments/BaseConversationFragment.java	198	5
app/src/main/java/com/intellisoft/arassistance/sdk/util/GpsCoordinates.java	46	5
app/src/main/java/com/intellisoft/arassistance/sdk/util/GpsCoordinates.java	51	5
app/src/main/java/com/intellisoft/arassistance/sdk/util/GpsCoordinates.java	56	5
app/src/main/java/com/intellisoft/arassistance/sdk/users/EndpointLogin.java	91	5
app/src/main/java/com/intellisoft/arassistance/sdk/util/GpsCoordinates.java	56	5
app/src/main/java/com/intellisoft/arassistance/sdk/users/EndpointLogin.java	86	5
app/src/main/java/com/intellisoft/arassistance/sdk/util/GpsCoordinates.java	51	5
app/src/main/java/com/intellisoft/arassistance/sdk/users/NameEndpoint.java	58	5
File	Line	Effort

Showing 1 to 10 of 44 entries

Figure 13. Commonly Violated Quality Rules Feature

Finally, if the user selects the "*Design Refactoring*" option from the drop-down list, he/she is navigated to the "*Design Refactoring Panel*" (see Figure 14), corresponding to the "***Refactoring Support Feature***". Through this panel, the user is getting refactoring suggestions. In the top part of Figure 14, we present the way that Extract Method Opportunities are presented: the box size corresponds to the lines of code the method has and the color density to the cohesion benefit from the suggested refactoring. In the bottom part of Figure 14, we present the way Move Class suggestions are presented: i.e., proposing a class / package structure that improves cohesion and coupling at project level.



Project

Extract Method Opportunities

Extract Method Opportunities
Size of BOX: lines of code, Color Density: Cohesion Benefit

imdcode.c:main(void)
Line Start: 468
Line End: 703

reader.cpp:reader()
Line Start: 273
Line End: 363

sisc.cpp:sisc()
Line Start: 214
Line End: 317

Move Class Refactorings

Before

Project Metrics
Coupling : 2
Cohesion : 4.333

Initial Source Code Design

```
▼ [redacted]
  ▼ imd-emulator
              -emulator.sims
              -emulator.misty1
              -emulator.api
              -emulator.sisc
              -emulator.sec_primitives
              -emulator.reader
              -emulator.body
              -emulator.main
    ▼ resources
                -emulator.resources.imdcode
      ▼ imdcode_v1
        ▼ 3
                  -emulator.resources.imdcode_v1.3.misty1
                  -emulator.resources.imdcode_v1.3.imdcode
```

After

Project Metrics
Coupling : 1.5
Cohesion : 7.5

Proposed Source Code Design

```
▼ root
  ▼ 1
              -emulator.misty1
              -emulator.sec_primitives
              -emulator.reader
              -emulator.resources.imdcode
              -emulator.resources.imdcode_v1.3.misty1
              -emulator.resources.imdcode_v1.3.imdcode
  ▼ 2
              -emulator.sims
              -emulator.api
              -emulator.sisc
              -emulator.body
              -emulator.main
```

Figure 14. Refactoring Support Feature

3. Empirical Validation

3.1 Objectives & Research Questions

The main target of the SDK4ED project was the provision of a platform that is ***relevant*** (i.e., applicable and useful) to the software industry, as well as ***usable***, that would ***foster a culture for TDM*** (i.e., increase the investment on maintainability), and ***reduce future maintenance costs***. These targets are the basis for the empirical validation presented in this study. According to the aforementioned goal we have derived four research questions (RQ):

RQ$_1$: Is the SDK4ED platform industrially relevant?

Through this research question, we first explore if the platform is appealing to industrial stakeholders and therefore if they would find it useful to adopt in the future. In this research question we are interested in understanding if some specific features of the platform are more useful than others. We note that the research rigor of SDK4ED approaches has already been evaluated in previous studies—see Section 2.2. The answer to this question can unveil interesting research directions for the TDM community, the Research & Development team of SDK4ED platform, and is interesting to participants in the sense that they can get a hint on which features of TDM are more applicable, based on the perception of their colleagues.

RQ$_2$: To what extent can the use of the SDK4ED platform contribute in the fostering of a TDM culture in software industry?

The motivation for this research question stems from the fact that despite the existence of multiple tools and methods on quality improvement in the last decades, in many cases, in practice quality is an afterthought of managerial and technical stakeholders. To this end, through this research question we aim to explore if the use of the SDK4ED platform can lead to an increase in the investment of software industry on maintainability. We note that the TD metaphor itself is "self-advertised" as a powerful communication medium between technical and managerial stakeholders; therefore, the answer to this research question acts as a useful validation of this claim for the TD community. Additionally, practitioners and especially quality managers can easily identify the features that they need to promote inside their company to build a quality assurance culture and adopt a TDM strategy.

RQ$_3$: To what extent can the use of the SDK4ED platform contribute to reducing future maintenance costs?

The motivation for this research question lies at the heart of the TD metaphor, which suggests that investing on quality *"now"* can save "*future*" costs. To this end, through this research question, we explore the features of the SDK4ED platform that can lead to the largest cost savings. The answer to this research question can be useful for both researchers and practitioners, in the sense that practitioners can identify the features of the SDK4ED platform (and TDM activities in general) that they can employ to achieve cost reduction, whereas researchers can get insights on the effectiveness of various techniques; thus, better focusing their future research endeavors.

RQ$_4$: What is the usability of the SDK4ED platform?

Apart from being relevant and useful in practice, in order for a research prototype to be industry-ready, a key factor is to be usable. Through this research question, we focus on the usability of the platform, assessing its ease of use, learning curve etc. We note that for this research question the evaluation is system-wide, since the usability cannot substantially differentiate among features. The outcome of this research question is of paramount importance to the Research & Development team of the SDK4ED platform for improvement suggestions, as well as the interested practitioners, since it guarantees to some extent the end-users' experience.

3.2 Case Selection and Units of Analysis

This study is an embedded multiple case study, in the sense that it involves multiple units of analysis within the multiple examined cases. As units of analysis, we consider software practitioners that participate in the study; whereas as cases the companies that these practitioners work on. The context of the study is the embedded systems domain, since all cases belong to this application domain. The companies are anonymized due to an NDA, but in Table 2, we list a more specific application domain, the country and the number of the units of analysis that they have contributed to the study setup. In total the study was comprised of 15 units of analysis coming from 4 companies. Additionally, in Figure 15 we provide some basic demographics of the participants

Table 2. Participating Companies Demographics

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

We note that all the involved companies have used the platform for a 30-day period before the evaluation (December 2020). C1 used the platform for managing the TD of a *drone application*, C2 for a *monitoring* system of *IoT bridges* in Sweden, C3 for a *smart glasses* application that is used for *manufacturing cars*, and C4 for a software that controls *heart implants*. Each company faced different maintenance issues: as an example, C4 needed to monitor and manage TD of two different components (the medical and the security component) that evolve at a significantly different pace (the medical part of the app is more stable, but security requirements emerge very often). The decoupling of the two components and the isolated testing was a key-issue for C4.

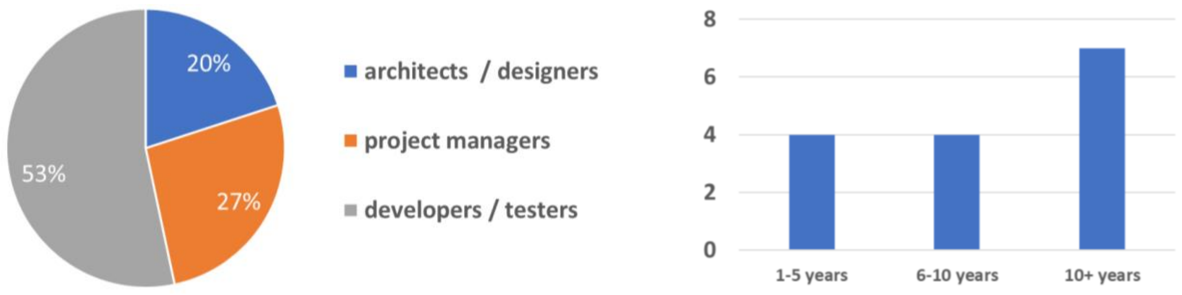


Figure 15. Sample Demographics

3.3 Data Collection

To answer the aforementioned research questions, and given the fact that the SDK4ED platform has been recently released (thereof participants did not have a prior experience with it), we have performed the data collection, upon a 30 days trial. We note that this period could have not been expanded further, due to the limited budget of participating industries for the purpose of this study. During this period, the participants have been acquainted to the platform (through a 1-day workshop from the authors) and then 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. Upon the completion of the trial period, we proceeded to data collection. Data collection was comprised of two methods (2 surveys sessions and 1 focus groups) aiming to achieve method triangulation for all research questions. A mapping between the research questions and data collection methods is presented in Table 3. Below, we discuss in detail each data collection method, and how it was applied for the purpose of our study.

Table 3. Mapping of Data Collection Methods to Research Questions

	Survey-1	**Survey-2**	**Focus Group**
RQ1	X		X
RQ2	X		X
RQ3	X		X
RQ4		X	X

Survey-1: The first survey was aiming to collect data for answering RQ_1, RQ_3, and RQ_4. Each participant was provided with an online questionnaire[5], focusing on the industrial relevance of the SDK4ED platform, its ability to foster a TD culture, and its contribution to cost reductions. To build the survey instrument, we developed a 6-section questionnaire (one for each feature), and 3 questions per section (one for each targeted impact—apart from usability):

Do you consider <<FEATURE>> as relevant (i.e., applicable and useful) for your company?
Do you believe that being aware of the <<FEATURE>> information can boost the investment on maintainability?
Do you believe that based on the information provide by <<FEATURE>> you can save future maintenance cost?

The responses were provided on a 5-point Likert scale. We note that the 3rd question was skipped for the "*Evolution Analysis*" and the "*System-Level Analysis*" features, since they do not provide actionable suggestions to reduce TD, but are only informative.

Survey-2: The second survey was aiming to evaluate the usability of the SDK4ED platform. Similarly to before, we used an online questionnaire[6] to get the responses of the participants. The usability questionnaire was structured based on the System Usability Scale (SUS) [4], which is one of the most well-known instruments for software usability assessment. Therefore, the participants have been given the following statements for which they indicated their level of agreement:

I think that I would like to use this system frequently
I thought this system was too inconsistent
I found the system unnecessarily complex
I felt very confident using the system
I thought the system was easy to use
I found the system very cumbersome to use
I think I would need the support of a technical person to be able to use this system
I would imagine that most people would learn to use this system very quickly
I found the various functions in this system were well integrated
I needed to learn a lot of things before I could get going with this system

The answer to the SUS questionnaire is again in a 5-point Likert scale. The way that the usability score is achieved through this usability instrument is described along the introduction of SUS [4]. Next, we briefly describe the process: The participants' scores for each question (sometimes 5 is best, for others 5 is the worst, based on the nature of the question) are converted to a number, added together and then multiplied by 2.5 to convert the original scores (of 0-40) to a range from 0 to 100. Though the scores are limited to an [0, 100] range, the score is not a percentage and should be considered only in terms of its percentile ranking. Based on the literature, SUS scores higher than 68 are considered above average and anything lower than 68 is below average [4].

[5] https://forms.gle/MpAxizFzo34pMUjE6

[6] https://forms.gle/jvTkyLcZ24EkNYPe9

Focus-Group: As a final means of data collection, we have performed 4 industrial focus groups (one for each company). In particular, during the ***planning*** of the focus group we defined the goals: "*to discuss: (a) applicability / usefulness of the platform; (b) tentative improvements in quality assurance process; (c) tentative cost reduction benefited by the use of the platform; and (d) usability issues*". Regarding the ***design***, each focus group was intended to last for 45' (with each company—3 hours in total)[7]; and was conducted using a teleconferencing platform. The first 3 blocks were intended to last for 12 minutes, whereas the last one for approximately 10 minutes. While ***conducting*** the focus group the discussion was focused on the aforementioned discussion axes, as outlined below. We note that in many cases, when there was an agreement among the participants, we asked the remaining participants to only provide complementary or contradictory claims. With respect to Block-4 (usability assessment) we have asked the participants to consider the most recent task that they have tried to accomplish using the platform and provide their answers, based on that experience. We preferred this strategy compared to providing a specific task, due to the differences in the roles and specific projects of the participants.

Block 1: Industrial Relevance Assessment

Can you explain us the tasks that you have performed over the last period, using the SDK4ED features? Have you found these features useful in practice, and why?

Block 2: Fostering TD Culture Assessment

Do you think that using the SDK4ED features, you have become more aware of TD practices? Do you think that using these following features is now more probable to invest more on maintainability?

Block 3: Cost Reduction Assessment

Can you explain the way that the SDK4ED features can help in the reduction of future maintenance costs?

Block 4: Usability Assessment

How do you perceive the usability of the SDK4ED platform in terms of:

- *Effectiveness (i.e., the accuracy and completeness with which users achieve specified goals)?*
  To what extend did you manage to complete the intended tasks?
  How did you experience the navigability in the tool?
- *Efficiency (i.e., the resources expended compared to the achieve goals)?*
  How much time did you spend per task?
  Was it more or less than the expected time needed based on your experience using the current process?
- *Satisfaction (the comfort and acceptability of use)?*
  How confident did you feel while using the system?
  Would you like to use this system frequently?
  How complex would you characterize the system?
  To what extend would you need the support to be able to use the system?
  How easy would you consider to learn how to use the tool?
  What kind of background would you consider as mandatory before you could get going with the system?
  How would you describe the experience in terms of the reactions of the system to possible stimuli?
- *Have you faced any other usability issues?*

Block 5: Conclusion

Thank you for your time.
Explain next steps (transcribe, analyze, results will be made available in a publication)
Ask if they want to receive the results by mail

We note that the transcriptions of the focus group, as well as the data obtained from questionnaires have not been made available, due to confidentiality reasons, and the signed NDA. The data collection

[7] The focus groups with most of the companies lasted for approximately 1 hour, due to the input that we received from the participants. Especially, regarding Blocks 1-3, the average discussion time was 15-20 minutes, depending on the company.

instrument has been piloted with software engineers, as part of a MSc course on *Advanced Software Engineering*. The group of graduate students involved in the piloting phase was not overlapping with the participants presented in Table 1. Upon piloting, based on the received feedback, as well as our experiences, we have finalized the data collection instrument, as presented above.

3.4 Data Analysis

To validate the SDK4ED platform, we have used quantitative analysis for providing a synthesized overview of the achieved impacts, and qualitative analysis for interpretation of the results. To synthesize qualitative and quantitative findings, we have relied on the guidelines provided by Seaman [12]. On the one hand, to obtain ***quantitative results***, we use the data obtained by the two surveys. To aggregate the ordinal values of the Likert-scale we have summed the scores assigned by all participants to a specific question (as they are expected to be homogeneously spaced). For presentation purposes, we used bar charts to visualize the sum score of responses for all participants. The maximum value in the y-axis for each bar would be 75 points (15 respondents * maximum Likert-scale rating), for all three targets. The closer the bar to the maximum, the higher is the grade that the participants have assigned to the feature, for each evaluation criterion (relevance, cost savings, and TD culture). For usability, we provided the total SUS score, along with the most common scales for interpretation, in terms of acceptance, adjective, and grade.

On the other hand, to obtain the ***qualitative assessments***, we use the focus group data, which we have analyzed based on the Qualitative Content Analysis (QCA) technique [16], which is a research method for the subjective interpretation of the content of text data through the systematic classification process of coding and identifying themes or patterns. This process involved open coding, creating categories, and abstraction. To identify the codes to report, we used the Open-Card Sorting [11] approach. Initially we transcribed the audio file from the focus group and analyzed it along with the notes we kept during its execution. Then a lexical analysis took place: in particular, we have counted word frequency, and then searched for synonyms and removed irrelevant words. Then we coded the dataset, i.e., categorized all pieces of text that were relevant to a particular theme of interest, and we grouped together similar codes, creating higher-level categories. The categories were created during the analysis process by both the first and the third author, and were discussed and grouped together through an iterative process in several meetings of all authors. The reporting is performed by using codes (frequency table) and participants' quotes. Based on Seaman [12] qualitative studies can support quantitative findings by counting the number of cases in which certain keywords occur and then comparing the counts of different keywords, or comparing the set of cases containing the keyword to those that do not. To visualize the mapping among the (feature and codes) pair, we used alluvial diagrams. The alluvial diagram represents links among the values of categorical variables, for which the width of the link represents the frequency with which each mapping appeared in the responses of the participants.

4. Findings / Discussion

In this section, we present the findings of the empirical evaluation of the SDK4ED platformed organized by research question. Along the discussion features are denoted with bold fonts, codes with capital letters, and quotes in italics. In Table 4 we present the codes that have been identified along the discussions of the focus group, accompanied by the most common synonyms, representative quotes and the frequency that participants used them. We note that through this empirical validation procedure we are not able to provide any empirical evidence on comparing SDK4ED to other Technical Debt Management tools. However, we believe that such an exploration would be an interesting future work opportunity for independent researchers.

Table 4. Codes of the Qualitative Analysis

Code	Synonyms	Quote	Frequency
Monetization	Effort Cost Currency	*"From a management perspective this view is very helpful for comparing different products and see where more effort is needed. Monetized assessment helps communication with non-technical roles in the company."*	34
Pinpointing To Code Hotspots	Classes Spots Areas	*"Reviewing the artifact-level analysis regularly along the development process helped me to highlight problem areas and focus the efforts where needed. Hidden debt can be found in time, and sometimes in places no-one thought to look."*	13
Quality Control Through Quality Gates	Commit Push	"*Given the tight schedule for release, having a quality gateway prevents sub-optimization of code and fosters the necessary habits for an economically sound development process in the long term."*	13
Habit Of Quality Control	Routine Daily	"Evolution analysis can be *very useful for the long perspective when managing products, and tuning quality control and training.*"	8
Training	Learn	"*Such a training is a must-have, and would definitely save significant future maintenance costs.*"	7
Experience	Old Used to	"*Refactoring suggestions should ideally comply with the coding models of the firm. Experienced developers may find suggestions beyond current practice annoying. But any suggestions are better than none.*"	4
Deadlines	-	*"It is very useful to have an evolution analysis. Increase or reduction of investment in quality control depends on project budget and could affect the deadlines."*	4
Budget	-		3
IDE Integration	-	"*It provides good quality advice with almost no extra effort, and also helps in training entry-level programmers for good coding habits. We already use this through our IDE, maybe SDK4ED could also be integrated there.*"	3
Doubts	Not sure	"*Method extraction support is useful, but I'm not so sure about class moving suggestions. Classes are grouped in packages by some logic and may be hard for a software to suggest a better grouping logic.*"	2

4.1 Industrial Relevance

In this section, we discuss the findings regarding RQ_1, on the industrial relevance of the SDK4ED features. In the left part of Figure 16, we present the results of the quantitative analysis through bar charts (100% corresponds to the maximum 75 points of relevance that each feature could get from the survey). The right side of the Figure corresponds to the outcomes of the qualitative analysis: we can explore which codes have been mapped to each feature, with respect to relevance. For instance, MONETIZATION was mentioned 20 times by the participants during the focus group. By following the links (from monetization to the left part of the diagram), we can deduce that it is mentioned while discussing the ***Evolution Analysis*** and ***System-Level Analysis*** (11 times in the former and 9 in the latter). Reading the chart in the opposite direction, by following the red link, we can deduce that while discussing REFACTORING (9 times), the participants mentioned DOUBT (2 times), EXPERIENCE (4 times), and IDE INTEGRATION (3 times).

The results suggest that the main benefit of **Evolution Analysis** and **System-Level Analysis** is the provision of MONETIZED assessments, whereas the most useful outcome of **Artifact-Level Analysis** (top ranked in this evaluation axis) is PINPOINTING TO CODE HOTSPOTS: “*Having an overview of low-quality artifacts is good for keeping an eye on overall quality and taking more care on further development on those artifacts*”. Regarding **New Code Analysis** (top ranked in this evaluation axis), as a main benefit the participants have highlighted QUALITY CONTROL THROUGH QUALITY GATES, as vividly explained by a participant: “*It is very important to check the quality of code before commits. Sometimes, because of deadlines it is not possible, but would be great to do*”. Regarding **Rules Violation Analysis**, the participants highlighted its value for TRAINING: “*We employ mostly self-directed training, this feature is certainly a good tool to show the direction and help in selecting where to put the most effort*”. Finally, with respect to **Refactoring Support** (least ranked in this evaluation axis) the participants were more skeptical, suggesting that such hints must be INTEGRATED WITH AN IDE, that EXPERIENCE plays an important role in their usefulness, and raised DOUBTS on the showcased refactoring: “*Method extraction refactoring support is useful. I'm not sure about class moving*” and “*Refactoring suggestions should comply with the coding principles of the firm; experienced developers may find suggestions beyond them less useful or annoying*”. Given the aforementioned views, we can deduce that practitioners consider TD prevention as more relevant to TD repayment, and that any TD repayment should be focused specifically on hot-spots and not be blindly guided by tools that parse the system holistically, making suggestions.

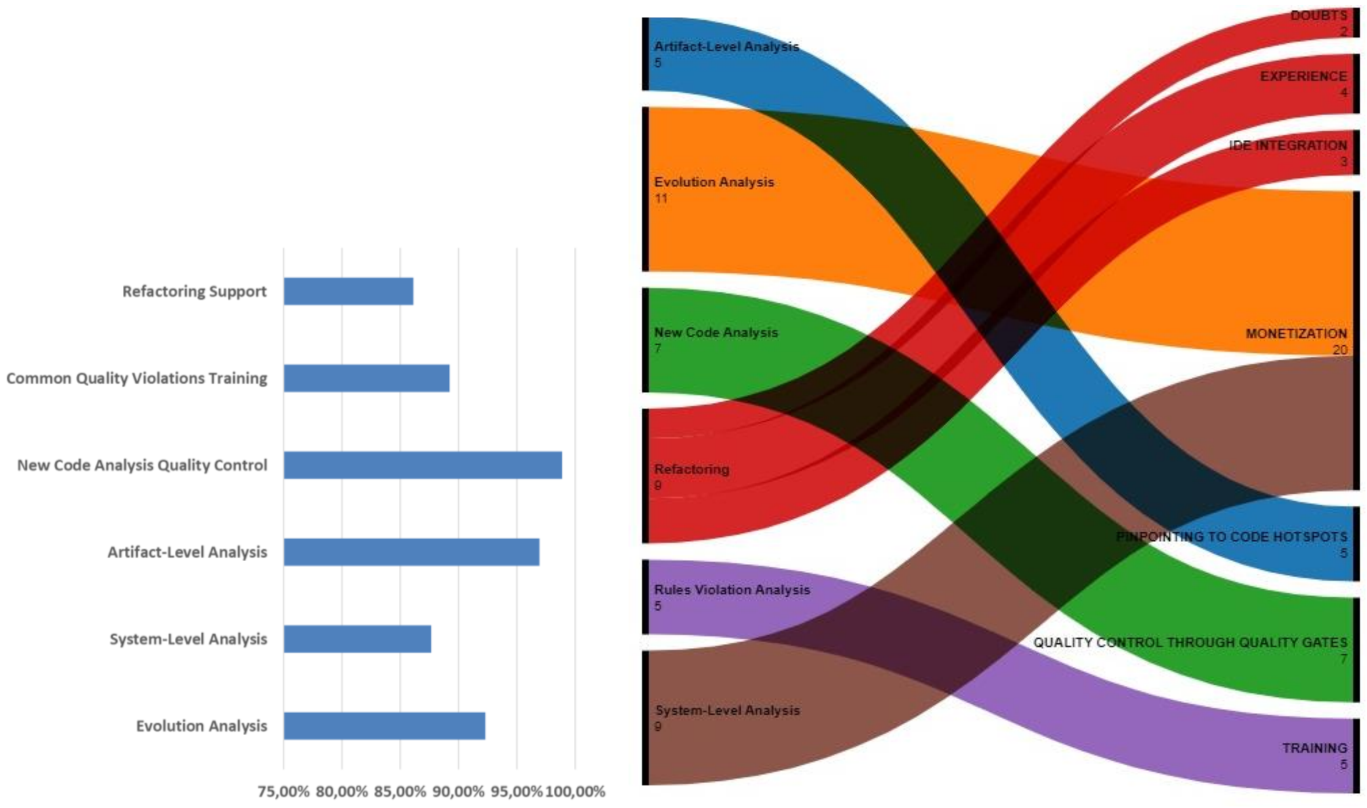


Figure 16. Industrial Relevance

New Code Quality Control, Evolution, and Artifact-Level Analysis are the most relevant features of the SDK4ED platform. The monetization that the TD metaphor offers is promoted to the key element contributing to the industrial relevance of the presented results.

4.2 Increase Investment on Maintainability

The results of our study on RQ_2 (increase TD-awareness and investment on maintainability) are visualized in Figure 17, following the same structure and format from Section 4.1. From the qualitative findings, we can claim that the **New Code Analysis** is the one that participants characterized as the feature that could increase the investment in maintainability, followed by **Artifact-Level Analysis** and **Evolution Analysis**.

A key finding upon the focus group is that after the use of the platform, one software industry has indeed taken action in this direction and hired a dedicated quality manager for inspecting the TD at a daily basis. The main vehicle for achieving the cultivation of a TDM culture is the MONETIZED nature of TD: "*From a management perspective this view is very helpful for comparing different products and see where more effort is needed. Monetized assessment helps communication with non-technical roles in the company*". Additionally, the HABBIT OF QUALITY CONTROL has been emphasized: "*It provides good quality advice with almost no extra effort, and helps in training entry-level programmers for good coding habits*". However, other participants noted that such a platform might not necessarily lead to an increase in investment in sight of inefficient BUDGET or close DEADLINES. Regarding management activities, TD prevention and TD monitoring have proven to be of paramount importance as drivers for increasing investment on maintainability. In particular, it seems that practitioners are more willing to invest in code quality if they see that the code quality decays along evolution and that the new code is of worse quality than the existing codebase. This can be interpreted by the fact that budget-related decisions are usually taken by higher management that are more probable to inspect system-level trends rather than artifact-specific information.

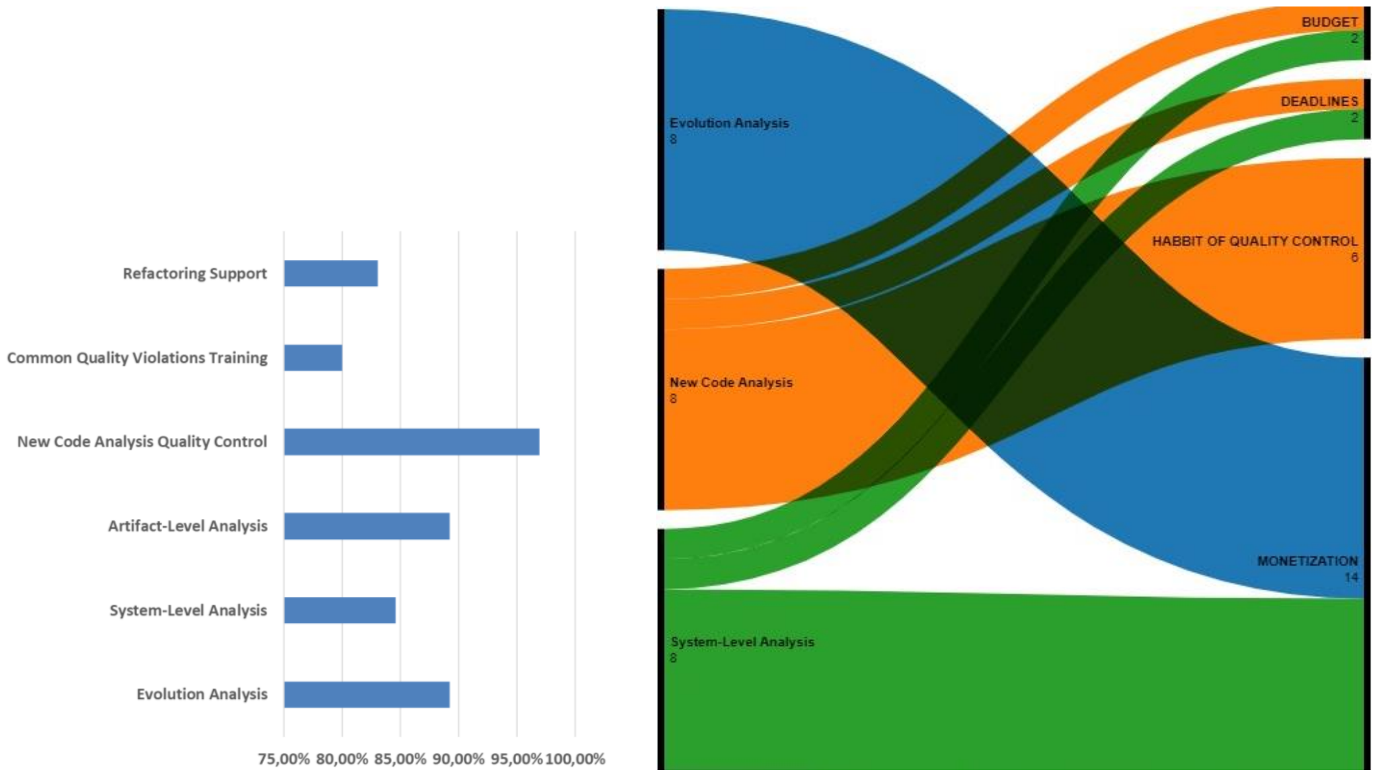


Figure 17. Investment on Maintainability

New Code Quality Control is the feature that yields the largest investment on maintainability. The key concept of TD, i.e., the monetization of the tentative financial loss because of TD, is the main driver for raising the awareness of industry to TDM. More than 50% of the identified reasons for increasing the investment on maintainability are related to monetization. System-Level and Evolution analysis contribute the most in visualizing monetization of TD.

4.3 Cost Saving

The results on RQ_3 are visualized in Figure 17. From the bar chart, we can observe that the **New Code Analysis** is the one that achieves the highest rank with respect to this criterion. The Artifact-Level Analysis has been positively assessed, since it PINPOINTS TO CODE HOTSPOTS that are more probable to produce high maintenance costs: "*Hidden debt can be found in time, and sometimes in places no-one thought to look*". The New Code Analysis was deemed as an interesting way to reduce future maintenance costs by QUALITY CONTROL THROUGH QUALITY GATES: "*Code analysis before committing can ensure that the committed codebase is kept at a high quality state, which means lower maintenance costs*", respecting BUDGET *and* DEADLINES: "*Given the often tight schedule for release, having a quality gateway prevents sub-optimization and fosters the necessary habits for an economically sound development process in the long term*". Furthermore, the TRAINING on commonly violated rules (Rule Violation Analysis) has been characterized as a must-have; whereas the relation of Refactoring Support and cost saving is well-known to participants: "*Support in terms of method extraction, class moving and coding practice are very useful and can save costs for future maintenance*".

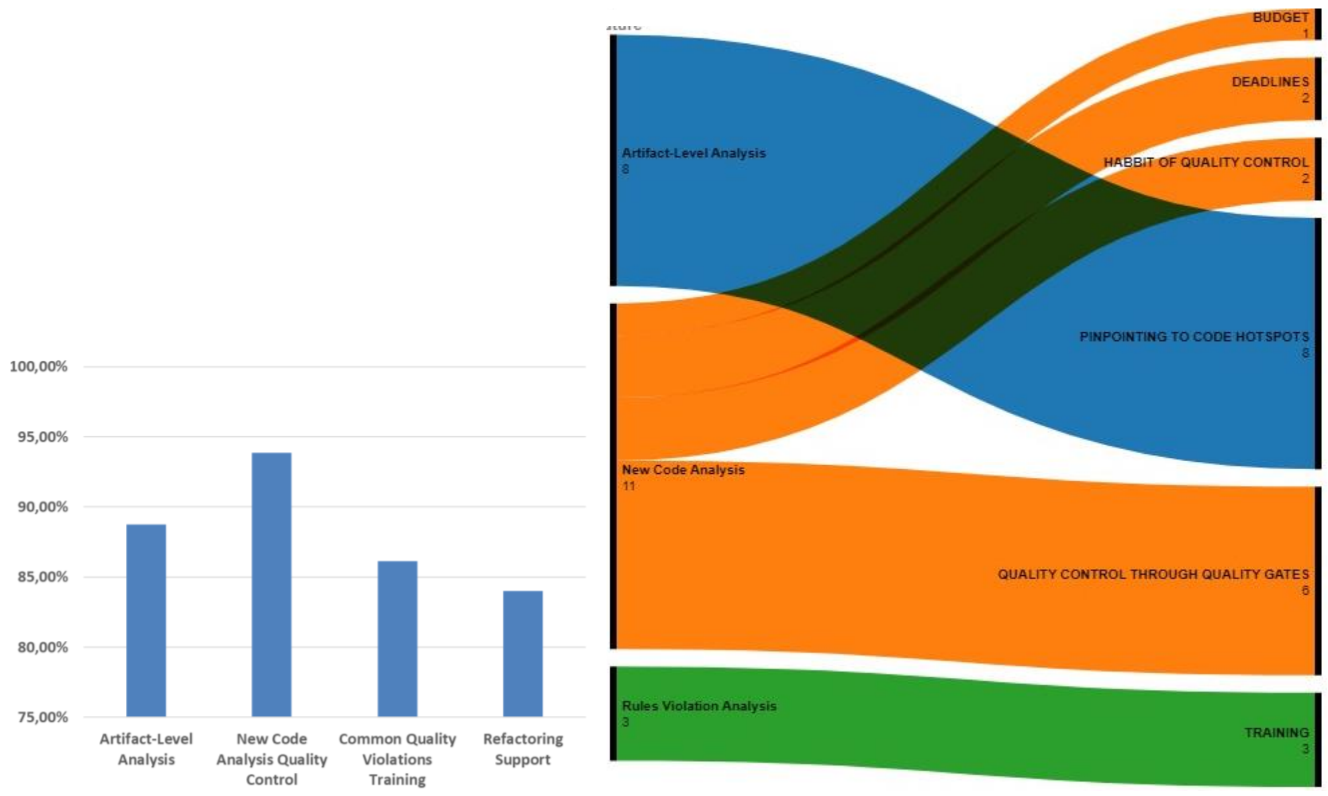


Figure 18. Cost Savings

TD prevention (in particular New Code Analysis) seems as the most promising way to reduce software costs, compared to training and refactoring. The rules violation analysis has been considered as important only for cost saving purposes.

4.4 Usability

The usability of the SDK4ED platform has been positively evaluated, with an average grade B (76.8%), ranging from C (min: 64) to A (max: 85)—see Figure 19. The frequency of C grades was 20%, B grades were the prominent ones (54%), and 26% of the participants evaluated the platform as A-class. Based on the feedback PROJECT CREATION, ERROR HANDLING, and LOADING PERFORMANCE has been registered as open issues for the development team to deal with in the upcoming months of development and maintenance of the SDK4ED platform.

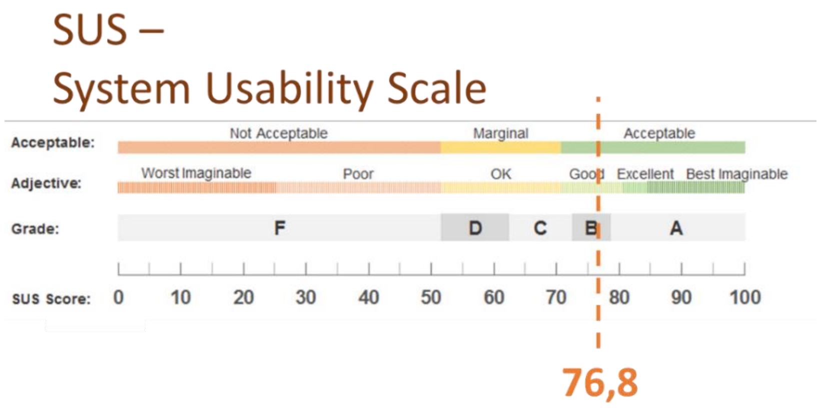


Figure 19. SDK4ED Platform Usability Assessment

The SDK4ED platform has received a positive evaluation in terms of usability, constituting it acceptable for industrial usage.

5. Limitations of the Platform / Threats to Validity

In this section, we discuss the limitations of the platform in its current status, as well as threats to the validity of the empirical component of this study. First, regarding the platform we acknowledge that not all types of TD are covered. In the literature, more than 10 types of TD have been identified; however, SDK4ED manages only code debt, as well as other types of debt as reflected in the source code of the applications (e.g., misinformed architectural decisions form Architectural TD, but it probably leads to poor quality implementation that is captured by SDK4ED platform). Additionally, we cannot claim for a ground truth calculation of TD Principal and TD Interest. As widely accepted in the community, TD Principal and Interest quantifications are wicked problems; since there is no state-of-the-practice tool for TD Principal, and almost no tool for accurately calculating TD Interest [14]. The reasons that quantification endeavors are not at the level of maturity to be considered as state-of-the-art, are: (a) the lack of a baseline on what TD is and what is not; (b) the uncertainty on the frequency and load of software maintenance; and (c) the lack of hard evidence on refactoring and maintenance costs. Based on the above, we admit that the absolute numbers in the platform might not correspond to the actual costs. Thus, we evaluate the ability of such numbers to help practitioners in identifying spots to perform refactoring, boost their focus on quality improvements, etc., refraining from validating the actual numbers. A validation on the accuracy of TD tools and their agreement is the focus of other studies (e.g., [15]). Additionally, the proposed refactoring opportunities correspond to a small portion of the available ones; therefore, TD repayment suggestions are not comprehensive.

With respect to the empirical component of this work, as threats to the validity of the study, we can refer to two main aspects on the validity of qualitative studies: reliability and representativeness. Usually, these two aspects are competing: high reliability demands the application of various data extraction methods to achieve triangulation; whereas representativeness requires the examination of various cases. In this study, we believe that we were able to achieve a good balance, since we included 15 participants from 4 software industries, of different domains (i.e., a quite broad coverage) and at the same time achieved method triangulation by performing focus groups and collecting questionnaires. The expansion of the study to additional cases was not possible, since before the workshop a 30-days trial of the platform was necessary. Also, the expansion of the period to more than 30-days was not possible, due to lack of resources allocated for this study in the 4 industries. However, we need to acknowledge that despite our effort to achieve balance between reliability and representativeness, a replication with more industries would substantially benefit the generalizability of the results.

6. Conclusions

This study presents the SDK4ED platform that aims at supporting TDM. The platform has been empirically validated, with respect to its industrial relevance, usability, and contribution towards increasing the investment in maintainability and cost reduction. To achieve this goal the platform has been placed in use from 4 industries, out of which 3 continued to use the platform beyond the trial period. It is notable that after using the platform for 1 month, one company (C3) hired a dedicated person for using SDK4ED for managing TD; providing further evidence on the ability to generate a TD culture through tool support and corresponding training. Overall, the platform has been positively evaluated in all aspects, especially with respect to the *monetized assessment* it provides (rendering the platform relevant to industrial needs and fostering investments on maintenance), cultivating a *habit of quality* control (thereby leading to cost savings and investments on maintenance) and *pinpointing to design hotspots* (also found relevant and valuable for reducing costs). In terms of specific features, the results suggest that assisting the writing of clean new code is a very welcomed method by practitioners to manage the TD of their code-bases. Nevertheless, all the above are subject to the mainstream challenges of software development of inefficient budget and strict deadlines. Regarding TDM activities, the results have suggested that TD Prevention is an appealing way to reduce TD, and therefore research teams should focus more on this aspect.

As possible follow-ups on this study we intent to perform artifact analysis within the 4 industries, after they use SDK4ED platform over a longer period, without any supervision, to record in practice the targeted benefits (e.g., actual cost savings). With respect to the next steps on evolving the platform, 5 out of 6 features are final, whereas refactoring so far supports only two object-oriented refactoring opportunities. In the near future, we plan to include in the platform additional refactoring identification approaches. Furthermore, regarding monitoring, we have already incorporated forecasting capabilities [29]; and for TD identification we have introduced a machine learning approach [30] that is able to identify the artifacts with high-levels of TD, using the intersection of three well-known tools (Sonar, CAST, and Squore) [19].

References

[1] Ar. Ampatzoglou, Ap. Ampatzoglou, P. Avgeriou, and A. Chatzigeorgiou, “Establishing a framework for managing interest in technical debt”, 5th International Symposium on Business Modeling and Software Design (BMSD 2015), Italy, 2015.

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

[3] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, “Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162),” Dagstuhl Reports, vol. 6, no. 4, pp. 110–138, 2016.

[4] J. Brooke, J. “System Usability Scale (SUS): A quick-and-dirty method of system evaluation user information”, In: P. W. Jordan, B. Thomas, B. A. Weerdmeester, & I. McClelland (Eds.), Usability evaluation in industry (pp. 189-194), Taylor & Francis, 1996.

[5] S. Charalampidou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, “Assessing Code Smell Interest Probability: A Case Study”, 9th International Workshop on Managing Technical Debt (MTD’ 17), ACM, Cologne, Germany, 22 May 2017.

[6] S. Charalampidou, A. Ampatzoglou, A. Chatzigeorgiou, A. Gkortzis, and P. Avgeriou, “Identifying Extract Method Refactoring Opportunities based on Functional Relevance”, Transactions on Software Engineering, IEEE Computer Society, 2017.

[7] J. Eisenberg “A threshold-based approach to technical debt”, ACM SIGSOFT Software Engineering Notes, 37 (2), pp. 1 - 6, ACM, 2012.

[8] G. Digkas, A. Chatzigeorgiou, A. Ampatzoglou, and P. Avgeriou, “Can Clean New Code Reduce Technical Debt Density?”, Transactions on Software Engineering, IEEE Computer Society, 2022.

[9] G. Digkas, M. Lungu, A. Chatzigeorgiou, A. Ampatzoglou and P. Avgeriou, “How Do Developers Pay Back Technical Debt in the Apache Ecosystem?”, 25th International Conference on Software Analysis, Evolution and Reengineering (SANER’ 18), IEEE, Campobasso, Italy, 20-23 March 2018.

[10] Z. Li, P. Avgeriou, and P. Liang, “A systematic mapping study on technical debt and its management”, Journal of Systems and Software, Elsevier, v. 101, pp. 193-220, March 2015.

[11] D. Spencer, “Card Sorting: Designing Usable Categories”, Rosenfeld Media, 1st edition, April 2009.

[12] C. Seaman, “Qualitative Methods in Empirical Studies of Software Engineering”, *IEEE Transactions on Software Engineering,* 25 (4), pp. 557–572, 1999

[13] N. Zazworka, A. Vetró, C. Izurieta, S. Wong, Y.Cai, C. Seaman and F. Shull, “Comparing four approaches for technical debt identification", Software Quality Journal, Springer, 22 (3), pp. 403 – 426, Sept. 2014.

[14] P. Avgeriou, D. Taibi, A. Ampatzoglou, F. Arcelli Fontana, T. Besker, A. Chatzigeorgiou, V. Lenarduzzi, A. Martini, A. Moschou I. Pigazzini, N. Saarimaki, D. D. Sas, S. S, de Toledo, A. A. Tsintzira, "An Overview and Comparison of Technical Debt Measurement Tools," in IEEE Software, vol. 38, no. 3, pp. 61-71, May-June 2021.

[15] M. T. Baldassarre, V. Lenarduzzi, S. Romano, N. Saarimäki, “On the diffuseness of technical debt items and accuracy of remediation time when using SonarQube”, Information and Software Technology, vol. 128, 2020.

[16] S. Elo and H. Kyngäs, “The qualitative content analysis process”, Journal of Advanced Nursing, vol. 62, issue 1, pp. 107-115, 2008

[17] H. van Vliet, “Software Engineering: Principles and Practice”, John Wiley (2008)

[18] J. Lefever, Y. Cai, H. Cervantes, R. Kazman, and H. Fang, “On the Lack of Consensus Among Technical Debt Detection Tools”, International Conference on Software Engineering (SEIP), pp. 121-130, 2021.

[19] T. Amanatidis, N. Mittas, A. Moschou, A. Chatzigeorgiou, A. Ampatzoglou, and L. Angelis, “Evaluating the agreement among technical debt measurement tools: building an empirical benchmark of technical debt liabilities”, Empirical Software Engineering, vol. 25, issue 5, pp. 4161-4204, 2020.

[20] J. L. Letouzey and M. Ilkiewicz, “Managing Technical Debt with the SQALE Method”, Software, IEEE 29 (6), pp. 44–51, 2012.

[21] Ar. Ampatzoglou, A. Michailidis, C. Sarikyriakidis, Ap. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, “A framework for managing interest in technical debt: an industrial validation”, Proceedings of the 2018 International Conference on Tech-nical Debt (TechDebt 2018), ACM, Gothenburg, Sweeden, pp. 115-124, May 2018

[22] A. Chatzigeorgiou, Ap. Ampatzoglou, Ar. Ampatzoglou, and T. Amanatidis, “Estimating the breaking point for technical debt”, 7th International Workshop on Managing Technical Debt (MTD’ 15), IEEE, Bremen, Germany, pp.53-56, Oct. 2015

[23] A. A. Tsintzira, Ar. Ampatzoglou, O. Matei, Ap. Ampatzoglou, A. Chatzigeorgiou, and R. Heb, “Technical Debt Quantification through Metrics: An Industrial Validation”, 15th China-Europe International Symposium on Software Engineering Education (CEISEE’ 19), IEEE TEMS, Lisbon-Caparica, Portugal, May 2019

[24] M. O’ Keeffe and M. O. Cinnéide, “Search-based refactoring for software maintenance”, Journal of Systems and Software 81, no. 4 (2008): 502-516

[25] A. Ouni, M. Kessentini, H. Sahraoui, K. Inoue, and K. Deb, “Multi-criteria code refactoring using search-based software engineering: An industrial case study’, ACM Transactions on Software Engineering and Methodology (TOSEM) 25(3), pp. 1-53, 2016.

[26] C. van Koten and A. Gray, “An application of Bayesian network for predicting object-oriented software maintainability” Information and Software Technology, Elsevier, 48 (1), pp. 59-67, 2006

[27] Y. Zhou and H. Leung, “Predicting Object-Oriented Software Maintainability using Multivariate Adaptive Regression Splines”, Journal of Systems and Software, Elsevier, 80 (8), pp. 1349-1361, 2007

[28] T. Maikantis, A. A. Tsintzira, A. Ampatzoglou, E. M. Arvanitou, A. Chatzigeorgiou, I. Stamelos, S. Bibi, and I. Deligiannis, “Software Architecture Reconstruction via a Genetic Algorithm: Applying the Move Class Refactoring”, 24th Pan-Hellenic Conference on Informatics (PCI ‘20), ACM, 20 – 22 November 2020, Greece.

[29] D. Tsoukalas, D. Kehagias, M. Siavvas, and A. Chatzigeorgiou, “Technical debt forecasting: An empirical study on open-source repositories”, Journal of Systems and Software, Elsevier, 170, 2020.

[30] D. Tsoukalas, N. Mittas, A. Chatzigeorgiou, D. Kechagias, A. Ampatzoglou, T. Amanatidis, and L. Angelis, "Machine Learning for Technical Debt Identification", *IEEE Transactions on Software Engineering*, 2023.