Evaluating the Agreement among Technical Debt Measurement Tools: Building an Empirical Benchmark of Technical Debt Liabilities

Theodoros Amanatidis[1], Nikolaos Mittas[2], Athanasia Moschou[1], Alexander Chatzigeorgiou[1], Apostolos Ampatzoglou[1], and Lefteris Angelis[3]

[1] Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece

[2] Department of Chemistry, International Hellenic University, Kavala, Greece

[3] Department of Informatics, Aristotle University of Thessaloniki, Greece

tamanatidis@uom.edu.gr, nmittas@chem.ihu.gr, nasiamoschou@gmail.com, achat@uom.gr, a.ampatzoglou@uom.edu.gr, lef@csd.auth.gr

Software teams are often asked to deliver new features within strict deadlines leading developers to deliberately or inadvertently serve "*not quite right code*" compromising software quality and maintainability. This non-ideal state of software is efficiently captured by the Technical Debt (TD) metaphor, which reflects the additional effort that has to be spent to maintain software. Although several tools are available for assessing TD, each tool essentially checks software against a particular ruleset. The use of different rulesets can often be beneficial as it leads to the identification of a wider set of problems; however, for the common usage scenario where developers or researchers rely on a single tool, diverse estimates of TD and the identification of different mitigation actions limits the credibility and applicability of the findings. The objective of this study is two-fold: First, we evaluate the degree of agreement among leading TD assessment tools. Second, we propose a framework to capture the diversity of the examined tools with the aim of identifying few "*reference assessments*" (or class/file profiles) representing characteristic cases of classes/files with respect to their level of TD. By extracting sets of classes/files exhibiting similarity to a selected profile (e.g., that of high TD levels in all employed tools) we establish a basis that can be used either for prioritization of maintenance activities or for training more sophisticated TD identification techniques. The proposed framework is illustrated through a case study on fifty (50) open source projects and two programming languages (Java and JavaScript) employing three leading TD tools.

Keywords: technical debt, software quality, archetypal analysis, inter-rater agreement, empirical benchmark

1. Introduction

Throughout the software lifecycle, practitioners speed up the development process by compromising software quality and maintainability in favor of shorter time-to-market. This compromise has been effectively captured by the concept of ***Technical Debt*** (TD), as coined by Ward Cunningham (Cunningham, 1992), offering an analogy to the financial debt. In financial debt, one party borrows capital from another party and repays it back with some added interest. In the TD metaphor, the development team 'borrows' a certain amount of effort by delivering non-ideal code and repays it gradually in future iterations in the form of additional time and effort to perform maintenance on the non-ideal code. The increased maintenance effort, which is caused by the degradation of software maintainability, is considered as the "interest" that the development team has to pay in the long term. In contrast to financial debt, TD is hard or even impossible to measure accurately. The suggested practice, according to the OMG specification on Automated Technical Debt Measure (ATDM)[1], is to consider as ***principal*** of TD (at the source code level) the total effort required to eliminate TD items, which are inefficiencies that have been identified in a software artifact under an established ruleset. However, even if developers are aware of parts of the code that "*do not feel right*" it is challenging to associate an exact numerical estimate with every rule violation. Software modules evolve over time and subtle or major changes in their TD might be incurred by the transition from one commit to the next, rendering the accurate monitoring of TD even more demanding.

The limitations on accurately measuring TD lead to various shortcomings in both academia and industry, in the sense that one cannot control (or manage) what he/she cannot measure (DeMarco, 1986). Despite the fact that several tools are available for measuring and monitoring TD (notable examples include CAST AIP[2], Squore[3],

[1] https://www.omg.org/spec/ATDM/About-ATDM
[2] https://www.castsoftware.com/
[3] https://www.vector.com/int/en/products/products-a-z/software/squore/squore-software-analytics-for-project-monitoring/

and SonarQube[4]), either commercial or open-source ones, the community has not concluded on a state-of-the-art solution that could be used as a basis for measuring TD (a full list with TD measurement tools that were found during our current research is presented in Section 2). Some shortcomings whose roots lie in the lack of a well-established way for assessing (i.e., measuring and identifying) TD principal, are presented in Figure 1.

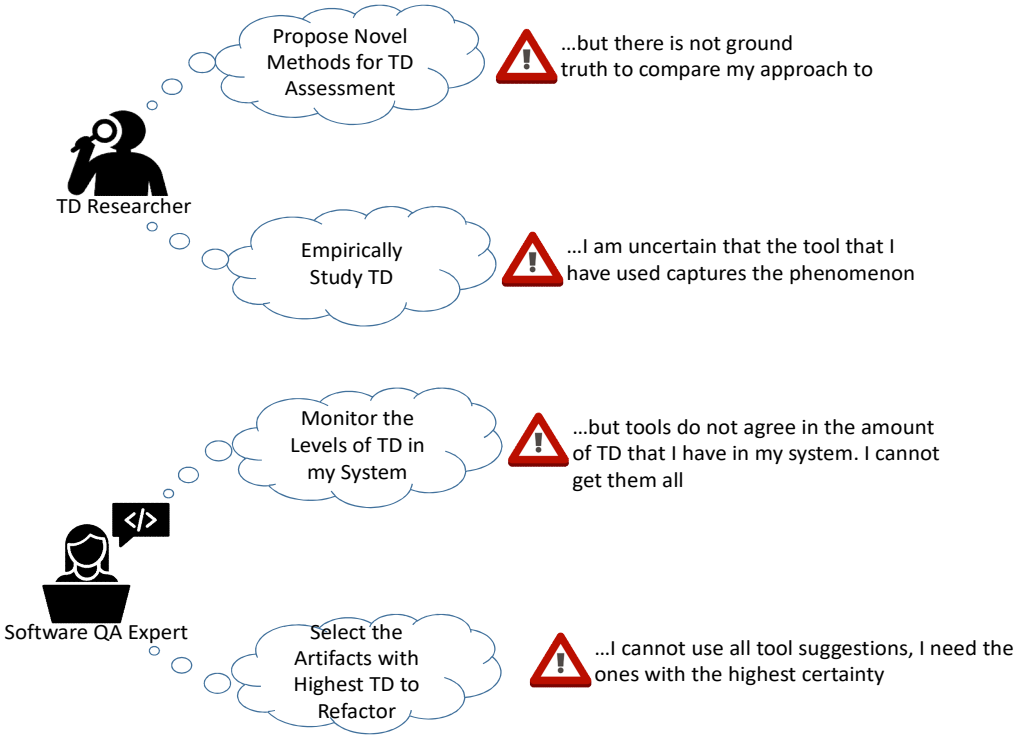


Fig. 1 Shortcomings from diverse TD measurements

Shortcomings in Research: The lack of a ground truth, even a commercial one, leads to construct validity threats in almost any kind of quantitative empirical study in the field, in the sense that it is not certain that any metric that attempts to capture TD principal is accurately measuring the real-world phenomenon. This problem does not lie only on limitations of the tools per se, but also on the underlying methodologies. In particular, each tool follows its own approach for detecting and measuring TD based on its own ruleset, while another tool might be based on an entirely different ruleset yielding a different amount for the total TD, but also pointing to different parts of the code that need to be mitigated. Moreover, there are several research efforts trying to associate TD items (i.e., violations of coding practices in software artifacts, which according to the OMG Specification on ATDM – see footnote in first page, are considered instances of TD principal) with quality attributes of software. For example, studies have focused on the relation between TD principal and the presence of crosscutting concerns in software requirements (Conejero et al., 2018), the existence of modularity violations, code smells and static analysis issues (Izurieta et al., 2012), code size, duplication and complexity (Nugroho et al., 2011) and architecture flaws (Nayebi et al., 2019). However, every such approach is heavily dependent on the employed tool for suggesting the ground truth, that is, the modules that actually have TD liabilities and need to be fixed. Obviously, if each tool identifies high-TD modules in a different way, the generalizability of these approaches is threatened to a large extent.

Shortcomings in Practice. Despite the widespread adoption of the TD metaphor, it is far from clear which tool IT managers should trust for monitoring TD, or deciding the mitigation actions to be applied. One option would be to employ more than one TD tools for the evaluation of their software, but this is a costly one, since most of the existing tools are available only with a commercial license. Moreover, someone should also consider the effort to deploy the tools on their premises, configure them properly and eventually familiarize development tools with their usage. In addition to that, even with the use of multiple tools, the union of all possible fixes sug-

[4] https://www.sonarqube.org

gested by different tools would yield an unrealistic amount of suggestions which would end up (even if they were accurate enough) to be useless in practice.

Acknowledging the widespread adoption of the TD metaphor and the inherent limitation of existing tools to capture TD principal in a globally accepted way that best fits developers' needs (Sadowski et al., 2015), in this study we aim at: (a) systematically ***investigating the degree of agreement*** among state-of-the-art TD measurement tools on identifying and prioritizing TD principal (i.e. the effort to remediate inefficiencies) at class/file level; and (b) proposing an ***agreement-based benchmark approach*** that contributes to: (i) the exploration of all *feasible assessments of TD principal* provided by a set of alternative TD tools, (ii) the *identification and characterization of few divergent "reference assessments"* (or archetypes); and (iii) the *extraction of a subset of modules* for which all employed tools agree on the presence of a high amount of TD principal, thus serving as an agreement-based benchmark of the "*validated*" top-rated classes/files[5] in terms of TD principal assessment.

To achieve the former goal, we make use of a well-known inter-rater agreement coefficient, namely the *Kendall's W coefficient of concordance* (Kendall, 1948). Regarding the second goal (benchmarking process), the *Archetypal Analysis* (AA) (Cutler and Breiman, 1994) is adopted, which is a multivariate statistical methodology that explores a multidimensional space of measurements with the aim of identifying a set of few reference points, namely the archetypes, located on the boundaries of the swarm of given points. The derived archetypes (or reference points) represent divergent profiles in the examined space, whereas the methodology encompasses a mechanism for the evaluation of resemblance coefficients contributing to the evaluation of similarity for each point to the derived archetypes.

To this end, we have employed three well-known tools that measure TD and analyzed 50 open source projects (25 Java and 25 JavaScript projects). The results of the proposed methodology are automatically reported through a web-based interactive toolbox to facilitate researchers and software practitioners to reproduce and explore the findings of the current study and easily retrieve a suitable benchmark for further experimentation (e.g., the training of other statistical or machine learning approaches to identify TD items). The ***TD Benchmarker*** toolbox is implemented using the Shiny framework[6] taking advantage of the R statistical language[7] in an easy-to-use frontend. The toolbox is a free and academic on-going research project developed by Statistics and Information Systems Group (STAINS)[8] at Aristotle University of Thessaloniki, Greece and is accessible through the paper's web page[9], at the website of the Software Engineering Group of University of Macedonia[10], Greece.

Apart from the empirical results and the extension of the body of knowledge in the field of TD management, an actionable outcome of this study is the provision of an agreement-based benchmark set of the most high-TD classes as indicated by the three tools altogether. The agreement-based benchmark is expected to alleviate the aforementioned limitations either directly or indirectly: regarding researchers the benchmark can be exploited for methodologies aiming at identifying TD items (targeting either high recall, or high precision), whereas it is also expected to aid practitioners since it will contribute to the development of novel tools that will be able to predict these items. More details on the implications of the benchmark for researchers and practitioners are provided in Section 5.

The rest of the paper is organized as follows: In Section 2 we present the available TD assessment tools that we have managed to locate throughout our research and explain why we ended up with the three employed TD tools. In Section 3, our case study design is presented along with the objectives, the research questions, the data analysis and the methodology. In Section 4 we present and discuss our results and in Section 5 the implications to researchers and practitioners are highlighted. In Section 6 we unfold possible threats to validity of our study while in Section 7 we provide related work regarding previous studies on comparison of tools measuring TD and benchmarks in software maintenance. Finally, we conclude in Section 8.

[5] The term 'class' refers to the unit of analysis for Java projects, while the term 'file' refers to the unit of analysis for JavaScript projects. Throughout the paper we primarily use the term 'class' for simplicity, but both units of analysis are considered, accordingly.
[6] https://shiny.rstudio.com
[7] https://www.r-project.org
[8] http://stains.csd.auth.gr
[9] https://se.uom.gr/index.php/projects/technical-debt-benchmarking
[10] https://se.uom.gr

2. TD Assessment Tools

In this section we discuss TD assessment tools that either have been proposed in the context of research efforts (usually open-source or free) or are available as commercial software, by providing a brief description of their capabilities. Then we provide more details on the three tools that have been selected for this study explaining the rationale for their selection.

During the previous years, numerous TD assessment tools have emerged; these tools are able to measure TD either in terms of cost or effort/time to repay TD. To identify as many tools as possible, we have conducted a non-systematic literature search, including grey literature (such as websites):

- Literature search: Regarding our literature search, we relied on the IEEE Xplore[11] and ACM Digital Library[12] search engines. Our search string was applied on the title and abstract fields and had the following form: "technical debt" AND (measurement OR assessment OR estimation) AND (tool OR platform). We gathered the studies that have been returned from the aforementioned search and filtered out those which neither introduce nor mention any TD tool in their title or abstract.
- Web search: Throughout our web search, we used major search engines such as Google, Bing and Yahoo, using the same query. The results led us either to the landing pages of the websites of the companies that own the tools or to articles introducing most well-known tools for assessing TD.

Right below we provide a short description of the TD assessment tools that we have managed to locate throughout our search. For each tool, we provide the study and the year in which it was first introduced or presented. The actual versions of the employed tools at the time of this study are provided in the end of this section.

AnaConDebt (Martini and Bosch, 2016) is a tool that focuses on Architectural Debt. Since a change in the architecture of a project can be really expensive and time consuming it is important to decide if and when this change should be implemented. The tool uses a large list of internal and external factors to estimate more accurately the future principal and interest. It helps managers to decide when it is the right time to refactor the code of their software.

CAST (Curtis et al., 2012) contains several sub-tools in order to provide the entire quality profile for the project. Health dashboard, Engineering dashboard, Security dashboard, CAST Appmark which is a benchmarking base to use as a comparison standard and CAST Enlighten with Imaging system that offers a visualization of the project. This tool helps companies to perform "Shift Left" techniques to detect the issues of a project in early stages of its life cycle. This way the cost of fixing the issues is more tolerable. The tool implements the C-CPP, CISQ, CWE, NIST-SP-800-53R4, OMG-ASCQM, OWASP, PCI-DSS-V3.2.1 and STIG-V4R8 standards. By performing static analysis, a list of issues is created. Only a part of the problems will be solved and this part defines the technical debt metric.

CodeScene (Tornhill, 2018) serves as a mean to preserve the quality of the code of the automated tests. It combines repository mining with static code analysis and machine learning. Static analysis can detect the problems in the project, but since the source code is treated as of the same importance, repository mining is necessary to recognize behavioral data and social factors that can affect future decisions of refactoring. The results of the metrics may have different meaning depending on the characteristics of each project. Machine learning is used to identify patterns in order to prioritize these metrics and assign them the appropriate weight. The final result of the tool is a catalogue with the problematic files ranked by their total impact.

DebtFlag (Holvitie and Leppänen, 2013) is a tool for capturing, tracking and resolving technical debt in Java systems. It consists of two parts; one plug in for Eclipse IDE which is responsible to collect the data from the source code, and one web application to visualize the results. These two applications connect via a database. The collected data is structured using the TDMF form, which was extended to cover the tool's needs. The tool offers the results in such a way that can be used to manage technical debt in two levels; project level and implementation level with micromanagement.

[11] https://ieeexplore.ieee.org
[12] https://dl.acm.org/

Debtgrep (Arvedahl, 2018) is an inhouse tool developed by Ericsson 4G 5G Baseband and its purpose is to prevent technical debt. It uses a file where all rules are declared using regex. The rules can contain forbidden words to restrict the usage of API and deprecated methods and also guidelines for design and architectural rules. The rules can be applied only to a specific part of code such as new code. This tool supports the communication between the developing team members and enhance the consistency and the uniformity of the project.

DV8 (Nayebi et al., 2019) is a commercial extension of Titan (Xiao et al., 2014a). DV8 functions with DRSpaces (Xiao et al., 2014b), which are groups of system's files that are architecturally related. Within DRSpaces, DV8 computes three modularity metrics (Decoupling level, Propagation Cost and Independence Level) and detects six architecture anti-patterns (Clique, Package Cycle, Improper Inheritance, Unstable Interface, Crossing and Modularity Violation). DRSpaces (i.e. the subsets of architecturally related files) that are involved in a selected set of issues are called 'architecture roots'. The tool calculates the added maintenance cost due to each instance of each anti-pattern, and the added maintenance cost of each architecture root. The source code analysis is performed by the Understand tool[13].

Kiuwan[14] is a proprietary code analysis tool that supports numerous programming languages and is capable of integrating with several IDEs. It can be obtained under a commercial license and it can also be tested within a free trial period.

NDepend (Chopra and Sachdeva, 2015) is a static analysis tool for .NET projects available in Visual Studio Market Place. It offers a variety of code quality metrics and a visualization of the dependencies in the project. The tool handles the source code as a form of database, and the user can define new evaluation rules using LINQ to perform queries on it. Other features of the tool include reporting service and the ability of comparison between the generations of the same project.

SonarQube (Campbell and Papapetrou, 2013) is a widely known tool used to track the quality and maintainability of source code. The tool implements the MISRA, CWE, SANS and CERT rule standards to provide measurements regarding complexity, duplications, code issues, maintainability, quality gates in combination with technical debt, reliability, security, project size and test coverage. In addition, there are many plugins to extend the available utilities, such as WebDriver for Selenium test analysis or AEM Rules set for Adobe. The measurement of technical debt is an important component of SonarQube. The tool calculates the debt by multiplying the number issues of each type with the average time the specific issue type needs to be fixed. Then the time is multiplied with the cost for each man-day. The average time and the cost can be configured by the user. It uses the SQALE method and provides a technical debt pyramid to help making decisions prioritizing tasks.

Squore (Baldassari, 2013) consists of three smaller tools. The first one, the analyzer, is used to collect data from different sources (source code, tests and hardware component information) and build the project's hierarchy tree. Then a more detailed measurement takes place for each one of the nodes based on the ISO, HIS, SPICE and MISRA rule standards. Last but not least, the tool also offers a dashboard for the visualization of the results. The tool can be a part of Jenkins continuous integration and can also recognize which files are most important to have Unit Tests in order to improve the efficiency.

TD-Tracker (Foganholi et al., 2015) is a web application, which provides a structured way to create a catalogue with the issues in a project. The protocol, which is implemented, consists of three stages. For the first stage there is a data collector where the problems are identified and a list is populated. The input data can come from either an external source where, with appropriate mapping, the data can be stored directly to the database of the application, or the integration with GitHub. After finishing the collection, the second stage begins where a semi-automated task takes place. A user has to review the previous list with the issues, and decide which of them are actual problems that need to be solved. Then there is the third stage with the longest duration of all three. In this stage a user assigns tasks related to technical debt and also monitors the progress of them.

TEDMA (Fernández-Sánchez et al., 2017) is an open tool, which analyzes different indices related to technical debt during the evolution of a project. It is open to integrate with third party tools to extend the analysis. It consists of three layers. The first is called Data Layer and holds the processes used to gather information about the project, which is examined. Currently, Git repositories are used as data input. The second is the Service Layer

[13] https://scitools.com/
[14] https://www.kiuwan.com/

where there are three basic services. (i) Data loader service is responsible for offering the source code in a processable form to the tool. Then analyzers such as PMD and Findbugs detect code smells and problems. (ii) Statistics service uses R to perform statistical analysis of the data. The analysis is performed at file level but it can be extended to other levels of abstraction. (iii) Technical debt management model service uses models in Java and R to support decision-making. The last layer is the Presentation Layer which is responsible for documentation and visualization.

VisminerTD (Mendes et al., 2019) is an open source web tool which monitors and manages technical debt comparing the results between different project's versions. When an issue is detected it can be tracked to determine whether its TD was paid off or not. It uses the Repository Miner tool to collect data and metrics from code repositories. VisminerTD uses queries to the database of the Repository Miner to gather the preferred information and present them to the user via a friendly interface. A set of graphical views are available to setup the search settings and then manage the technical debt items.

Table 1 lists the tools that we have identified along with information, such as the website with contact or download information, the corresponding study in which it was first introduced or presented, the type of license under which the tool is available (commercial/free), the programming languages that the tool supports for static code analysis and the type(s) of TD that it captures (as identified in previous studies (Alves et al., 2016; Li et al., 2015)). TD types refer to specific categories of TD (e.g., architectural, design, code) or sub-categories based on the cause of TD (e.g., architectural TD can be caused by architecture smells) (Li et al., 2015).

Table 1 List of identified TD assessment tools

TD Tool (Website)	Study	License	Supported Programming Languages	Captured TD Type(s)
AnaConDebt (https://anacondebt.com/node/7)	(Martini and Bosch, 2016)	commercial	Java	Architectural
CAST AIP[15] (http://www.castsoftware.com/)	(Curtis et al., 2012)	commercial	Java, ASP, C/C++, JavaScript, IOS, .NET, PHP, Python, ABAP, SQL and more (see full list at website)	Architectural, Code, Defect
CodeScene (https://codescene.io/)	(Tornhill, 2018)	commercial	C/C++, C#, Java, JavaScript, TypeScript, Python, Go, Visual Basic .Net, PHP, Ruby and more (see full list at website)	Code, Design
DebtFlag (-)	(Holvitie and Leppänen, 2013)	-	Java	Code
Debtgrep (-)	(Arvedahl, 2018)	Inhouse use only	Language agnostic	Architectural, Code, Design, People
DV8 (https://archdia.com/pages/dv8-user-guide) Understand: third party tool for source code analysis (https://scitools.com/)	(Nayebi et al., 2019)	commercial	Java, JavaScript, C/C++, C#, Python , PHP and more (see full list here: https://scitools.com/feature/supported-languages/)	Architectural
Kiuwan	-	commercial	ASP.NET, C, C#, C++,	Code

[15] We will refer to it as "CAST" from this point on

TD Tool (Website)	Study	License	Supported Programming Languages	Captured TD Type(s)
(https://www.kiuwan.com/)			Java, JavaScript, JSP, PHP, Python, VB.NET, SQL, Ruby and more (see full list at website)	
NDepend (https://www.ndepend.com)	(Chopra and Sachdeva, 2015)	commercial	.NET	Architectural, Code, Design, Test
SonarQube (https://www.sonarqube.org)	(Campbell and Papapetrou, 2013)	free	C/C++, C#, CSS, Go, Java, JavaScript, PHP, Python, Ruby, TypeScript, VB.NET and more (see full list at website)	Architectural, Code, Design, Defect, Test
Squore (https://www.vector.com)	(Baldassari, 2013)	commercial	Ada, C, C++, C#, Java, Cobol, PL, SQL, ABAP, PHP, Python, JavaScript	Code, Test
TD-Tracker (http://www2.fct.unesp.br/grupos/lapesa/tdr/)	(Foganholi et al., 2015)	free	Java, JavaScript, PLSQL, Apache Velocity, XML, XSL	Code, Design, Defect, Documentation, Infrastructure, Test
TEDMA (-)	(Fernández-Sánchez et al., 2017)	-	Java	Architectural, Code
VisminerTD (https://visminer.github.io/)	(Mendes et al., 2019)	free	Java	Architectural, Build, Code, Design, Defect, Documentation, Requirement, People, Test

Employed TD Assessment tools. Despite our goal to include in the study as many tools as possible, it has not been possible to employ all of the above tools for the measurement of TD for the target systems. Each tool had to fulfill the following conditions in order to be included in our study. In Table 2 we present which tools have been included in our study and which have been excluded (failing to satisfy all of the following conditions).

- Condition 1: The tool had to be accessible somehow (download link, ftp server, etc.) with comprehensive and sufficient documentation.
- Condition 2: The tool had to be able to analyze Java and JavaScript code (as the target systems of the study are open source Java and JavaScript projects).
- Condition 3: It was necessary to be able to obtain academic or research license for commercial or proprietary tools. For non-proprietary tools the condition was considered fulfilled.
- Condition 4: The tool had to provide an aggregate TD Principal index at class/file level, expressing effort in time or monetary terms, to remediate the identified inefficiencies (OMG Specification on ATDM[16]). Estimation of TD only at project level cannot be exploited to extract a benchmark set of most high-TD classes (for Java projects) and files (for JavaScript projects). This criterion is important for guaranteeing the uniformity of tools' output, so that the results are comparable

Table 2 List of TD tools with the conditions that they satisfied for their inclusion

[16] https://www.omg.org/spec/ATDM/About-ATDM

TD Tool	Condition 1	Condition 2	Condition3	Condition 4	Tool used?
AnaConDebt	✓	X	X		no
CAST	✓	✓	✓	✓	**yes**
CodeScene	✓	✓	✓	X	no
DebtFlag	X	X	✓		no
Debtgrep	X	✓	X		no
DV8	✓	✓	✓	X	no
Kiuwan	✓	✓	X		no
NDepend	✓	X	✓		no
SonarQube	✓	✓	✓	✓	**yes**
Squore	✓	✓	✓	✓	**yes**
TD-Tracker	✓	✓	✓	*could not deploy*	no[17]
TEDMA	X	X	✓		no
VisminerTD	✓	X	✓		no

*In case a tool did not fulfill Conditions 1 - 3 or could not be successfully installed and deployed, Condition 4 could not be checked and thus the field was left blank.

Ultimately, three tools were included in our study, namely CAST (version 8.3, year 2018), Squore (version 19.0, year 2019), and SonarQube (version 7.9, year 2019). All three tools are major TD tools, widely adopted by software industries and researchers and actively maintained, including comprehensive documentation.

3. Case Study Design

3.1 Goal and Research Questions

The goal of this study described according to the Goal-Question-Metric (GQM) approach (Solingen et al., 2002), is as follows: "**analyze** the TD of software projects **for the purpose of** assessing the level of agreement of state-of-the-practice TD assessors (tools) and forming agreement-based TD benchmarks of high-TD (or low-TD) classes **with respect to** the estimated level of principal, **from the point of view of** software researchers and practitioners **in the context of** Technical Debt Management (TDM)". For the sake of generalization, we perform the assessment of the level of agreement among tools for two programming languages, namely Java and JavaScript. The analysis of the two populations enables a meta-analysis in which we explore if the use of a different language has an effect on the level of agreement. The exploration of the programming language as a factor affecting the level of agreement between tools is performed for each one of the following research questions:

RQ1: *To what extent do the assessors (tools) agree in the ranking of classes in terms of TD measurement*?

RQ1 aims at investigating the degree to which widely employed TD tools agree upon the identification and assessment of TD at class level. The investigation of this RQ provides an insight to the diversity of the rules examined by each tool, in the sense that a low level of agreement essentially means that tools check for different rule violations. With a non-satisfying degree of agreement, it would be pointless to proceed with the benchmarking process and seek classes, which are identified as equally high-TD (or low-TD) by all assessors. Thus, RQ1 serves as a gate for the rest of our study.

[17] TD-Tracker was not included in our study because we were not able to install and deploy it successfully.

RQ2: *How many archetypes (reference assessments) are required to capture the diversity of the tools*?
The TD of classes in any examined system, as measured by the employed tools, form a set of observations in a multidimensional space, in which each dimension represents TD evaluations provided by a specific tool. RQ2 aims at exploring this multidimensional space and determine the optimal number of archetypes, located on the boundaries of this space, so as to efficiently capture the diversity of all feasible assessments provided by the set of the examined TD tools. For example, this RQ can answer, whether few reference assessments are able to approximate the convex hull of the TD evaluations, which practically means low diversity among TD assessors or whether a higher number of archetypes would be required to accurately characterize the spectrum of TD measurements for a given system.

RQ3: *Which are the characteristics of the extracted archetypes*?
RQ3 aims at characterizing the extremal points that accurately encompass the space of TD measurements for all examined classes. The identified reference assessments essentially form a set of distinct archetypes, i.e., class profiles according to the measured level of TD. Two expected archetypes correspond to the profiles of classes having high or low TD based on the results of all employed tools. However, other archetypes may be identified based on the shape of the space of the obtained TD measurements.

RQ4: *Which classes should be selected to form an agreement-based benchmark of top-TD modules?*
To facilitate the work of developers or researchers who seek a golden set of classes that can be safely assumed to be high-TD or low-TD, this RQ aims at formally extracting sets of classes which are close to a selected class profile or archetype. Retrieving for example the classes, which are in the close vicinity of the archetype depicting high TD in all employed tools, a development team can be confident that these classes suffer significantly from rule violations. Similarly, a researcher can use such a benchmark for training effective machine learning techniques to identify TD based on different parameters of the code, people or processes involved in the development.

3.2 Selection of cases

For this case study we analyzed 50 open source projects (listed in Table 3). The selected projects, which are 25 Java and 25 JavaScript projects, have been analyzed considering their classes (for Java) and their files (for JavaScript) as units of analysis. The choice of classes/files as units of analysis allows us to trace the existence of TD at a low level of granularity, providing a common ground for comparison among the three tools. The criteria for selecting the 50 projects were the following:

- All cases had to be Java and JavaScript projects stored on public repositories.
- All selected cases had to be among the most popular repositories, with more than 3K stars in GitHub.
- In order to obtain a representative dataset, the selected projects had to vary in terms of size, per language.
- All cases had to be actively maintained till the time of this study. This was not a strict criterion since projects with a release around the last year before the project selection process were not excluded from the study.

3.3 Data Collection

The source code (excluding test files) of each project was analyzed three times: one time for each of the employed TD tools. All three tools provide a metric of the total effort needed to eliminate technical debt in each class/file. This is the metric that was chosen for analysis since it provides a common ground for comparison. An issue that had to be addressed was that each tool has a different way to provide the results of its analysis. It was necessary to convert the result sets from each tool to the same form so as to proceed with further data processing.

Table 3 Characteristics of analyzed projects

Java				JavaScript			
Project	**Description**	**LOC**	**Version**	**Project**	**Description**	**LOC**	**Version**
arduino	Physical computing platform	27K	1.8.10	ace	Code editor	117K	1.4.8
arthas	Java Diagnostic tool to troubleshood production issues	28K	3.1.7	angular.js	Web development framework	53K	1.7.9
azkaban	Workflow manager	79K	3.81.0	atom	Text editor	138K	1.44.0
cayenne	Java object to relational mapping framework	348K	3.1.2	bluebird	Promise library	20K	3.7.2
deltaspike	CDI management	146K	1.8.2	bower	Front end package management	10K	1.8.8
exoplayer	Android media player	155K	2.11.1	brackets	Code editor	129K	1.14.1
fop	Print formatter using XSL objects	292K	2.3	Chart.js	Chart designer	10K	2.9.3
gson	Java library to convert Java Objects to JSON	25K	2.8.6	exceljs	Excel Workbook Manager	23K	3.8.0
javacv	Wrappers of commonly used libraries	23K	1.5.2	fabric.js	Framework for HTML5 canvas element	20K	4.0.0
jclouds	Toolkit for java cloud applications	482K	2.0.2	jquery	Javascript library	20K	3.4.1
joda-time	Date and time handling	86K	2.10.5	karma	Tool for test driven development	5K	4.4.1
libgdx	Game development framework	280K	1.9.10	Leaflet	Mobile friendly interactive maps	24K	1.6.0
maven	Software project management and comprehension tool	106K	3.5.4	less.js	Language extension for CSS	12K	3.11.1
mina	Network application framework	35K	2.0.19	moment	Parsing validating manipulating and formatting dates	183K	2.24.0
nacos	Cloud application and microservices build and management	60K	1.1.4	mongoose	Tool for MongoDB object modeling	22K	5.8.12
opennlp	Natural Language Processing toolkit	93K	1.8.4	mysql	MySQL protocol implementation	8K	2.18.1
openrefine	Data management	69K	3.2	node	Node.js JavaScript runtime	130K	13.9.0
pdfbox	Library of processing pdf documents	213K	2.0.9	pdf.js	PDF viewer	69K	2.2.228
redisson	Java Redis client and Netty framework	133K	3.12.0	plotly.js	Chart design library	92K	1.52.2
RxJava	Composing asynchronous and event-based programs with observable sequences	310K	3.0.0	pm2	Production process manager	15K	4.2.3
testng	Testing framework	85K	7.1.1	prettier	Code formatter	25K	1.19.1
vassonic	Performance framework for mobile websites	7K	3.1.1	sails	Realtime MVC Framework for Node.js	10K	1.2.2
wss4j	Java implementation for security standards in web applications	136K	2.2.2	sequelize	Node.js ORM	17K	5.21.4
xxl-job	Distributed task scheduling framework	9K	2.1.2	webpack	Bundler for js files for usage in a browser	36K	4.41.6
zaproxy	Security tool	187K	2.9.0	yarn	Dependency management	24K	1.22.0

- SonarQube has a WEB API available, so with the use of appropriate tools the results have been gathered in json format. The API allows the filtering of the results in order to exclude test and properties files. SonarQube provides the results grouped by file. Besides file name, the number of the issues for each severity level, blocker, critical, major, minor and info, was summed up to the total amount of issues of each class. All of them contribute to the SQALE index of the file, which is the metric depicting the effort to eliminate TD.
- Squore provided the results in .csv files, which could be exported through platform's user interface. In this case a parser was necessary to read the .csv files. Using the previous SonarQube exports as reference, the files were filtered to exclude test and property files as before. Blocker, critical, major and minor issues were summed up to get the total issues for each class. Technical debt metric is provided in man days and man hours and it had to be converted in minutes to form a canonical technical debt index with the same units as for the previous tool.
- CAST provides metrics for the total project and not per file through its user interface. In this case, the results were retrieved directly from the database schema that the software uses during the code quality analysis. With appropriate SQL query, which was provided by the CAST team, csv files were extracted containing a list of total occurrences of each issue per class. With a new parser these issues were grouped, aggregating the TD in minutes and the total violations per class. Then again, the files of the classes were filtered with those of SonarQube as reference (test and property *files* were filtered out).

To obtain a common and structured form of the results, the exports from the tools were transformed into XML files. As a result, an XML file per project for every tool was generated. The XML contains all the classes/files with some TD in the system, along with the total issues detected in the class and the amount of TD as calculated by the corresponding tool. With the results in the same form it was possible to merge them into a single dataset. This dataset was finally grouped by class for Java and by file for JavaScript projects, containing the path of the file and the TD of the class/file as calculated by each tool. The dataset for the 25 Java and the 25 JavaScript projects is publicly available at Zenodo[18].

3.4 Data Analysis Methodology

In this section, we present background information necessary for facilitating the understanding of the statistical methodologies used to address the research questions of the current study.

3.4.1 Inter-rater Agreement (RQ1)

For the formal representation of our experimental setup, consider that the collection of TD assessments generated by all three tools, as described in Section 3.3, resulted in a $n \times p$ matrix X (Table 4), in which, each row represents a class, whereas each of the p column vectors provides the rankings of TD measurements evaluated by a specific tool for a given class. At this point, we have to clarify that in the proposed approach we decided to utilize the rankings instead of the raw TD measurements, since our intention was to keep the dataset immune to variations of TD measurements due to different scales among the three tools. Indeed, a tool might follow a stricter ruleset for the measurement of TD which might result in much higher TD of classes compared to the assessments of the rest of the tools. However, the ranking of the measurements among all tools remain unaffected by absolute values and thus is a more suitable approach for comparison. As far as the ranking mechanism concerns, we adopted the fractional ranking approach, in which the sample ranks of the values in a vector are computed, whereas in cases of ties the average of the ordinal rank (or fractional rank) is assigned to each tied observation.

Table 4 Representation of the dataset from the TD assessment results from each employed tool[19]

Class	Tool 1	Tool 2	…	Tool *p*
C_1	r_{11}	r_{12}	…	r_{1p}
C_2	r_{21}	r_{22}	…	r_{2p}

[18] https://doi.org/10.5281/zenodo.3966202

[19] Although we have used 3 tools, we generalize the theoretical presentation of our approach for *p* tools

…	…	…	…	…

For reasons of simplicity, we present the methodology of the proposed framework on a demonstrative example (opennlp project) utilizing the TD assessments from two tools (CAST and Squore). In this case, the TD assessments can be visualized through a scatter plot (Figure 2), in which each point represents a specific class with coordinates the TD rankings evaluated by the CAST (x -axis) and Squore (y -axis) tools.

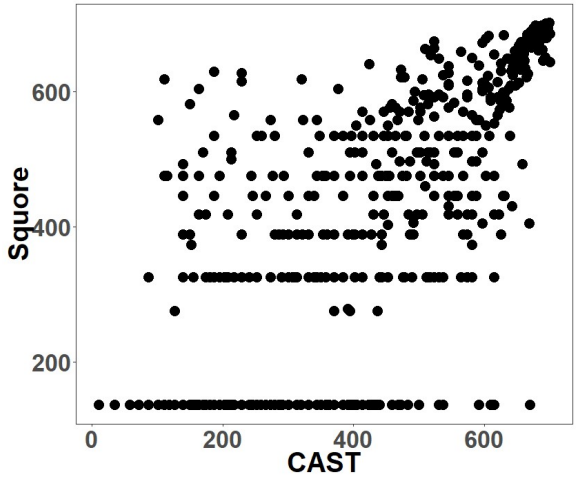


Fig. 2 Scatter plot for rankings of TD measurements of opennlp project as evaluated by TD tools (CAST, Squore)

The exploration of the pattern for the swarm of points provides certain information regarding the agreement of the employed tools. More precisely, it seems that there is a subset of classes lying to the upper right corner that are identified as the most high-TD (high rankings for TD measurements) by both tools. On the other hand, the inspection of the graph indicates also that Squore identifies a subset of classes that accumulate the lowest TD assessments but at the same time, these specific classes present an amount of TD ranging from the lowest up to the highest ranks according to the CAST tool. Finally, there is also a small number of classes assessed as high-TD by the Squore tool, but at the same time, the CAST tool tags them as classes accumulating a relatively small amount of TD. Hence, a critical question that deserves further investigation is the extent to which these tools agree upon the assessments of TD for a given set of classes.

To this regard, we make use of a statistical measure, namely the *Kendall's W coefficient of concordance* (Kendall, 1948), which belongs to the broader branch of methodologies known as *inter-rater agreement analysis*. In general, there is a plethora of measures for evaluating the agreement among assessors and the choice should be based on (*i*) the total number of assessors that assign to each subject a unique measurement (or rating), (*ii*) the scale of measurement (nominal with two or more categories, ordinal, continuous scale) that is assigned to each subject and (*iii*) the objectives of the analysis (Gwet, 2014). More specifically, the Scott's π (Scott, 1955) and Cohen's κ (Cohen, 1960) are well-known measures for inter-rater agreement on a nominal dichotomous (No/Yes, Negative/Positive) scale that can be used in cases, where there are exactly two assessors. For the case of multiple assessors (more than 2) on nominal (either dichotomous or with multiple categories) or ordinal scales, the Fleiss's κ (Fleiss, 1971), which is a generalization of Scott's π coefficient and the weighted Cohen's κ (Cohen, 1968) are possible choices that take into account not only the agreement but also the disagreement among them. All the aforementioned coefficients share the same rationale that is to evaluate and statistically test whether the average agreement between two (or more assessors) is significantly different than chance. An additional problem to the ordinal ratings, besides the fact that agreement and disagreement are no longer distinct notions (Gwet, 2014), is the fact that there is another kind of agreement that may be of interest. This can be defined "*as the agreement among raters with respect to the ranking of subjects*" (Gwet, 2014), which, in our case, is related to the process of evaluating whether all assessors, agree on which classes are the highly-ranked, the second highly-ranked and so on. In this case, the selection of the most appropriate agreement coefficients should belong to the branch of measures of concordance (Gwet, 2014), since in general the variation of kappa statistics evaluate the absolute agreement between ratings, while concordance coefficients measure the association between ratings. Finally, a well-known limitation of kappa statistics is their dependence on the number of categories of the response measurement, since they tend to be generally higher, when there are fewer categories (Watson and Petrie, 2010).

Summarizing, the choice of Kendall's W concordance coefficient instead of other kappa measures of agreement was based on the facts that (*i*) it serves in a straightforward manner the investigation of RQ1, which is related to the evaluation of the degree of agreement among TD measurement tools and (*ii*) it handles in an appropriate way the characteristics of our experimental design, which involves three TD assessment tools (CAST, Squore, SonarQube) and the derived rankings ranging from 1 up to n (the total number of the examined classes). Due to the existence of a high number of tied ranks in each tool (Figure 2), we make use of a modification of the original statistic that provides a correction for ties. The Kendall's W statistic (Salkind, 2010) is defined as

$$W = \frac{12\sum_{i=1}^{n} r_i^2 - 3p^2 n(n+1)^2}{p^2 n(n^2-1) - pT} \quad (1)$$

where, n is the total number of the examined classes, $\sum_{i=1}^{n} r_i^2$ is the sum of the squared sums of ranks for each of the n classes and p is the total number of the examined tools (three in our case). The term T is a correction factor for tied ranks that is evaluated via the following formula

$$T = \sum_{k=1}^{g} (t_k^3 - t_k) \quad (2)$$

in which, t_k is the number of tied ranks in each of g groups of ties, whereas the sum is evaluated over all groups of ties found in all p tools of Table 4. Kendall's W can take a range of values from 0 (indicating no agreement) to 1 (indicating a perfect agreement among assessors). In addition, Schmidt (Schmidt, 1997) provides specific guidance through rules of thumb on how researchers should interpret experimental results based on the evaluation of the Kendall's W statistic. More specifically, a coefficient of 0.7 or higher can be interpreted as a strong agreement among the set of assessors. For example, the evaluation of the Kendall's W concordance coefficient for the set of classes of our demonstrative example indicates a statistically significant strong agreement between the CAST and Squore tools regarding their TD assessments, $W = 0.874, p < 0.001$.

3.4.2 Benchmarking through Archetypal Analysis (RQ2 - RQ4)

From what we have already mentioned, there are several available tools for assessing TD, whereas each tool is based on a different ruleset that may result to divergent TD assessments for a given project. Although, this fact could lead to the identification of alternative mitigation actions, the empirical evidence reveals that software practitioners and development teams usually base the measurement process of TD on a single tool. Having in mind that there is no ground truth for assessing TD, there is an imperative need for the empirical examination of the diversity produced by the utilization of a set of alternative TD tools. Indeed, the findings from the indicative example discussed in the previous section revealed that despite the fact that there is a strong agreement between the assessments provided by the two examined tools, the tools also disagree upon the measurement of TD of some classes.

Towards this direction, we propose an agreement-based benchmark approach contributing to the empirical characterization of the assessments provided by a set of p alternative tools with respect to the derived TD evaluations for a given set of n examined classes. The benchmark framework is based on a statistical approach, namely *Archetypal Analysis* (AA) (Cutler and Breiman, 1994). Describing the general principles of the methodology, AA is a data-driven multivariate method that explores a multidimensional space of points (or observations) with the aim of identifying certain observations, namely the archetypes, located on the boundaries of a swarm of given points (or *convex hull*). An interesting property of the methodology is the fact that the swarm of points can be represented as convex combinations of the archetypes. The latter provides a straightforward mechanism supporting the identification of a subset of points that are closer to a specific archetype, which in turn, can be used for benchmarking purposes. We note that although AA and Cluster Analysis (CA) share common ground, i.e., the exploration of a multidimensional space of points with the aim of identifying certain observations that represent specific profiles, they also present fundamental differences in terms of their goals. More specifically, AA aims at identifying points that lie on the convex hull of observations, or in other words

observations that can be thought as "extreme" (at the edge of the set of points). In contrast, CA techniques focus on the exploration of points within the multivariate space with the objective of identifying points at the center of the profile.

In our context, the input for AA is the $n \times p$ matrix X (Table 4) representing the rankings of TD assessments derived from the analysis conducted through the utilization of a set of p tools for a given project with n classes. The algorithm of AA seeks for a matrix Z of $k \times p$, where k and p are the number of archetypes and dimensions (examined tools in our case), respectively through the computation of two coefficient matrices a and b minimizing the residual sum of squares (RSS) defined as

$$\text{with } Z = X^T b \tag{3}$$

where $\|\cdot\|_2$ denotes the Euclidean matrix norm, subject to the following constraints:

$$\sum_{j=1}^{k} a_{ij} = 1 \quad \text{with } a_{ij} \geq 0 \text{ and } i = 1, \ldots, n \tag{4}$$

$$\sum_{i=1}^{n} b_{ji} = 1 \quad \text{with } b_{ji} \geq 0 \text{ and } j = 1, \ldots, k \tag{5}$$

These constraints frame the two general properties of AA which are: (*i*) the approximated data (swarm of points) are convex combinations of the archetypes, i.e. $X = aZ^T$, and (*ii*) the archetypes are convex combinations of the data points, i.e. $Z = X^T b$. The term "convex combination" refers to the linear combination of points, when all coefficients are non-negative and their sum is equal to 1. Computationally, the algorithm reduces the RSS in Eq. (3) by iteratively calculating the archetypes along with the coefficient matrices a and b. Summarizing, the archetypal solution provides an approximation of the convex hull defined by the swarm of points in the multi-dimensional space through the evaluation of a few, not necessarily observed points, lying on the boundaries of the observed points.

Due to the intuitive rational and interesting properties of AA, the method has been widely used for benchmarking purposes in many scientific domains (Moliner and Epifanio, 2019), e.g. such as marketing (Li et al., 2003), astrophysics (Chan et al., 2003), sports analytics (Eugster, 2012), biology (Thøgersen et al., 2013), medicine (Elze Tobias et al., 2015), scientometrics and bibliometrics (Seiler and Wohlrabe, 2013), multi-document summarization (Canhasi and Kononenko, 2014), neuroscience (Tsanousa et al., 2015) etc. In Software Engineering, AA has been introduced in (Mittas et al., 2014; Mittas and Angelis, 2020), in which the objectives were the evaluation of the predictive capabilities of a set of Software Effort Estimation (SEE) models and the building of ensembles using a subset of inferior models, whereas in (Kosti et al., 2016), the authors explored psychometric data in order to extract different software engineers profiles based on measurements from their personality and behavioral characteristics.

Following a similar approach to (Porzio et al., 2008), in this study, AA constitutes the core methodology of a three-step process that facilitates the examination of the diversity of TD assessments provided by a set of alternative tools with the aim of identifying a set of classes exhibiting similarity to a selected archetype that can be used, in turn, for benchmarking purposes. Such classes can, for example, be classes with increased levels of TD as measured by all three tools, or TD-clean classes, which present limited inefficiencies. The three basic steps of the proposed approach summarized into the following points constitute the basis of our methodology for providing answers to RQ2 - RQ4:

1. Identification of archetypes representing the reference assessments through the exploration of the diversity of TD assessments derived from the set of employed tools (RQ2).
2. Reification of archetypal solution into the context of TDM through the identification of their characteristics (RQ3).
3. Identification and retrieval of a set of classes that are close to archetypes depicting either high TD or low TD assessments as suggested by all employed tools (RQ4).

The implications of the three previous steps are clearly demonstrated through the application of the approach on the indicative example described in previous section. In Figure 3, the boundary of the grey area defines the convex hull of all TD assessments derived from the CAST and Squore tools through the examination of classes from opennlp project. Based on the principles of AA, the archetypes representing the reference assessments will lie on this boundary, whereas the shape of the convex hull provides straightforward answers regarding the diversity of the examined set of TD tools.

A critical decision that someone has to take is the selection of an appropriate number of the k archetypes that approximates the convex hull in an efficient way. Certainly, the number of archetypes plays a significant role to the efficient representation of the swarm of the observed points, since the diversity of the convex hull may be better captured, as the number of archetypes increases. In contrast, one has to take into consideration that an unnecessary large number of archetypes might not contribute further to the approximation of the convex hull, whereas it would also affect the benchmarking process, since the objective is the extraction of few reference assessments representing useful profiles of practical importance to both researchers and practitioners in TDM.

To this regard, the graphical inspection of the swarm of TD assessments (Figure 3) suggests that the efficient number of archetypes capturing the diversity of the two examined TD tools is $k = 4$ archetypes. In the trivial case of $k = 1$, the archetypal solution is the centroid of the two-dimensional space representing the TD assessments matrix (Table 4), whereas its coordinates are easily calculated by the univariate sample mean values of TD rankings from each tool (sample means of CAST and Squore TD columns in Table 4).

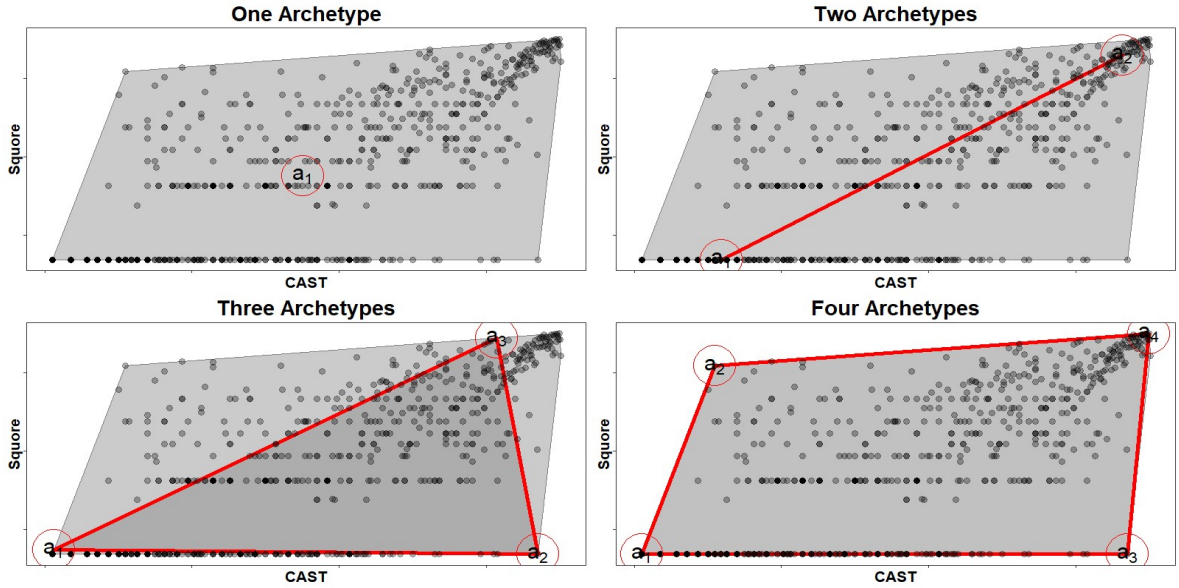


Fig. 3 Archetypal solutions (CAST, Squore) for opennnlp project

Although the graphical inspection constitutes a straightforward manner for the identification of the appropriate number of archetypes in the special case of the two-dimensional space, i.e. the examination of assessment provided by two TD tools, this is not the case, when the number of the examined TD tools is higher than two (p>2). In order to provide certain guidelines about the decision upon the appropriate number of archetypes, Cutler and Breiman (Cutler and Breiman, 1994) suggest the utilization of the graphical inspection of the RSS reduction plot (or elbow plot). The RSS plot (Figure 4) constructed after consecutive executions of AA for different values of k , $(k = 1,2,3,4,5)$ confirms our intuitive beliefs derived from the graphical inspection of the two-dimensional example. More specifically, if we consider that the line displaying the RSS reduction looks like an arm, then an elbow appears at $k = 4$, pointing out the optimal number of archetypes. The idea is that after this specific point ($k > 4$), the line flattens and hence, the extracted solution ($k = 5$) does not contribute to any further reduction of RSS. Summarizing, the practical implication of the first step (Step 1) of our proposed approach on the indicative example, is that four reference assessments (archetypes) can capture the diversity of TD rankings derived from the static code analysis (by two tools) for the set of the examined classes of opennlp project.

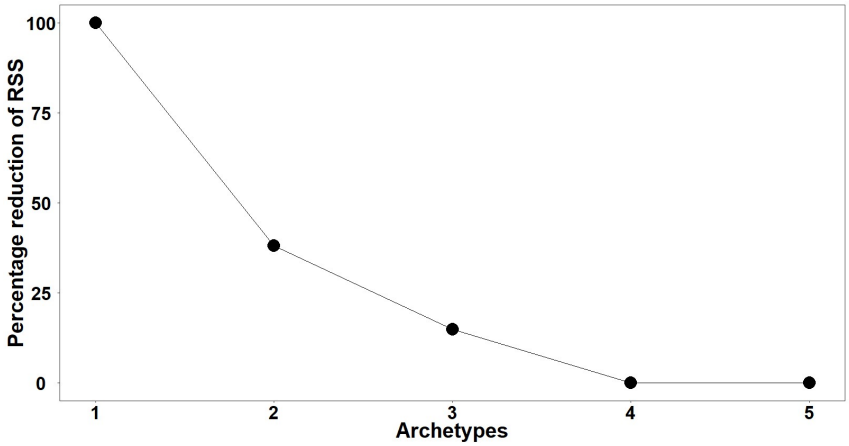


Fig. 4 RSS plot (CAST, Squore) for opennnlp project)

In the second step (Step 2), we focus on understanding the characteristics of the derived archetypes with the aim of extracting information regarding their meaning from a practical point of view in TDM. The relative position of the four archetypal solutions (Figure 3) and the graphical examination of the profiles plot (Figure 5) provide a clear overview of what each archetype really represents. More specifically, the profiles plot shows the evaluated TD rankings (CAST and Squore coordinates, Figure 3) for each archetype of the final solution. In addition, we can also observe that the examination of the characteristics provides also a semantic categorization of the derived archetypes into two distinct groups, which are (i) the *Ruler* and (ii) the *Rebel* archetypes[20] (Pearson, 2015). The former group (The Ruler) reifies a reference assessment profile, in which the two tools agree upon either on low (The Min-Ruler archetype a_1) or high (The Max-Ruler archetype a_4) TD rankings assessments. The latter group (The Rebels) reifies a reference assessment profile, in which the two tools do not agree on their TD rankings assessments signifying a completely divergent behavior of the two assessors. Overall, the four archetypes represent the following distinct reference assessment profiles with the following characteristics:

- **The Max-Ruler** (archetype a_4 in Figure 3d) represents the reference assessment corresponding to the profile of classes accumulating high amount of TD based on the results of both tools (CAST and Squore).
- **The Min-Ruler** (archetype a_1 in Figure 3d) represents the reference assessment corresponding to the profile of classes accumulating low amount of TD based on the results of both tools (CAST and Squore).
- **The Rebel 1** (archetype a_2 in Figure 3d) represents the reference assessment corresponding to the profile of classes accumulating low amount of TD based on the results of the analysis from the CAST tool, but on the same time, high amount of TD based on the results of Squore tool.
- **The Rebel 2** (archetype a_3 in Figure 3d) represents the reference assessment corresponding to the profile of classes accumulating high amount of TD based on the results of the analysis from the CAST tool, but on the same time, low amount of TD based on the results of Squore tool.

[20] The idea of archetypes was developed by psychologist C. Jung in his studies about drivers of human behavior. Pearson suggested the use of 12 archetypes among which the 'Ruler' denotes personalities whose goal is to *create a prosperous, successful family or community*, while for a 'Rebel' (also known as Outlaw) the motto is that *rules are made to be broken*. In our context, the 'Ruler' profile denotes a community of classes sharing the same assessment by all employed tools, while the 'Rebel' points to tools that in some sense break the rules and identify TD items in a different way than the rest.

Fig. 5 Reference assessment profiles (archetypes) (opennnlp project)

After the reification of the archetypes, the final step (Step 3) of the proposed approach involves the identification and retrieval of a set of classes that are close to a specific archetype gathering certain characteristics that can be used for TDM purposes. A critical challenge in the TD community raises from the fact that although there are several available tools for measuring and monitoring TD, the community has not concluded on a state-of-the-art solution that could be used as a ground truth for measuring TD. Developers and researchers acknowledge that TD estimates provided by any single tool are inherently subjective, reflecting a particular strategy for the identification of TD items. The existence of a basis of classes that are assessed as high TD modules by various tools would point to classes that can objectively be classified as validated high-TD modules and would boost relevant research. Currently, the lack of a commonly agreed way of quantifying TD impedes the development of approaches that could built on top of TD measurements, as in the case of machine learning approaches seeking to identify code or design problems employing alternative parameters as inputs. The ability to derive a benchmark of classes being close to the Max-Ruler archetype can be directly leveraged for training supervised learning-based algorithms. Similarly, the classes which have been validated as high-TD by all tools can be analyzed by development teams to seek non ideal coding practices and patterns so as to avoid them in future releases. On the other hand, benchmarks of classes formed by those that are close to Rebel archetypes essentially designate design or code inefficiencies which are captured by only one of the available tools, possibly pointing to unique features identified by a particular ruleset. As a result, the union of classes belong to these sets would ensure the widest possible coverage of TD liabilities.

The evaluation of the adjacency of a certain TD assessment (representing a given class) to each archetype can be practically accomplished through the matrix of the a -coefficients (Eq. 4). More importantly, due to the first property of AA, i.e. the approximated points are convex combinations of the archetypes that are summed to unity, the computed a -coefficients for each TD assessment provide an easily interpretable mechanism for quantifying its resemblance to all archetypes. Table 5 displays the classes and their TD assessments that are close to the Max-Ruler archetype according to the threshold value of $a = 0.80$ for characterizing the neighboring classes. By setting the threshold value of $a = 0.80$, a set of 84 out of 701 total classes (almost 12% of the examined classes) can be considered as adjacent to the Max-Ruler archetype. This practically means that a practitioner has access to a set of classes that have been validated as high-TD classes by all tools. Due to space limitations, we present only the first and last five classes from the 84 that are close to the Max-Ruler archetype. Interpreting the vector of α-coefficients for a randomly selected class, e.g. C_{42} with ($\alpha_{\text{Min-Ruler}}$, $\alpha_{\text{Rebel 1}}$, $\alpha_{\text{Rebel 2}}$, $\alpha_{\text{Max-Ruler}}$) = (0.091, 0.000, 0.001, 0.908) (last four columns of Table 5), we can infer that C_{42} is 9.1%, 0.0%, 1.0% and 90.8% similar to the Min-Ruler, Rebel 1, Rebel 2 and Max-Ruler archetypes, respectively, and for this reason it is considered as a neighboring class to the Max-Ruler archetype.

Finally, Figure 6 visualizes the neighbourhood of the Max-Ruler archetype (corresponding to the TD measurements of the abovementioned 84 classes) with a black-scaled colour indicating the degree of resemblance for each TD assessment to this specific reference assessment profile. Moreover, points denoted by empty red circles represent classes that are not similar to the Max-Ruler archetype ($a < 0.80$) in terms of their TD assessments.

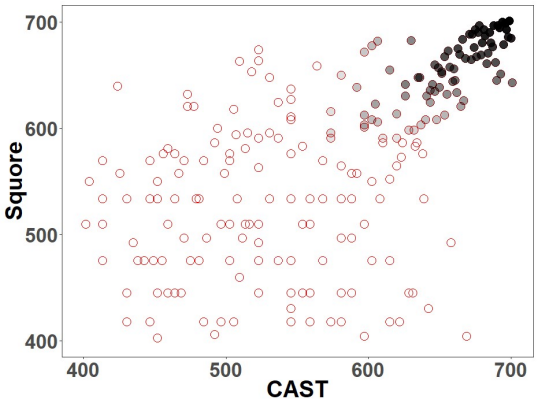


Fig. 6 Scatter plot for neighboring classes to the Max-Ruler archetype (CAST, Squore) (opennnlp project)

The final step of the benchmarking process is supported by a web application (TD Benchmarker) that has been developed which enables the extraction of benchmarks, consisting of classes being close to a selected archetype, for varying threshold values. Interested researchers can download the agreement-based benchmark of choice and retrieve the identified classes for further experimentation. Moreover, the application provides graphical illustrations of the RSS plots and the reference assessment profiles. TD Benchmarker is available online[21].

RQ1: For RQ1, where we examine the level of agreement of the used tools with respect to the measured TD of classes, we employed the Kendall's W coefficient of concordance which belongs to the broader branch of methodologies known as inter-rater agreement analysis.

For RQ2 – RQ4 we propose an agreement-based benchmark process, which is based on a statistical approach, namely Archetypal Analysis (AA).

RQ2: In the first step of the benchmarking process, our aim is to calculate the required number of archetypes to effectively capture the diversity of the tools. In this regard, we determined the appropriate number of archetypes via the graphical inspection of the RSS reduction plot (or elbow plot).

RQ3: In the second step of the benchmarking process we focus on understanding of the characteristics of the derived archetypes in our attempt to interpret them from the Technical Debt Management (TDM) point of view. Through the graphical examination of the Archetypal Solutions figure we were able to distinguish two main categories of the archetypes; the Ruler and the Rebel archetypes.

RQ4: The final step of the benchmarking process involves the identification and extraction of a set of classes that are close to a specific archetype with specific characteristics that can be interpreted in terms of TDM. The extraction of the aforementioned set of classes was accomplished through the matrix of α -coefficients (Eq. 4).

[21] **tool**: https://se.uom.gr/index.php/projects/technical-debt-benchmarking
source code: https://github.com/theoam/TDBenchmarker

Table 5 Indicative set of classes that are close to the Max-Ruler archetype (CAST, Squore) (opennnlp project)

Class		**Ranking**		**a -coefficient**			
ID	Name	Squore	CAST	The Min-Ruler	The Rebel 1	The Rebel 2	The Max-Ruler
C_1	/main/java/opennlp/tools/stemmer/snowball/turkishStemmer.java	701	699	0.000	0.000	0.000	1.000
C_2	/main/java/opennlp/tools/stemmer/snowball/englishStemmer.java	699	696	0.001	0.004	0.000	0.995
C_3	/main/java/opennlp/tools/stemmer/snowball/frenchStemmer.java	700	694	0.000	0.008	0.000	0.992
C_4	/main/java/opennlp/tools/stemmer/snowball/portugueseStemmer.java	695	692	0.008	0.002	0.000	0.989
C_5	/main/java/opennlp/tools/stemmer/snowball/hungarianStemmer.java	693	697	0.003	0.000	0.010	0.988
…	…	…	…	…	…	…	…
C_{42}	/main/java/opennlp/tools/formats/Conll03NameSampleStream.java	648	636	0.091	0.000	0.001	0.908
…	…	…	…	…	…	…	…
C_{80}	/main/java/opennlp/tools/formats/ontonotes/OntoNotesNameSampleStream.java	650	581	0.072	0.117	0.000	0.812
C_{81}	/main/java/opennlp/tools/ml/BeamSearch.java	616	573.5	0.142	0.047	0.000	0.811
C_{82}	/main/java/opennlp/tools/util/ObjectStreamUtils.java	591	573.5	0.182	0.000	0.012	0.807
C_{83}	/main/java/opennlp/tools/cmdline/namefind/TokenNameFinderTrainerTool.java	591	620	0.111	0.000	0.082	0.807
C_{84}	/main/java/opennlp/tools/lemmatizer/LemmatizerME.java	591	610	0.126	0.000	0.067	0.807

4. Results

4.1 RQ1: To what extent do the assessors (tools) agree in the ranking of classes in terms of TD measurement?

Based on the proposed methodology (see Section 3.4), the objective is to investigate the degree of agreement among the applied TD tools (RQ1). Table 6 summarizes the results concerning the evaluation of the Kendall's W concordance coefficient for the set of the examined 50 projects. The results suggest that, in general, the three TD tools converge on the identification and measurement of TD at class/file level. Overall, the coefficient values range from 0.520 (for atom JavaScript project) to 0.853 (for javacv Java project). To this regard, it is meaningful to continue with the benchmarking process and extract the subset of classes which have been indicated as high-TD (or low-TD) classes by all tools. On the other hand, the graphical inspection of the aggregated results (Figure 7 (dot plots)) and the distributions of the coefficients for Java and JavaScript projects (Figure 8(a), (boxplots)) shows that the type of language seems to present an effect on the estimated agreement of TD tools. Indeed, an *independent-samples t-test* indicated a statistically significant difference between the mean values of Kendall's W concordance coefficient for Java ($M = 0.777$, $SD = 0.045$) and JavaScript ($M = 0.647$, $SD = 0.075$) projects, $t = 7.403$, $p < 0.001$ (Figure 8(b), (error bars)). *Levene's* test indicated unequal variances, $F = 7.628$, $p = 0.008$, so the t-test under the unequal variances assumption was used, whereas the *Kolmogorov-Smirnov* test for normality assumption showed that the estimated coefficients satisfied the normality assumption, K-S $Z = 0.893$, $p = 0.403$.

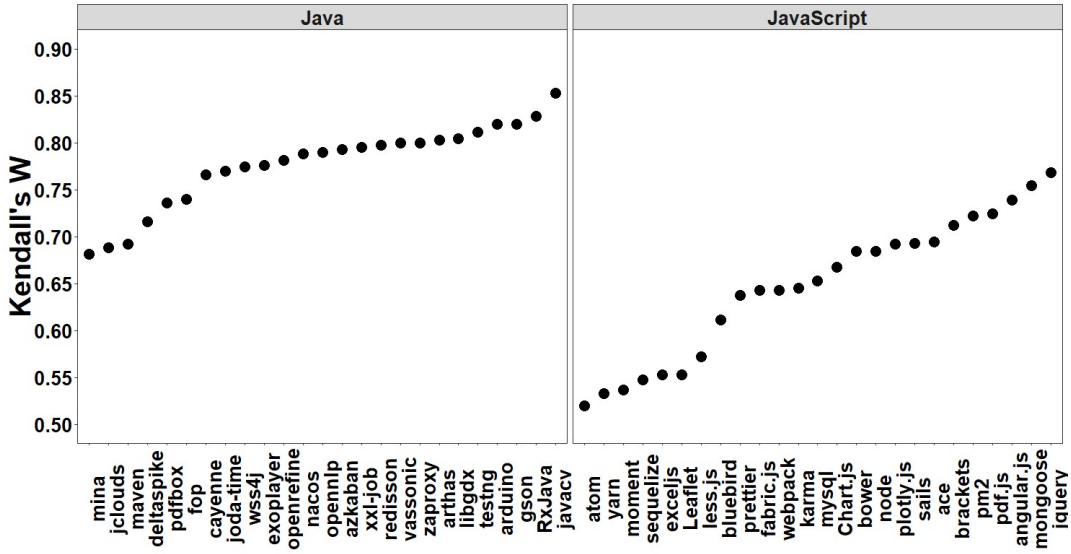
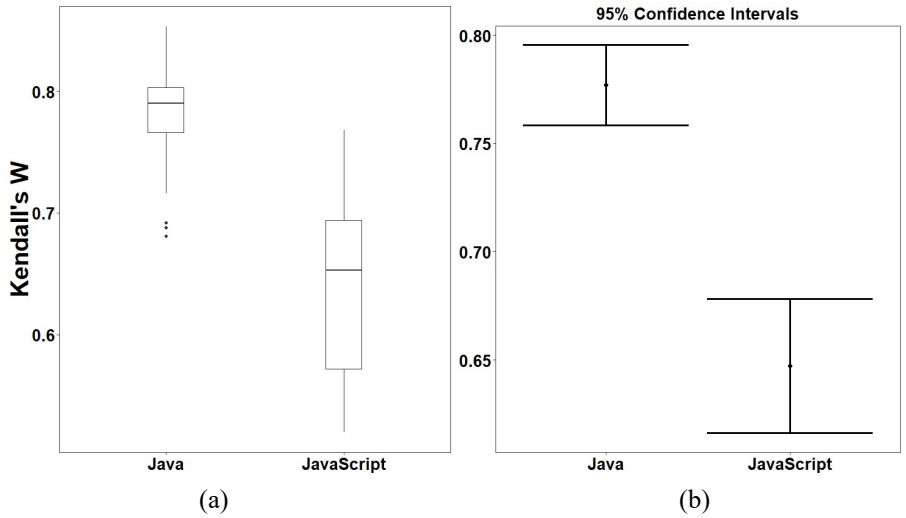


Fig. 7 Dot plots with the aggregated results of Kendall's W concordance coefficient



(a) (b)

Fig. 8 Box plots (a) and error bars (b) of the distributions of Kendall's W concordance coefficient

Table 6 Kendall's W Concordance Coefficient among all three TD tools for each analyzed system

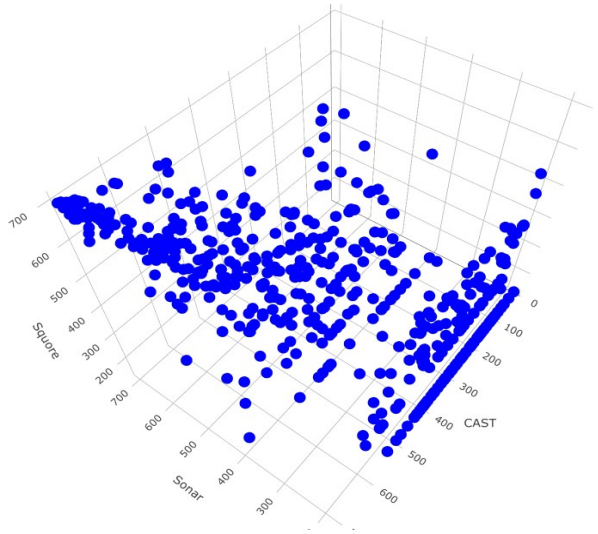
Project	W (p-value)	W (p-value)	W (p-value)	W (p-value)	W (p-value)	W (p-value)	W (p-value)	W (p-value)	W (p-value)
Java									
arduino	0.820 ($p < 0.001$)	exoplayer	0.776 ($p < 0.001$)	joda-time	0.770 ($p < 0.001$)	opennlp	0.790 ($p < 0.001$)	testng	0.811 ($p < 0.001$)
arthas	0.803 ($p < 0.001$)	fop	0.740 ($p < 0.001$)	Libgdx	0.804 ($p < 0.001$)	openrefine	0.781 ($p < 0.001$)	vassonic	0.800 ($p < 0.001$)
azkaban	0.793 ($p < 0.001$)	gson	0.820 ($p < 0.001$)	Maven	0.692 ($p < 0.001$)	pdfbox	0.736 ($p < 0.001$)	wss4j	0.774 ($p < 0.001$)
cayenne	0.766 ($p < 0.001$)	javacv	0.853 ($p < 0.001$)	Mina	0.681 ($p < 0.001$)	redisson	0.797 ($p < 0.001$)	xxl-job	0.795 ($p < 0.001$)
deltaspike	0.716 ($p < 0.001$)	jclouds	0.688 ($p < 0.001$)	Nacos	0.788 ($p < 0.001$)	RxJava	0.828 ($p < 0.001$)	zaproxy	0.800 ($p < 0.001$)
JavaScript									
ace	0.694 ($p < 0.001$)	brackets	0.712 ($p < 0.001$)	Karma	0.645 ($p < 0.001$)	mysql	0.653 ($p < 0.001$)	prettier	0.637 ($p < 0.001$)
angular.js	0.739 ($p < 0.001$)	Chart.js	0.667 ($p < 0.001$)	Leaflet	0.553 ($p < 0.001$)	node	0.684 ($p < 0.001$)	sails	0.693 ($p < 0.001$)
atom	0.520 ($p < 0.001$)	exceljs	0.553 ($p < 0.001$)	less.js	0.572 ($p < 0.001$)	pdf.js	0.724 ($p < 0.001$)	sequelize	0.547 ($p = 0.002$)
bluebird	0.611 ($p < 0.001$)	fabric.js	0.643 ($p < 0.001$)	moment	0.537 ($p < 0.001$)	plotly.js	0.692 ($p < 0.001$)	webpack	0.643 ($p < 0.001$)
bower	0.684 ($p < 0.001$)	jquery	0.768 ($p < 0.001$)	mongoose	0.754 ($p < 0.001$)	pm2	0.722 ($p < 0.001$)	yarn	0.533 ($p < 0.001$)

The general conclusion from the evaluation of the Kendall's W concordance coefficients and the rule of thumb proposed by Schmidt (1997) (see Section 3.4.1) is that in the case of Java projects, there is noted a statistically significant ($p < 0.001$) and strong agreement among the three tools regarding the TD assessments for the set of the conducted experiments with a mean value of 0.777 accompanied by a 95% CI ranging into the interval $[0.758, 0.795]$. In contrast, despite the fact that a statistically significant agreement among TD assessments is also indicated for the set of JavaScript projects, the strength of the agreement is characterized as moderate, since it presents a mean value of 0.647 with a 95% CI of $[0.616, 0.678]$. A possible interpretation for this finding is that tools for analyzing the quality of Java code (e.g. through static analysis) are more mature, compared to those for analyzing JavaScript, which are substantially younger. Therefore, it seems that along with their evolution Java analyzers have also converged on how the analysis is performed and what is deemed as an important problem for a codebase. On the other hand, it seems that JavaScript analyzers are in a more experimental stage, and therefore lower consensus is reached.

4.2 RQ2: How many archetypes (reference assessments) are required to capture the diversity of the tools?

After the verification of a statistically significant agreement among the three TD tools for the set of Java and JavaScript projects, the next challenge involves the benchmarking process with the aim to extract a set of classes identified as the most high-TD ones from all applied tools. Due to the extensive numerical and graphical results, we indicatively present the findings derived from the analysis (Step 1 - Step 3, see Section 3.4.2) on opennlp project. Through this manner, it can be also highlighted to both researchers and practitioners how the proposed methodology can be easily generalized to any experimental setup without constraints regarding the number of applied TD tools. Finally, we remind that the set of the experimental results along with the raw dataset of TD estimates for the 25 Java and 25 JavaScript projects are publicly available at Zenodo[22].

Generalizing the methodology presented above (Section 3.4.2), the relative positions of the TD assessments via the three tools can be represented by a scatter plot in a three-dimensional space (Step 1). Figure 9 displays the TD assessments, in which each point represents again, a specific class with coordinates the TD rankings evaluated by the SonarQube (x-axis), CAST (y-axis) and Squore (z-axis) tools. Despite the fact that drawing conclusions from the inspection of a three-dimensional plot is not a straightforward task, the shape of the swarm of points reveals an intrinsic pattern. More precisely, there is a subset of classes that are concentrated on the upper left corner of the plot, corresponding to classes that accumulate a high amount of TD as it is assessed by the whole set of the applied tools. On the other hand, it is also obvious that there are also other regions on the graph indicating divergent behaviour of the applied tools in terms of their TD assessments. The practical implication of this phenomenon is that the three TD tools signify different mitigation actions, which is the consequence of the utilization of different rulesets in the evaluation process of TD.



[22] https://doi.org/10.5281/zenodo.3966202

Fig. 9 Scatter plot (3D) for the rankings of the TD assessments (all three tools) (opennlp project)

Indeed, the examination of the RSS (Figure 10) after the consecutive executions of the AA algorithm for different values of archetypes shows that the convex hull of the swarm of points can be adequately approximated by $k = 8$ archetypes. Generally, the examination of the RSS plots for the remaining datasets led us to conclude that this specific number of archetypes $k = 8$ is a rational generalization for the whole set of our experiments.

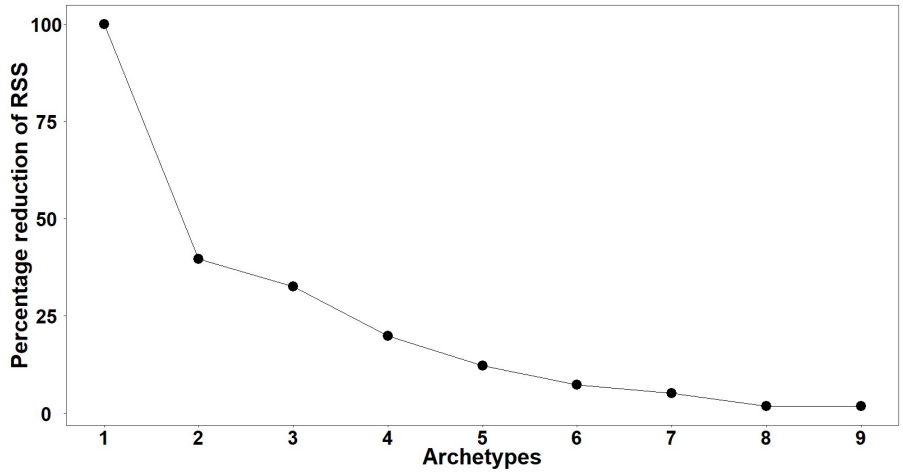


Fig. 10 RSS plot (SonarQube, CAST, Squore) (opennnlp project)

4.3 RQ3: Which are the characteristics of the extracted archetypes?

Having defined the appropriate number of archetypes ($k = 8$), the next step (Step 2) of the proposed approach concerns the reification of the extracted reference assessment profiles through the examination of their characteristics. Figure 11 summarizes the profile plots for each archetype of the derived solution. The examination of the characteristics of the eight profiles reveals, again, that there are two distinct groups (*Ruler* and *Rebel*) that have also been identified in the case of the TD assessments on the two-dimensional space (CAST and Squore) (see Section 3.4.2). Besides this fact, the analysis brings to the surface a new type of profile with specific characteristics regarding the assessments of the three tools. More specifically, the *Partner*[23] archetype represents a reference assessment profile, in which two of the applied tools indicate a high amount of TD, whereas on the same time, the third tool is not able to identify it indicating a low amount of TD.

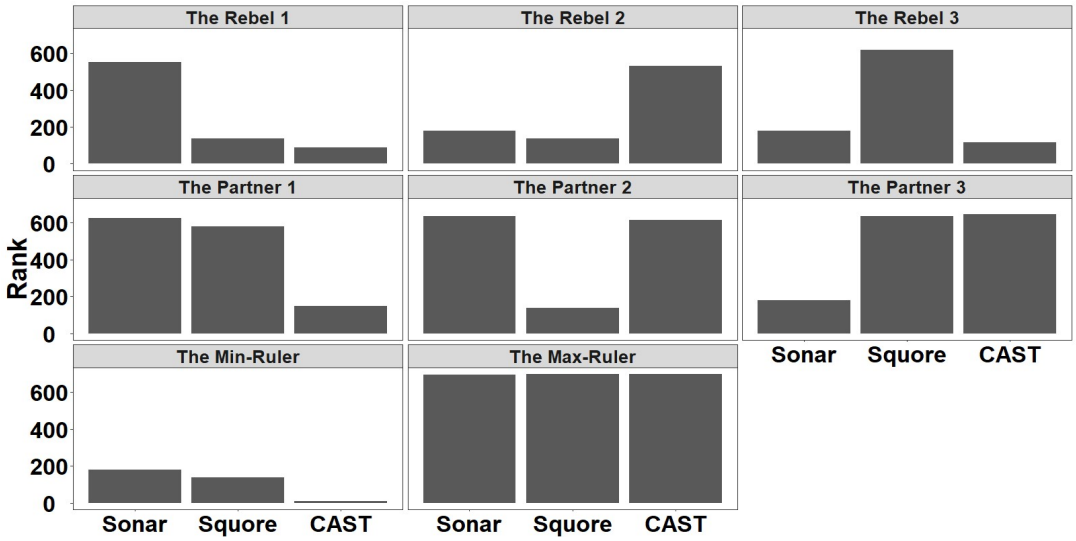


Fig. 11 Reference assessment profiles (archetypes) from the assessments by all three tools (opennnlp project)

[23] The Partner archetype refers to personalities whose goal is being in a relationship with people and surroundings. In analogy, the Partner profile in our case denotes cases where two of the three tools exhibit high agreement.

The characteristics of the $k = 8$ reference assessments (Figure 11) are fully described below:

- **The Max-Ruler** is the type of the reference assessment indicating a high amount of TD based on the results of all applied tools (SonarQube, CAST, Squore).
- **The Min-Ruler** is the type of the reference assessment indicating a low amount of TD based on the results of all applied tools (SonarQube, CAST, Squore).
- **The Partner 1** is the type of the reference assessment indicating a high amount of TD based on the results from SonarQube and Squore tools and simultaneously, a low amount of TD based on the results of CAST tool.
- **The Partner 2** is the type of the reference assessment indicating a high amount of TD based on the results from SonarQube and CAST tools and simultaneously, a low amount of TD based on the results of Squore tool.
- **The Partner 3** is the type of the reference assessment indicating a high amount of TD based on the results from Squore and CAST and tools and simultaneously, a low amount of TD based on the results of Sonar tool.
- **The Rebel 1** is the type of the reference assessment indicating a high amount of TD based on the results from SonarQube tool and simultaneously, a low amount of TD based on the results of Squore and CAST tools.
- **The Rebel 2** is the type of the reference assessment indicating a high amount of TD based on the results from CAST tool and simultaneously, a low amount of TD based on the results of SonarQube and Squore tools.
- **The Rebel 3** is the type of the reference assessment indicating a high amount of TD based on the results from Squore tool and simultaneously, a low amount of TD based on the results of SonarQube and CAST tools.

An interesting conclusion of the analysis on the remaining forty-nine datasets is that the abovementioned types of archetypes are applicable for the entire spectrum of projects and classes. It is reasonable to assume that the identified types of archetypes would be valid for any number of employed tools. For example, there will always be some classes identified as having high TD (or low TD) by all assessors (conforming to the Max-Ruler or the Min-Ruler archetype). Nevertheless, the number of commonly identified high-TD (or low-TD) classes is expected to decrease with the number of tools. Similarly, it is also highly probable that one of the employed tools will tag some classes as high-TD while all other tools will not, according to the Rebel archetype, or that some subsets of tools might agree to a larger extent (Partners). This inherent trade-off should be considered by development teams when opting for particular quality assurance tools. The 'intersection' of commonly agreed artefacts with TD principal is expected to become lower as the number of tools increases and the benefit of obtaining wider coverage should be weighed against the diversity of the findings and the difficulties in incorporating multiple tools in the workflow. Practitioners and researchers should be assisted in focusing on the modules that are most likely to suffer from TD and to this end the next RQ aims at selecting the right set of classes for further analysis.

4.4 RQ4: Which classes should be selected to form an agreement-based benchmark of top-TD modules?

In the last step of the methodology (Step 3), the focus is now on the identification of classes that are close to the archetype signifying top-TD classes as assessed by all tools. Practically, we seek for classes settled in the neighborhood of the Max-Ruler archetype, which in turn can be specified through the definition of a threshold value for a coefficient. For example, in project opennlp, if we set $a = 0.80$ as a threshold value to capture a strong similarity (or adjacency) to the Max-Ruler archetype (in analogy to the 2-tool representative) example presented in Section 3.4.2), for three tools we would obtain 54 top-rated TD classes (7.70% of the total), while for two tools we obtained 84 top-rated TD classes (11.98%). The decrease in the number of commonly identified high-TD classes confirms the observation that the higher the number of assessors, the smaller the number of top-rated classes pointed out by all tools.

To examine the effect of the defined threshold value a on the percentage of top-rated classes extracted by the proposed approach, based on the source code analysis via the set of selected TD tools, we conducted sensitivity analysis. More precisely, we evaluated the percentage of top-rated classes for a set of threshold values of a - coefficients ranging from 0.60 to 0.90 increasing by a step of 0.05. In addition, there is an imperative need to investigate whether the type of language presents an effect on the percentages of top-rated classes for the above set of threshold values, since the inter-rater agreement analysis presented in Section 4.1 revealed a statistically

significant effect of the type of language on the estimated concordance coefficients. Thus, an interesting issue that deserves further investigation is whether the type of language also affects the percentages of the top-rated classes.

Figure 12 summarizes the results from which, we can generally infer that the percentage of top-rated classes decreases as the threshold value increases for both language types. Practically, the selection of a higher threshold value imposes a stricter policy for the identification of high-TD classes by all employed tools. Another interesting finding is the fact that the percentages of top-rated classes/files seems to be generally higher for Java projects in comparison to JavaScript projects.

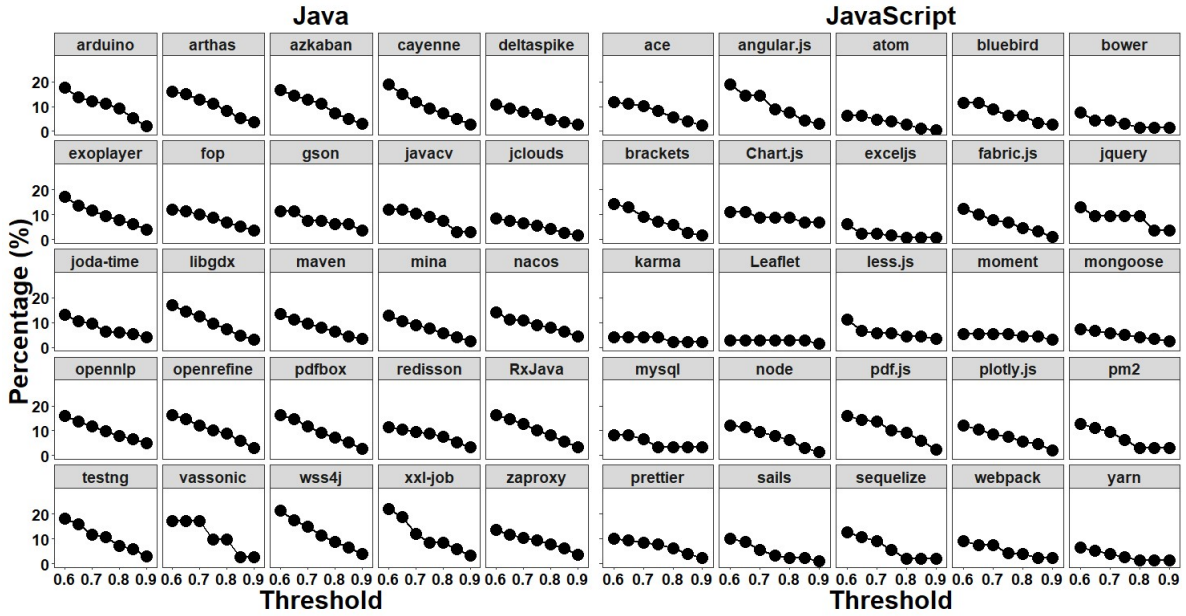


Fig. 12 Percentage of top-rated classes assessed by all three tools for increasing levels of threshold values a (sensitivity analysis)

So, the next issue is to investigate, whether the observed phenomena can be generalized to the population of OSS projects with similar characteristics. For this reason, we use the *Linear Mixed Effects* (LME) models (Pinheiro and Bates, 2000) that are able to model simultaneously two types of effects that are (*i*) the fixed effects, a term that is used to represent factors that may affect the mean value of interest, and (*ii*) the random effects that may have an impact only the variance of the response variable.

In our experimental setup, the *experimental unit* for which, we wish to draw conclusions regarding the response variable *Percentage* (i.e. the percentage of top-rated classes) is the project, which in fact, represents a unit drawn at random from an infinite unknown population of projects. For this reason, one should take into account and incorporate into the analysis, the random effect of the factor *Project*, in order to model the inherent variability caused by this random selection from the set of all possible OSS projects. Regarding the fixed effects that can been thought as the effect of specific factors of interest on the response *Percentage*, we have to examine two factors that are (*i*) the threshold value (*Threshold*) of a denoting the closeness to the Max-Ruler archetype and (*ii*) the type of language (*Language*). Besides the abovementioned two *main effects* (*Threshold*, *Language*), there is also a need to examine the *interaction effect* of Threshold and Language (*Threshold*× *Language*), since the effect of the threshold value of a on the percentage of top-rated classes may not be the same at the two levels of language types (Java/JavaScript).

Regarding the fixed component structure, which describes the main and interaction terms that will be included in the inferential process, the optimal structure was defined through the protocol proposed by Zuur et al. (Zuur et al., 2009). Described briefly, a model (defined as the *beyond model*) examining all factors of interest and their possible interactions is fitted and tested against a second model after omitting the higher order interaction term through the *Likelihood Ratio* (LR) test. In case of an insignificant finding, the selection is based on the principle of parsimony, which practically means that simpler models with similar explanatory power are preferred over more complex models with more parameters but slightly better fit. To this end, the *Akaike Information Criterion* (AIC) is used for the comparison process, while the model with the lowest AIC value should be preferred over the competitive ones.

The comparison of the beyond model (mentioned above) incorporating the main effects of *Threshold* and *Language* and their interaction term *Threshold*× *Language* against the model without the interaction term *Threshold*× *Language* did not reveal a statistically significant difference $\chi^2 = 6.055, p = 0.417$. The practical implication of this result is that the effect of the threshold value of α on the percentage of top-rated classes is the same for both language types (Java/JavaScript). The fitting of the final LME model containing only the main effects revealed statistically significant main effects for both *Threshold* ($F = 299.634, p < 0.001$) and type of *Language* ($F = 29.493, p < 0.001$) on the mean percentage values of top-rated classes. We have also to note that all models were fitted on the logarithmic transformations of the raw percentages, due to the violation of homoscedasticity assumption of model's residuals.

Moreover, the post-hoc analysis through Tukey's HSD test (Pinheiro and Bates, 2000) for the factor *Threshold* indicates statistically significant differences ($p < 0.05$) between the pairs of consecutive levels of threshold values (as shown in Figure 13, the error bar does not cross the vertical dashed line of zero). Finally, in Table 7, we report the expected mean percentage (accompanied by 95% CI) of top-rated classes for both language types in the population of OSS projects with similar characteristics in order to provide an indication of how many classes will be assessed as top-rated by all applied tools.

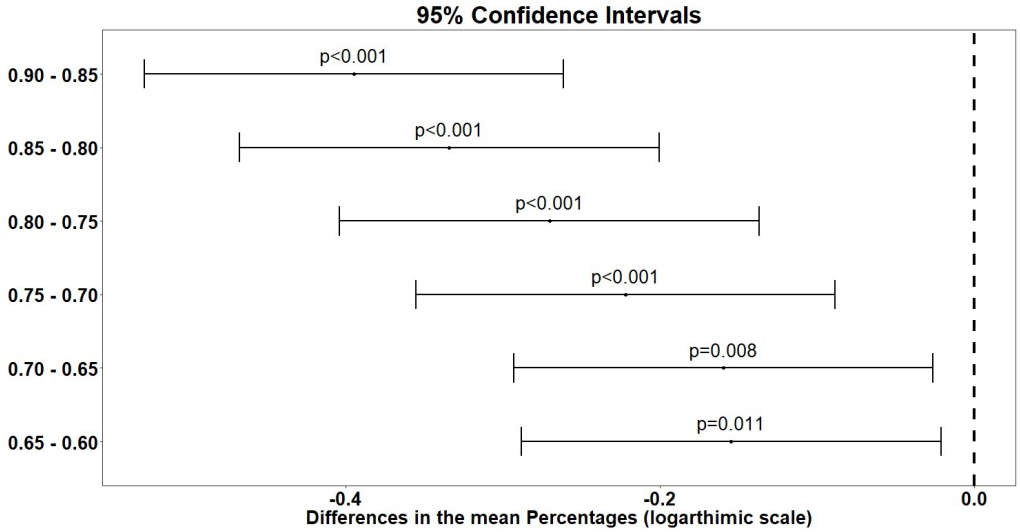


Fig. 13 Post-hoc analysis for LME model (sensitivity analysis)

Table 7 Estimated mean percentage with 95% CI for each threshold value α (sensitivity analysis)

	Java		JavaScript	
Threshold	**Estimation**	**95 % CI**	**Estimation**	**95 % CI**
0.60	15.24	[13.17, 17.64]	9.11	[7.87, 10.55]
0.65	13.06	[11.28, 15.12]	7.81	[6.75, 9.04]
0.70	11.13	[9.62, 12.89]	6.65	[5.75, 7.70]
0.75	8.92	[7.71, 10.32]	5.33	[4.61, 6.17]
0.80	6.81	[5.88, 7.88]	4.07	[3.52, 4.71]
0.85	4.87	[4.21, 5.64]	2.91	[2.52, 3.37]
0.90	3.28	[2.84, 3.80]	1.96	[1.70, 2.27]

As it can be observed from Table 7, out of the total population of classes in each project and depending on the threshold value, only a small portion lying into the intervals [3.28%, 15.24%] and [1.96%, 9.11%] for Java and JavaScript projects, respectively, is characterized as having high-TD based on the findings of all three tools. Generally speaking, and without taking into consideration the type of language, this relatively low number of classes, in the neighborhood of the Max-Ruler archetype can be acknowledged as a basis concerning the high-TD classes. The resulting agreement-based benchmark can drive further research by denoting the few modules carrying "real-TD", rather than dealing with all candidates extracted by a single tool, which are not confirmed by other tools. Any future approach, leveraging also the power of machine learning, could be trained to accu-

rately identify the top-rated classes capturing TD in a more realistic manner. It should be noted that a similar methodology could be applied for extracting a benchmark of low-TD classes. Such a set of classes might be valuable for studying the principles and practices resulting in cleaner code. Nevertheless, given the current priorities of development teams and researchers we have focused on benchmarks of high-TD classes. Besides the abovementioned findings, the analysis also indicates that irrespective to the applied threshold value a for characterizing similar classes to the Max-Ruler archetype, the percentages of the high-TD classes are expected to be higher for Java projects compared to the corresponding percentages derived from the analysis on JavaScript projects. In the Appendix, we conduct a sensitivity analysis in order to examine the variability of the classes belonging to the Max-Ruler archetype.

5. Implications to Practitioners and Researchers

In this section, we revisit the main outcomes of this study from the perspectives of practitioners and researchers. However, it should be borne in mind that any identified implications are subject to the limitations of the context in which our study has been performed. In particular, the current study has considered only the types of TD identified by the selected tools (namely design and code debt) and two programming languages (namely Java and JavaScript). Moreover, the findings are based on a single measure of TD (i.e. principal) excluding other indices such as the severity or the type of the identified inefficiencies.

Overcome construct validity threats in research (researchers). As mentioned in the introductory section, the research community within the TD field lacks an ultimate process to accurately capture TD principal and thus, any empirical study or technique based on TD estimates runs the risk of not accurately measuring the real-world phenomenon under study. Each tool follows its own approach for detecting and measuring TD, based on a distinct ruleset, yielding a different amount for the total TD, but also pointing to different parts of the code that need to be mitigated, compared to other tools. There are several studies trying to identify high-TD modules and studies investigating the association of accrued TD with other factors. However, such approaches are heavily dependent on the employed tool for suggesting the ground truth, that is, the modules that actually have TD liabilities and need to be fixed. Apparently, because each tool evaluates TD in a different way, the generalizability of these approaches is threatened to a large extent.

The two aspects of the proposed methodology, that is, the estimation of inter-rater agreement among TD tools and the use of archetypal analysis for identifying classes having a desired profile (e.g. high-TD levels by all tools) can be applied by researchers to form a more reliable basis for their experiments. More conveniently, researchers can also employ the already available benchmarks of high-TD classes (but also classes having a different profile if needed) from the online TD Benchmarker web application. Consequently, leveraging the power of multiple TD tools using the proposed approach can assist in the mitigation of construct validity threats that is currently present in the field of TD.

Highlight critical modules with validated highest TD (practitioners). Despite the widespread adoption of the TD metaphor, it is far from clear which tool IT managers should integrate in the development and maintenance process. Employing more than one TD tool for the evaluation of their software might be a costly option, since most of the existing tools are available only with a commercial license. Moreover, each tool requires significant effort to deploy, properly configure and familiarize with. However, even if a development team employs more than one tool, the union of all findings, would result in an unrealistic amount of suggestions, rendering the process intractable. Based on the proposed methodology, practitioners can highlight the classes that have been identified as high-TD classes by all employed tools leading to a manageable number of target classes. Development teams can take advantage of such agreement-based benchmark sets and focus only on the modules of their system that are validated as high-TD modules. With respect to the benefit of the already derived benchmarks from the analyzed systems, developers can focus on the classes close to the Max-Ruler archetype and gain insight into the root causes of the accumulation of TD in these classes and potentially avoid non-ideal coding practices in the future. Moreover, the non-unanimous archetypes (Rebel and Partner archetypes) can be valuable, as well. The existence of these archetypes is the key factor that differentiates one tool from the others. If only unanimous archetypes existed, this would mean that all tools generate the same results pointing to the same classes/files with accrued TD principal. Through the exploration of classes/files in the vicinity of non-unanimous archetypes development teams can gain insight into how TD tools differ on the measurement and prioritization of TD principal. With such knowledge, developers can more confidently invest in the TD tool that best fits their perception of when a class/file is tagged as high-TD (or low-TD).

Collection of available TD tools (researchers and practitioners). Last but not least, another contribution of our work is the localization and collection of available TD assessment tools, as presented in Section 2. The list is by no means an exhaustive one, as numerous other tools offer functionality related to the identification of code smells, anti-patterns, rule violations, excessive metric values, etc. all of which are indicators of the existence of TD in software. Nevertheless, the presented tools can serve as starting point both for practitioners who are searching for a TD tool to integrate into their development process as well as researchers who are seeking an appropriate assessor of TD principal. In both cases, the proposed methodology can assist in the critical appraisal of the agreement or the diversity among tool findings.

6 Threats to Validity

In this section, we present and discuss potential threats to the validity of our case study, focusing on construct, reliability, and external validity (Runeson and Höst, 2008; Wohlin et al., 2000). Internal validity is not considered, since causal relations have not been studied.

Construct Validity. Concerning construct validity, it can be argued that the basis of TD cannot be formed solely on the findings of TD assessment tools and as a result the study might inaccurately capture the actual phenomenon. The employed tools perform static source code analysis and thus the identified liabilities are primarily related to code TD, and in certain cases might also point to design or architectural problems. But according to the literature (Alves et al., 2016; Li et al., 2015) several other types of TD have been identified and might be present throughout all phases of the software development lifecycle, including Test, Documentation, Build, Infrastructure TD, etc. Consequently, the extracted TD measurements and the resulting benchmark represent only a portion of the system TD. However, code TD has been one of the mostly studied type of TD (Li et al., 2015) and the target of most available tools, including the ones that have not been used in this study. Furthermore, the steps of the proposed methodology are equally applicable to the findings regarding any type of TD and thus benchmarks can be derived for other types of problems, provided that suitable measurement tools are available.

Another important threat to construct validity pertains to the exclusion from the study of other TD-related information, such as the specific type of the identified inefficiencies or their severity. Indeed, it might be the case that the level of agreement among tools varies depending on type/severity of issues and we believe that this warrants a further study. Although TD principal is an aggregate measure encompassing all kinds of identified problems, development teams would be more assured in case different tools agree on the more severe problems or the type of problems which they consider relevant to their software. Nevertheless, both aspects of the proposed approach for the quantification of the level of agreement among the tools and the extraction of representative archetypes can be applied to any subset of the identified TD issues.

Reliability. The described methodology outlines all steps followed to carry out the inter-rater agreement and archetypal analysis along with the provided web application that allows the extraction of benchmarks (sets of classes close to the Max-Ruler archetype) mitigates reliability threats. One potential threat to the ability of replicating this study and reaching the same results is related to the optimal number of archetypes defined in Step 1 of the proposed approach (Section 3.4.2). The selection of the appropriate number of archetypes that is able to capture the diversity of the examined TD tools based on the inspection of the multidimensional space is, to some extent, a subjective process, especially in the case of a three-dimensional plot. In addition, the above visualization practice is not applicable in case the number of tools is higher than three. In these cases, the practitioner should base his/her choice on the examination of the profile plots and most importantly, on the inspection of the RSS plot to conclude on the appropriate number of archetypes. In our experimental setup, the investigation of these graphical manners led us to the definition of the optimal number of eight archetypes, which is a rational and common-sense finding, since the derived archetypes represent expected behaviors, in cases where three TD tools with partially different rulesets are used for benchmarking purposes. Finally, the Open Science Replication Package (TD Benchmarker source code[24] and AA tool support with dataset[25]) are validated by the Open Science Board. The links to these resources are provided as footnotes in this manuscript.

[24] https://github.com/theoam/TDBenchmarker
[25] https://doi.org/10.5281/zenodo.3966202

External Validity. Regarding the external validity of the proposed approach, a potential threat to the generalization of the results is related to the identification and retrieval of the set of classes that are close to the Max-Ruler archetype (Step 3, Section 3.4.2), since the extracted set is certainly affected by the subjectivity and strictness of the practitioner. To this regard, we conducted a sensitivity analysis in order to examine how the choice of the threshold value for a -coefficient defining the neighbour classes affects the percentage of classes that belongs to the extracted benchmark set. Moreover, this work investigates the research questions in the context of 50 open source projects. Due to the limited number and types of the analyzed systems the conclusions regarding the observed level of agreement among the tools and the number of archetypes which are sufficient to capture the swarm of the observed points, probably cannot be generalized across other domains, programming languages or to proprietary software. A similar threat to external validity stems from the selection of TD assessment tools in the sense that our analysis was based on the identified violations, which in turn reflect the particular ruleset of each tool. Therefore, the findings on the agreement of TD assessment tools cannot be generalized beyond the employed tools.

7. Related Work

Since our first goal was to study the level of agreement among TD tools, in this section we present previous studies that compare the techniques and results of tools that explicitly or implicitly measure TD. Our second goal was to extract an agreement-based benchmark set of validated high-TD classes; therefore, we discuss other approaches to build such benchmarks or extract thresholds in the broader area of software maintenance.

7.1 Comparison of Tools measuring Technical Debt

In a previous case study (Kazman et al., 2015), the authors aimed at locating the architecture debts of a proprietary web portal system owned by a software outsourcing company using their own tool, *Titan*. The results of the Titan tool (TitanDebts) were compared to the results of the SonarQube tool (SonarDebts) that the company was already using. By examining the overlap between TitanDebts and SonarDebts, the authors found that ¼ of the total files (25 files) were found in the intersection of the most problematic files that Titan and SonarQube have identified. To this regard, the authors concluded that the Titan tool (which identifies architecture debts more effectively) and the SonarQube tool detect substantially different and complementary sets of files.

A case study in 2014 (Zazworka et al., 2014) compared four different techniques of TD evaluation (with the associated tool to run the analysis) including code smells (tool: codevizard), automatic static analysis issues (tool: FindBugs), grime build up and modularity violations (tool CLIO). The authors investigated whether the set of selected techniques/tools report the same set of modules as problematic and which was the overlap among them. The classes of 13 Hadoop releases were measured and 30 metrics were compared. The results of the study showed that the four techniques/tools had very little overlap, pointing to different problems in different modules.

In an experimental study (Griffith et al., 2014), the authors investigated the correspondence between several technical debt estimation approaches and external software quality models. Specifically, they evaluated (a) SonarQubes's, (b) CAST's and (c) Marinescu's method (Marinescu, 2012) of technical debt estimation against the QMOOD quality model, which encompasses the quality attributes; reusability, flexibility, understandability, functionality, extendibility and effectiveness. They did not find evidence for strong relationship between the TD estimates and the quality attributes of the QMOOD model, except for one estimation method regarding only the flexibility and effectiveness quality attributes. The authors concluded that "*it is important that industry practitioners, ensure that the technical debt estimate they employ accurately depicts the effects of technical debt as viewed from their quality model*".

In a recent study (Ernst et al., 2017), the authors, being motivated by the perception that design problems are more significant than coding errors for long-term software maintenance, aimed at investigating how three major TD tools (CAST, NDepend, SonarQube) capture design debt. Particularly, the authors distinguished the rules that capture design debt from a total of 466 examined rules from all three tools. Their results showed that all three tools mainly focus on non-design debt (only 19% of the rules captured design issues). Particularly, NDepend focuses the most on design rules (26% of its total rules are design-related), then follows CAST with 17% and SonarQube with 13%.

Fontana et al. (2016) examined the impact of the elimination of architectural problems in four Java projects on the quality indices of four tools (SonarQube, inFusion, Structural Analysis of Java (SA4J) and Structure101). The results showed that the architectural refactorings in the four examined systems did not have any impact on the SQALE index of SonarQube and as far as SA4J is concerned, its stability index was affected only in one system. Consequently, the authors concluded that the SQALE index of SonarQube and the stability index of SA4J are not capable of effectively capturing the notion of architectural debt.

In another study (F. A. Fontana et al., 2016), the authors compared the techniques of five tools (CAST, inFusion, Sonargraph, SonarQube and Structure101) that provide some kind of Technical Debt Index (TDI). The comparison of the tools showed that all tools except for SonarQube exploit architectural information to form their TDIs. Moreover, two of the tools (inFusion and Structure101) do not calculate the cost for TD remediation (TD principal) whilst they only calculate the cost of keeping the software as it is (TD interest). On the other hand, CAST and SonarQube calculate only TD Principal and not TD interest. As far as the output measurement, CAST and Sonargraph output cost in terms of US dollars, SonarQube in terms of time to remedy issues, while the rest produce either abstract values or values that are not expressed in money or time.

According to the abovementioned studies that compared different TD measurement tools, the results of each tool diverged from the results of the others. This phenomenon emphasizes our motivation to compare the TD estimates of several TD tools and extract the high-TD modules as identified by the tools altogether. It should be also noted that the aforementioned studies employed tools that measure TD either explicitly (generating a direct Technical Debt Index) or implicitly (generating a general quality index). Nevertheless, we remind that, in our study, we employed tools that explicitly output a Technical Debt Index to allow for more focused and direct comparison of the results on TD measurement.

7.2 Benchmarks in Software Maintenance

Several studies attempt to establish benchmark datasets so that software quality assessment approaches can be compared against them. Quite often the related research effort aims at building benchmarks to extract representative thresholds for source code metrics or quality indices, which can then serve as baseline for comparison with actual values of the systems under evaluation. A notable example of such benchmarks is the benchmark repository of Software Improvement Group (SIG) against which any selected system can be compared in terms of code quality and maintainability (Baggen et al., 2012). Below we provide an overview of studies, in which the authors developed benchmarks and aimed at deriving thresholds for the evaluation of software quality (in descending chronological order).

In a recent study (Mori et al., 2018), the authors defended the idea that the extraction of metric thresholds should be tailored to each software domain. They collected a large set of 3107 Java systems across 15 domains from GitHub[26] and measured a set of 8 source code metrics with the CK Tool[27]. The aforementioned metrics reflected size, complexity and inheritance aspects of software. Then, the authors derived metric thresholds using the method supported by TDTool (Veado et al., 2016). In particular, thresholds have been selected so as to represent various groups (i.e. high-90% and very high-95%) of the sorted metric values. The authors found evidence that "*metric thresholds vary across domains and most domain-specific thresholds differ from generic thresholds*".

Döhmen et al. (2016) built a benchmark for maintainability evolution with data from approximately 1750 industrial software systems. The data was collected from the Software Analysis Warehouse (SAW), a property of the Software Improvement Group (SIG). SAW contains the results of the software quality analyses that SIG conducts. The study focused on the production source code of the projects excluding testing and auto-generated code. The authors created a prototype of a benchmark for maintainability evolution. The benchmark was based on a group of systems, which were close to a selected open source system, *Crawljax*, in terms of maintainability and volume. The authors, first, selected the systems which had the 5% closest maintainability transitions to Crawljax and then, with the use of Empirical Cumulative Distribution Function (ECDF) found the systems that developed equal or worse than the compared system.

[26] https://github.com/
[27] https://github.com/mauricioaniche/ck

Comparison against existing systems has also been used as a method for assessing the software quality of a commercial system, property of an international company in the logistics domain (Yamashita, 2015). The system was analyzed in terms of size, complexity, modularity, redundancy and technical debt with the utilization of SonarQube and NDepend. To evaluate the quality of the system, the author compared it with the quality of a set of 1892 open source projects from GitHub of similar age and programming language. The author calculated the metrics of each project with SonarQube and then extracted the percentile thresholds of the metrics with RTTool (Oliveira et al., 2014a). The system's metric was considered "normal" if its value was near the middle percentiles and vice versa. The aforementioned benchmark was applied at file and at system level with aggregated values.

The notion of balance between real and ideal software design was used in a study in 2014 (Oliveira et al., 2014b), in which the authors described a method for deriving relative thresholds for source code metrics. The method was based on evidence that source code metrics follow fat-tailed distributions, meaning that there is no typical value for them (Ferreira et al., 2012). Therefore, the authors suggested that it is acceptable for some metrics not to follow absolute thresholds. To this regard, they proposed the concept of relative thresholds for evaluating source code metrics, where a percentage of source code entities should have values lower than an upper limit, whilst another percentage of entities is accepted to exceed upper limit due to specific requirements. The method was evaluated by applying it on the classes of 106 Java systems and extracting thresholds for seven metrics.

A benchmark-oriented calculation of TD was proposed by Mayr et al. (2014). Their benchmark-based model for calculation of Remediation Costs of software combined features from three existing TD calculation approaches; CAST model, SQALE model and the SIG model. Measures obtained with these models were normalized in terms of lines of code before used in the proposed model. For each metric, the authors calculated a quartile-based distribution dividing the normalized values of the metric in four areas. Metrics with values that laid below the lower or above the upper areas were considered non-conforming to the benchmark dataset. Ultimately, the authors tested their model by applying it on two open source projects, the quality of which had been previously evaluated and compared against the benchmark database. The experiment showed that the model was able to calculate remediation costs that reflected the relative (to the benchmark database) quality of the projects.

In another study (Alves et al., 2010), a method for extracting metric thresholds from benchmark data was designed. The method was applied on a benchmark of 100 C# and Java systems proprietary and open-source from a broad range of domains. The metrics were extracted for every entity of the system (method and file level) and were normalized with the weight of the entity. As weight of the entity, its size in terms of LOC was considered. Then the normalized metrics were placed in percentiles, from which the thresholds derived. Their contribution to the industry was to successfully use the thresholds derived with their methodology instead of the thresholds based on experts' opinion.

8. Conclusions and Future Work

The Technical Debt metaphor successfully captures, in monetary terms, the penalty that has to be paid because of shortcuts during software development. These shortcuts are known to introduce architectural, design and code inefficiencies in software systems and various TD tools aim at identifying them by testing the source code against specific rulesets. However, TD tools provide different estimates of TD principal pointing to different mitigation actions. These discrepancies make a lot of people in academia and practice skeptical about the validity of existing TD tools and hinder the further development of TD research as no ground truth for accurate TD instances can be established.

To address these limitations in the TD community we performed an empirical study whose goal was twofold: (a) to determine the level of agreement among three well-known TD tools and (b) build agreement-based benchmarks of high-TD classes/files from a dataset resulting from 50 open-source projects. Inter-rater agreement has been assessed, using Kendall's W coefficient of concordance. To capture the diversity of the examined tools with the aim of identifying representative class profiles we relied on archetypal analysis. Once the derived reference assessments are characterized, it is straightforward to extract sets of classes exhibiting similarity to a selected profile (e.g. that of high TD levels in all employed tools) and in this way establish a basis.

The findings of the inter-rater agreement analysis suggest that there is a statistically significant and strong agreement among the three TD tools on the measurement of TD at class level. However, a substantial degree of disagreement has also been observed for the measured TD level for numerous classes. The application of the archetypal analysis revealed that three types of reference assessments can successfully capture the spectrum of TD measurements provided by three tools: One set of archetypes represents classes identified as high-TD modules by only one of the tools, the second profile encompasses classes for which two of the tools agree on the measured TD level, while the final type of archetype signifies a high amount (or low amount) of TD based on the results of all applied tools. Selecting the classes in the vicinity of the latter archetype yields an agreement-based benchmark of classes tagged as high-TD by all tools. Such benchmarks, beyond their value as fields of study for poor development practices that led to low quality classes, can potentially form the basis for training more sophisticated TD identification and measurement approaches.

The goal of this study was to shed light into the level of agreement among TD tools and to establish a process for deriving an agreement-based benchmark set of high/low TD artifacts. Any interpretation of the results considering different perspectives, such as development context, role of developers (tester, designer, analyzer, etc.) was beyond the scope of this paper. Nevertheless, this forms a really interesting area of future work. Another interesting line of research would be to investigate to which degree TD tools are compliant with the guidelines of the OMG Specification on Automated Technical Debt Measure[28], accompanied by an experience report on how to use these tools, problems that practitioners might face during their installation, configuration, and analysis, as well as a guide on how the TD Benchmarker can be used.

We also acknowledge that the nature of the examined rules by each tool might be a decisive factor for the TD principal estimates per class/file. Drilling down to the level of individual rule violations which are detected by each tool, can shed light into the cause of their agreement or discrepancy. We plan to conduct such a study to investigate the similarity among the examined rules by mapping the rules adopted by each tool to the rules employed by the other tools.

Acknowledgement

This research is funded by the University of Macedonia Research Committee as part of the "Principal Research 2019" funding program.

References

Alves, N.S.R., Mendes, T.S., de Mendonça, M.G., Spínola, R.O., Shull, F., Seaman, C., 2016. Identification and management of technical debt: A systematic mapping study. Inf. Softw. Technol. 70, 100–121. https://doi.org/10.1016/j.infsof.2015.10.008

Alves, T.L., Ypma, C., Visser, J., 2010. Deriving metric thresholds from benchmark data, in: 2010 IEEE International Conference on Software Maintenance. Presented at the 2010 IEEE International Conference on Software Maintenance, pp. 1–10. https://doi.org/10.1109/ICSM.2010.5609747

Arvedahl, S., 2018. Introducing Debtgrep, a Tool for Fighting Technical Debt in Base Station Software, in: Proceedings of the 2018 International Conference on Technical Debt, TechDebt '18. ACM, New York, NY, USA, pp. 51–52. https://doi.org/10.1145/3194164.3194183

Baggen, R., Correia, J.P., Schill, K., Visser, J., 2012. Standardized code quality benchmarking for improving software maintainability. Softw. Qual. J. 20, 287–307. https://doi.org/10.1007/s11219-011-9144-9

Baldassari, B., 2013. SQuORE: a new approach to software project assessment.

Campbell, G.A., Papapetrou, P.P., 2013. SonarQube in Action, 1st ed. Manning Publications Co., Greenwich, CT, USA.

Canhasi, E., Kononenko, I., 2014. Weighted archetypal analysis of the multi-element graph for query-focused multi-document summarization. Expert Syst. Appl. 41, 535–543. https://doi.org/10.1016/j.eswa.2013.07.079

Chan, B.H.P., Mitchell, D.A., Cram, L.E., 2003. Archetypal analysis of galaxy spectra. Mon. Not. R. Astron. Soc. 338, 790–795. https://doi.org/10.1046/j.1365-8711.2003.06099.x

[28] https://www.omg.org/spec/ATDM/About-ATDM

Chopra, K., Sachdeva, M., 2015. EVALUATION OF SOFTWARE METRICS FOR SOFTWARE PROJECTS. Int. J. Comput. Technol. 14, 5845–5853. https://doi.org/10.24297/ijct.v14i6.1915

Cohen, J., 1968. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. Psychol. Bull. 70, 213–220. https://doi.org/10.1037/h0026256

Cohen, J., 1960. A coefficient of agreement for nominal scales. Educ. Psychol. Meas. 20, 37–46. https://doi.org/10.1177/001316446002000104

Conejero, J.M., Rodríguez-Echeverría, R., Hernández, J., Clemente, P.J., Ortiz-Caraballo, C., Jurado, E., Sánchez-Figueroa, F., 2018. Early evaluation of technical debt impact on maintainability. J. Syst. Softw. 142, 92–114. https://doi.org/10.1016/j.jss.2018.04.035

Cunningham, W., 1992. The WyCash Portfolio Management System, in: Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum), OOPSLA '92. ACM, New York, NY, USA, pp. 29–30. https://doi.org/10.1145/157709.157715

Curtis, B., Sappidi, J., Szynkarski, A., 2012. Estimating the Principal of an Application's Technical Debt. IEEE Softw. 29, 34–42. https://doi.org/10.1109/MS.2012.156

Cutler, A., Breiman, L., 1994. Archetypal Analysis. Technometrics 36, 338–347. https://doi.org/10.1080/00401706.1994.10485840

DeMarco, T., 1986. Controlling Software Projects: Management, Measurement, and Estimates, 1 edition. ed. Prentice Hall, Englewood Cliffs, N.J.

Döhmen, T., Bruntink, M., Ceolin, D., Visser, J., 2016. Towards a Benchmark for the Maintainability Evolution of Industrial Software Systems, in: 2016 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA). Presented at the 2016 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA), pp. 11–21. https://doi.org/10.1109/IWSM-Mensura.2016.014

Elze Tobias, Pasquale Louis R., Shen Lucy Q., Chen Teresa C., Wiggs Janey L., Bex Peter J., 2015. Patterns of functional vision loss in glaucoma determined with archetypal analysis. J. R. Soc. Interface 12, 20141118. https://doi.org/10.1098/rsif.2014.1118

Ernst, N.A., Bellomo, S., Ozkaya, I., Nord, R.L., 2017. What to Fix? Distinguishing between Design and Non-design Rules in Automated Tools, in: 2017 IEEE International Conference on Software Architecture (ICSA). Presented at the 2017 IEEE International Conference on Software Architecture (ICSA), pp. 165–168. https://doi.org/10.1109/ICSA.2017.25

Eugster, M.J.A., 2012. Performance Profiles based on Archetypal Athletes. Int. J. Perform. Anal. Sport 12, 166–187. https://doi.org/10.1080/24748668.2012.11868592

Fernández-Sánchez, C., Humanes, H., Garbajosa, J., Díaz, J., 2017. An Open Tool for Assisting in Technical Debt Management, in: 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA). Presented at the 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 400–403. https://doi.org/10.1109/SEAA.2017.60

Ferreira, K.A.M., Bigonha, M.A.S., Bigonha, R.S., Mendes, L.F.O., Almeida, H.C., 2012. Identifying thresholds for object-oriented software metrics. J. Syst. Softw., Special issue with selected papers from the 23rd Brazilian Symposium on Software Engineering 85, 244–257. https://doi.org/10.1016/j.jss.2011.05.044

Fleiss, J.L., 1971. Measuring nominal scale agreement among many raters. Psychol. Bull. 76, 378–382. https://doi.org/10.1037/h0031619

Foganholi, L.B., Garcia, R.E., Eler, D.M., Correia, R.C.M., Junior, C.O., 2015. Supporting Technical Debt Cataloging with TD-Tracker Tool. Adv Soft Eng 2015, 4:4–4:4. https://doi.org/10.1155/2015/898514

Fontana, Francesca Arcelli, Roveda, R., Vittori, S., Metelli, A., Saldarini, S., Mazzei, F., 2016. On Evaluating the Impact of the Refactoring of Architectural Problems on Software Quality, in: Proceedings of the Scientific Workshop Proceedings of XP2016, XP '16 Workshops. ACM, New York, NY, USA, pp. 21:1–21:8. https://doi.org/10.1145/2962695.2962716

Fontana, F. A., Roveda, R., Zanoni, M., 2016. Technical Debt Indexes Provided by Tools: A Preliminary Discussion, in: 2016 IEEE 8th International Workshop on Managing Technical Debt (MTD). Presented at the 2016 IEEE 8th International Workshop on Managing Technical Debt (MTD), pp. 28–31. https://doi.org/10.1109/MTD.2016.11

Griffith, I., Reimanis, D., Izurieta, C., Codabux, Z., Deo, A., Williams, B., 2014. The Correspondence Between Software Quality Models and Technical Debt Estimation Approaches, in: 2014 Sixth International Workshop on Managing Technical Debt. Presented at the 2014 Sixth International Workshop on Managing Technical Debt, pp. 19–26. https://doi.org/10.1109/MTD.2014.13

Gwet, K.L., 2014. Handbook of Inter-Rater Reliability: The Definitive Guide to Measuring the Extent of Agreement Among Raters, 4 edition. ed. Advanced Analytics, LLC, Gaithersburg, MD.

Holvitie, J., Leppänen, V., 2013. DebtFlag: Technical Debt Management with a Development Environment Integrated Tool, in: Proceedings of the 4th International Workshop on Managing Technical Debt, MTD '13. IEEE Press, Piscataway, NJ, USA, pp. 20–27.

Izurieta, C., Vetrò, A., Zazworka, N., Cai, Y., Seaman, C., Shull, F., 2012. Organizing the technical debt landscape, in: 2012 Third International Workshop on Managing Technical Debt (MTD). Presented at the 2012 Third International Workshop on Managing Technical Debt (MTD), pp. 23–26. https://doi.org/10.1109/MTD.2012.6225995

Kazman, R., Cai, Y., Mo, R., Feng, Q., Xiao, L., Haziyev, S., Fedak, V., Shapochka, A., 2015. A Case Study in Locating the Architectural Roots of Technical Debt, in: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. Presented at the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, pp. 179–188. https://doi.org/10.1109/ICSE.2015.146

Kendall, M.G., 1948. Rank correlation methods, Rank correlation methods. Griffin, Oxford, England.

Kosti, M.V., Feldt, R., Angelis, L., 2016. Archetypal personalities of software engineers and their work preferences: a new perspective for empirical studies. Empir. Softw. Eng. 21, 1509–1532. https://doi.org/10.1007/s10664-015-9395-3

Li, S., Wang, P., Louviere, J., Carson, R., 2003. ARCHETYPAL ANALYSIS: A NEW WAY TO SEGMENT MARKETS BASED ON EXTREME INDIVIDUALS 6.

Li, Z., Avgeriou, P., Liang, P., 2015. A systematic mapping study on technical debt and its management. J. Syst. Softw. 101, 193–220. https://doi.org/10.1016/j.jss.2014.12.027

Marinescu, R., 2012. Assessing technical debt by identifying design flaws in software systems. IBM J. Res. Dev. 56, 9:1-9:13. https://doi.org/10.1147/JRD.2012.2204512

Martini, A., Bosch, J., 2016. An Empirically Developed Method to Aid Decisions on Architectural Technical Debt Refactoring: AnaConDebt, in: 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C). Presented at the 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), pp. 31–40.

Mayr, A., Plösch, R., Körner, C., 2014. A Benchmarking-Based Model for Technical Debt Calculation, in: 2014 14th International Conference on Quality Software. Presented at the 2014 14th International Conference on Quality Software, pp. 305–314. https://doi.org/10.1109/QSIC.2014.35

Mendes, T.S., Gomes, F.G.S., Gonçalves, D.P., Mendonça, M.G., Novais, R.L., Spínola, R.O., 2019. VisminerTD: a tool for automatic identification and interactive monitoring of the evolution of technical debt items. J. Braz. Comput. Soc. 25, 2. https://doi.org/10.1186/s13173-018-0083-1

Mittas, N., Angelis, L., 2020. Data-driven benchmarking in software development effort estimation: The few define the bulk. J. Softw. Evol. Process n/a, e2258. https://doi.org/10.1002/smr.2258

Mittas, N., Karpenisi, V., Angelis, L., 2014. Benchmarking Effort Estimation Models Using Archetypal Analysis, in: Proceedings of the 10th International Conference on Predictive Models in Software Engineering, PROMISE '14. ACM, New York, NY, USA, pp. 62–71. https://doi.org/10.1145/2639490.2639502

Moliner, J., Epifanio, I., 2019. Robust multivariate and functional archetypal analysis with application to financial time series analysis. Phys. Stat. Mech. Its Appl. 519, 195–208. https://doi.org/10.1016/j.physa.2018.12.036

Mori, A., Vale, G., Viggiato, M., Oliveira, J., Figueiredo, E., Cirilo, E., Jamshidi, P., Kastner, C., 2018. Evaluating Domain-Specific Metric Thresholds: An Empirical Study, in: 2018 IEEE/ACM International Conference on Technical Debt (TechDebt). Presented at the 2018 IEEE/ACM International Conference on Technical Debt (TechDebt), pp. 41–50.

Nayebi, M., Cai, Y., Kazman, R., Ruhe, G., Feng, Q., Carlson, C., Chew, F., 2019. A Longitudinal Study of Identifying and Paying Down Architecture Debt, in: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). Presented at the 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 171–180. https://doi.org/10.1109/ICSE-SEIP.2019.00026

Nugroho, A., Visser, J., Kuipers, T., 2011. An empirical model of technical debt and interest, in: Proceedings of the 2nd Workshop on Managing Technical Debt, MTD '11. Association for Computing Machinery, Waikiki, Honolulu, HI, USA, pp. 1–8. https://doi.org/10.1145/1985362.1985364

Oliveira, P., Lima, F.P., Valente, M.T., Serebrenik, A., 2014a. RTTool: A Tool for Extracting Relative Thresholds for Source Code Metrics, in: 2014 IEEE International Conference on Software Maintenance and Evolution. pp. 629–632. https://doi.org/10.1109/ICSME.2014.112

Oliveira, P., Valente, M.T., Lima, F.P., 2014b. Extracting relative thresholds for source code metrics, in: 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE). Presented at the 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE), pp. 254–263. https://doi.org/10.1109/CSMR-WCRE.2014.6747177

Pearson, C.S., 2015. Awakening the Heroes Within: Twelve Archetypes to Help Us Find Ourselves and Transform Our World, First Edition, First Pinting edition. ed. HarperOne, San Francisco.

Pinheiro, J., Bates, D., 2000. Mixed-Effects Models in S and S-PLUS, Statistics and Computing. Springer-Verlag, New York.

Porzio, G.C., Ragozini, G., Vistocco, D., 2008. On the use of archetypes as benchmarks. Appl. Stoch. Models Bus. Ind. 24, 419–437. https://doi.org/10.1002/asmb.727

Runeson, P., Höst, M., 2008. Guidelines for conducting and reporting case study research in software engineering. Empir. Softw. Eng. 14, 131. https://doi.org/10.1007/s10664-008-9102-8

Sadowski, C., Gogh, J. van, Jaspan, C., Söderberg, E., Winter, C., 2015. Tricorder: Building a Program Analysis Ecosystem, in: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. Presented at the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, pp. 598–608. https://doi.org/10.1109/ICSE.2015.76

Salkind, N.J. (Ed.), 2010. Encyclopedia of Research Design, 1 edition. ed. SAGE Publications, Inc, Thousand Oaks, Calif.

Schmidt, R.C., 1997. Managing Delphi Surveys Using Nonparametric Statistical Techniques*. Decis. Sci. 28, 763–774. https://doi.org/10.1111/j.1540-5915.1997.tb01330.x

Scott, W.A., 1955. Reliability of Content Analysis: The Case of Nominal Scale Coding. Public Opin. Q. 19, 321–325.

Seiler, C., Wohlrabe, K., 2013. Archetypal scientists. J. Informetr. 7, 345–356. https://doi.org/10.1016/j.joi.2012.11.013

Solingen, R. van, Basili, V., Caldiera, G., Rombach, H.D., 2002. Goal Question Metric (GQM) Approach, in: Encyclopedia of Software Engineering. American Cancer Society. https://doi.org/10.1002/0471028959.sof142

Thøgersen, J.C., Mørup, M., Damkiær, S., Molin, S., Jelsbak, L., 2013. Archetypal analysis of diverse Pseudomonas aeruginosatranscriptomes reveals adaptation in cystic fibrosis airways. BMC Bioinformatics 14, 279. https://doi.org/10.1186/1471-2105-14-279

Tornhill, A., 2018. Assessing Technical Debt in Automated Tests with CodeScene, in: 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). Presented at the 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 122–125. https://doi.org/10.1109/ICSTW.2018.00039

Tsanousa, A., Laskaris, N., Angelis, L., 2015. A novel single-trial methodology for studying brain response variability based on archetypal analysis. Expert Syst. Appl. 42, 8454–8462. https://doi.org/10.1016/j.eswa.2015.06.058

Veado, L., Vale, G., Fernandes, E., Figueiredo, E., 2016. TDTool: Threshold Derivation Tool, in: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, EASE '16. ACM, New York, NY, USA, pp. 24:1–24:5. https://doi.org/10.1145/2915970.2916014

Watson, P.F., Petrie, A., 2010. Method agreement analysis: A review of correct methodology. Theriogenology 73, 1167–1179. https://doi.org/10.1016/j.theriogenology.2010.01.003

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A., 2000. Experimentation in Software Engineering: An Introduction, International Series in Software Engineering. Springer US.

Xiao, L., Cai, Y., Kazman, R., 2014a. Titan: a toolset that connects software architecture with quality analysis, in: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2014. Association for Computing Machinery, Hong Kong, China, pp. 763–766. https://doi.org/10.1145/2635868.2661677

Xiao, L., Cai, Y., Kazman, R., 2014b. Design rule spaces: a new form of architecture insight, in: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014. Association for Computing Machinery, Hyderabad, India, pp. 967–977. https://doi.org/10.1145/2568225.2568241

Yamashita, A., 2015. Experiences from performing software quality evaluations via combining benchmark-based metrics analysis, software visualization, and expert assessment, in: 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME). Presented at the 2015 IEEE International

Conference on Software Maintenance and Evolution (ICSME), pp. 421–428. https://doi.org/10.1109/ICSM.2015.7332493

Zazworka, N., Vetro', A., Izurieta, C., Wong, S., Cai, Y., Seaman, C., Shull, F., 2014. Comparing four approaches for technical debt identification. Softw. Qual. J. 22, 403–426. https://doi.org/10.1007/s11219-013-9200-8

Zuur, A., Ieno, E.N., Walker, N., Saveliev, A.A., Smith, G.M., 2009. Mixed Effects Models and Extensions in Ecology with R, Statistics for Biology and Health. Springer-Verlag, New York. https://doi.org/10.1007/978-0-387-87458-6