

An Overview and Comparison of Technical Debt Measurement Tools

Paris Avgeriou, University of Groningen

Davide Taibi, Tampere University

Apostolos Ampatzoglou, University of Macedonia

Francesca Arcelli Fontana, University of Milano-Bicocca

Terese Besker, Chalmers University of Technology

Alexander Chatzigeorgiou, University of Macedonia

Valentina Lenarduzzi, LUT University

Antonio Martini, University of Oslo

Athanasia Moschou, University of Macedonia

Ilaria Pigazzini, University of Milano-Bicocca

Nyyti Saarimäki, Tampere University

Darius Sas, University of Groningen

Saulo Soares de Toledo, University of Oslo

Angeliki Tsintzira, University of Macedonia

// Different tools adopt different terms, metrics, and ways to identify and measure technical debt. We attempt to clarify the situation by comparing the features and popularity of technical debt measurement tools and analyzing the existing empirical evidence on their validity. //

TECHNICAL DEBT (TD) has grown to be one of the most important metaphors^{1,2} to describe development shortcuts that are taken for expediency but cause the degradation of internal software quality. The metaphor has also served well the discourse between engineers and management on how to invest resources on maintenance alongside features and bugs.

Due to its importance, several tools have been released that offer to measure TD through static code analysis (the most common way of addressing TD). These are both commercial tools and research prototypes. However, each tool uses different metrics, indices, quality models, static analysis rules, TD remediation models, and definitions of the various TD concepts. This leaves developers baffled as to how to select the most fitting TD tool for the task at hand.

Moreover, many of the tools that proclaim themselves to be TD measurement tools do not even calculate a TD index (TDI) in terms of money or effort but simply report the detection of smells or other code issues. This poses the risk that



anything wrong in the code will be considered as TD; thus, the TD metaphor will be diluted and lose its value as a means of translating internal quality issues into monetary values (currency or effort) and risks.

Our aim is to provide an overview of the current landscape of TD measurement tools through a set of objective criteria related to the offered features and their popularity. Practitioners can use this overview to assess the tools, understand their strengths and weaknesses, and ultimately select the most suitable one for their needs. The scope of the comparison is limited to three specific types of TD—namely, code, design, and architecture—as they are the most studied types.³

We considered 26 tools and filtered them to select nine for analysis based on whether they actually measure TD, either directly or through a proxy. Subsequently, we used multiple sources to collect information on their features and popularity, and we devised a set of criteria to evaluate each tool. To verify our findings in terms of correctness and completeness, we asked the corresponding tool vendors to review them and provide us with feedback.

Acknowledging that users would be reluctant to rely on tools that provide inaccurate results, we further looked into the way these tools were validated in the literature, and we present the amount of collected empirical evidence. Finally, to better guide practitioners, we offer our own interpretation of the findings by discussing how to select a tool, which tools are best for what, which are popular in different communities, as well as what is still missing.

Background

TD is a “design or implementation construct that is expedient in the short term, but sets up a technical context that can make a future change more costly or impossible” and is “limited to internal system qualities, primarily maintainability and evolvability.”¹ TD expresses the development of an artifact 1) in a “quick and dirty” way for the sake of speeding up development or 2) optimally but later rendered suboptimally because of a change in context (e.g., third-party libraries getting outdated). In any case, this debt may need repayment, e.g., through refactoring, as maintainability and evolvability become harder. Many types of TD have been studied by researchers and academics, such as code, architectural, testing, and requirements debt.⁴

The TD metaphor relies on two main concepts borrowed from economics: principal and interest. *Principal* refers to the cost of refactoring software artifacts so that they reach the desired level of maintainability and evolvability.¹ *Interest* is the extra effort that developers spend when making changes because of the existence of TD, e.g., because of code smells or unnecessarily complex code.¹

In related work, Arcelli et al.⁵ investigated in detail how TDIs are calculated by five tools in terms of both their input (e.g., code violations) and output (e.g., remediation cost). Results showed that not all tools use architectural information, while the estimation of remediation costs relied predominantly on static analysis. However, to the best of our knowledge, there is no comprehensive comparison of the available TD tools, especially taking into account the overall set of offered features and their popularity among practitioners and researchers.

Setting the Stage

To systematically perform the tool comparison, we have set up an empirical study comprising five steps.

Identifying Relevant Tools

For the first step, identifying relevant tools, we performed an academic literature search and a web search.

- *Literature search:* We relied on the IEEE Xplore and ACM Digital Library search engines. Our search string was applied on the title and abstract and had the following form: “technical debt” AND (“measurement” or “assessment” or “estimation”) AND (“tool” or “platform”). We gathered the studies that resulted from the aforementioned search and filtered out those that neither introduced nor mentioned any TD tool. We then checked the articles that cited them (forward snowballing).
- *Web search:* We used major search engines, such as Google, Bing, and Yahoo, using the same query as in the literature search. The results led us either to the landing pages of the websites of companies that own the tools or to articles introducing tools for assessing TD.

We note that, although many synonyms (or near synonyms) of TD could be used in the search string, we opted not to broaden it using terms similar to *TD symptoms* or *remediation actions*, such as *refactorings*, *code smells*, *antipatterns*, and so on. This could lead to multiple narrow-scoped tools that would be excluded later because they do not aim at estimating the effort required to eliminate the identified inefficiencies.

To ensure we did not miss relevant tools, we manually cross-checked with 1) the tool demo sessions of the first and second International Conference on TD in 2018 and 2019, respectively, and 2) all tools mentioned in a tertiary study on TD management.³ No additional tools were identified through the cross-check. The complete list of tools from this step is available in the replication package.

Tool Filtering

For the second step, tool filtering, we checked the aforementioned list of tools against the following criteria.

- *Inclusion criterion:* The tool calculates an aggregate measure of the system's TD principal and/or interest either directly (in terms of money or effort) or as a proxy based on static code analysis.
- *Exclusion criterion:* The tool is not accessible; e.g., it is not able to be downloaded or installed, lacks documentation for installation/deployment, or has an inactive website.

The inclusion criterion ensures that the selected tools match the scope of the article: they actually estimate the key concepts of the TD metaphor (interest and principal). Tools that identify code smells, without any assessment of the time that is required to resolve them, fail this criterion.

As a proxy for the TD principal and interest, we refer to any measure that does not directly represent these quantities but is correlated to them. For example, DV8 does not provide a complete TD interest index, but an accompanying study⁶ explains how the extra time spent on fixing bugs due to the presence of TD was used as a proxy for TD interest. After applying the inclusion/exclusion

criteria, nine tools were retained for data extraction (see Table 1).

Tool Assessment Criteria

For the third step, tool assessment criteria, we performed a focus group discussion (among the authors of this article) to derive a set of criteria that can be used by practitioners to assess the strengths and weaknesses of each solution. The selected criteria can be classified into three main groups: features, popularity, and validation. The offered features were collected by inspecting the documentation and websites of the tools and by trying them out (whenever a demo license was available). The major criteria are shown in Table 1. (See the replication package⁷ for the full set of 18 criteria.)

We worked in groups of either two or three researchers to collect data, whereas we discussed in plenary how to classify calculated measures into principal and interest. The second group of criteria refers to the industrial and research popularity of the tools. We evaluated popularity in terms of how often the tools are mentioned in public online sources. The following sources were investigated:

- *Online media:* We examined a number of channels used by practitioners to share information online (posts, tags, users, groups, or websites pertaining to the tools). In particular, we searched the tools' own communities, LinkedIn and Google groups, as well as the number of appearances in commonly used communities and discussion forums, such as StackOverflow, Reddit, DZone, and Medium.
- *Scientific literature:* We used Google Scholar and Scopus to

investigate the popularity of each tool by applying the following search string on all fields including the title, abstract, body, and references: ("tool_Name" or "tool_url") and "Technical Debt." In the case of tools with different names (e.g., CAST), we considered all variants in the or term, e.g., ("CAST software" or "Castsoftware" or "CAST AIP"). Two authors independently evaluated the relevance of each publication reported by Google Scholar and Scopus so as to exclude non-English articles, false positives, or articles from different domains. In the case of a disagreement, a third author provided his or her opinion.

Verifying Our Analysis

For the fourth step, verifying our analysis, we contacted the tool vendors by email and asked them to assess the correctness of our evaluation and update any data point that was incorrectly recorded. During this process, all tool vendors responded, and only minor corrections were suggested.

Empirical Evidence on the Accuracy of Each Tool

For the fifth step, empirical evidence on the accuracy of each tool, we performed a multivocal literature review,⁸ including peer-reviewed (Scopus and Google Scholar) and gray literature. In both cases, we applied the following search string: "tool_name and (evaluation or empirical or validation or accuracy or assess*)." For the keyword "tool_name," we adopted the same combinations of keywords used for the popularity search. We also asked the tool vendors to send us any related documents. The origin of each

Table 1. The characteristics of the TDIs and other features in the analyzed tools.

	Characteristics of the TDIs				
Name (Release year)	Type	Principal	Interest	Index	
CAST (1998)	Architectural, design, and code	Time to remove issues	Yes	Violations × rule criticality × effort	
SonarGraph (2006)	Architectural and design	Computation of several metrics	No	Structural debt index × minutes to fix	
NDepend (2007)	Architectural, design, and code	Estimated person time to fix issues	Yes	Violations × fix effort	
SonarQube (2007)	Code	Time to remove issues	No	Cost to develop one line of code × number of lines of code	
SQuORE (2010)	Design and code	Time to remove issues	No	No	
CodeMRI (2013)	Design	Not estimated	Yes	Interest—not mentioned	
Code Inspector (2019)	Architectural, design, and code	Effort needed to avoid high TD	No	A function of violations, duplications, and readability/ maintainability issues	
DV8 (2019)	Architectural	Number of affected files and lines of code	Yes	Penalties: additional bugs and/or changes in lines of code	
SymfonyInsight (2019)	Code	Time to remove issues	No	Number of issues × time needed to remove the issue	
Additional features					
Name	Platform	Integration	Output	Other quality attributes	Execute
CAST	Windows	Jenkins and Maven	API and GUI	Security, efficiency, changeability, robustness, and transferability	Asynchronous
SonarGraph	Independent	Eclipse, Gradle, IntelliJ Jenkins, Maven, and VS	GUI	Changeability	Real time
NDepend	Windows	Azure, Jenkins, and VS	GUI	Changeability, robustness, and testability	Asynchronous
SonarQube	Independent	Eclipse, IntelliJ, and VS	All [*]	Security and reliability	Real time
SQuORE	Independent	No	API and GUI	Changeability, reliability, efficiency, portability, security, and testability	Asynchronous
CodeMRI	Windows, Linux	No	CLI	Security, efficiency, robustness, portability, and testability	Asynchronous
Code Inspector	Independent	GitHub, GitLab, Bitbucket, Jenkins, and Travis	API	Security, changeability, portability, testability, and maintainability	Asynchronous
DV8	Windows and Mac	Depends and Jenkins	GUI	Maintainability, evolvability, and security	Real time
SymfonyInsight	Independent	No	GUI and CI	Security, maintainability, and reliability	Asynchronous

API: application programming interface; CI: continuous integration; CLI: command line interface; GUI: graphical user interface; VS: Visual Studio.

*All refers to API, GUI, CLI, and CI.

article (peer reviewed, gray literature, or from a vendor) is referenced in the replication package.

Findings on Features

Table 1 reports our key findings regarding the tools selected for comparison. (Tools are sorted in chronological order.) The table comprises two parts: 1) the characteristics of the different TDIs and 2) additional tool features (such as export, integration with other tools, and customizability).

For every index, we look into the interest, principal, and measurement method (which factors are used to compute the index value). Interestingly, not all of the tools consider the interest, but all (except CodeMRI) compute the principal. The latter is usually identified with a heuristic based, in some cases, on software metrics and, in other cases, on the effort needed to fix the identified software violations, expressed in either effort (in minutes) or monetary form.

In general, every selected tool is able to inspect both sources and binaries of a given software project and analyze at different granularity levels: project, package, class, method, and line of code. The analysis usually results in the identification of violations and anomalies, which are highlighted in the code through the tool's own user interface or in the IDEs that support plug-ins for six out of nine of the analyzed tools.

All tools have different degrees of customization. All of the tools in the study allow developers to select the rules for the analysis. In addition, five tools (CAST, NDepend, SonarGraph, CodeMRI, and SonarQube) allow users to add rules (e.g., define a new metric) and customize their thresholds. One tool

(SymfonyInsight) allows only customization of the thresholds, and two tools (Code Inspector and DV8) do not allow users to add rules or customize thresholds. Finally, all of the tools, except NDepend and CodeMRI, allow the creation of new plug-ins.

Furthermore, all tools address additional quality attributes. We report the names of the qualities as reported by the vendors in Table 1, and we also provide a mapping to the software quality standards that the qualities refer to in the replication package.⁷

Findings on Popularity

In Figure 1(b), we report the results related to the popularity of the tools in the Stack Overflow, LinkedIn, and Google groups as well as other popular sites, such as Reddit, Dzone, and Medium. Search strings and raw data are available online in the replication package.⁷ Please note that the results are normalized against the number of years since the introduction of each tool.

SonarQube is by far the most popular tool, and it is visible in all of the channels. In most cases, NDepend comes in second, being present in all of the channels as well but with lower magnitude than SonarQube. SonarGraph covers almost all channels, although with fewer hits than NDepend and SonarQube, while it does not have tags in Stack Overflow. CAST scores only a few hits in Stack Overflow and other channels, while it has a large community on LinkedIn compared to the other tools (although it is still second after SonarQube). Finally, DV8, CodeInspector, CodeMRI, SQuORE, and SymfonyInsight are the least popular tools, with only a handful of posts.

As for the popularity in the scientific literature [the radial bar charts in Figure 1(a)], SonarQube and CAST are clearly the most popular tools, matching the results reported earlier (see Lenarduzzi et al⁹). Combining the findings from the research literature and online media, it is clear that SonarQube is the most popular tool, whereas the results for CAST and SonarGraph are comparable. In the case of NDepend, it seems to be more popular in industry than academia.

Findings on Validation

Applying the search string returned a total of 5,313 publications. Next, we filtered the obtained studies based on their relevance to TD and, in particular, to the evaluation of the proposed indices for TD principal or interest, obtaining a list of 122 articles for a more detailed inspection. As a final step of study inclusion/exclusion, we proceeded to a full-text reading, through which we excluded 72 additional studies as irrelevant.

The data extraction was performed on the remaining 50 studies. These articles were classified based on the relevance of the empirical evaluation. A full relevance point was given to articles that evaluate the TD principal or interest index with respect to its accuracy of measurement in terms of the used unit (i.e., effort or money); a partial point was assigned to articles that assess the relation of TD principal or interest index to other qualities (e.g., maintainability, reliability, and so on). This aligns with the scope of this article, i.e., the ability of the tools to provide indices for TD principal and/or interest. All raw data extracted during this process are available in the replication package.

As shown in Figure 1(c), SonarQube is the tool whose measures

Literature (middle) and Web (right) popularity of TD tools

in # of hits/years since release

(Tools with no hits are not visualized)
(Web popularity normalized by tool)

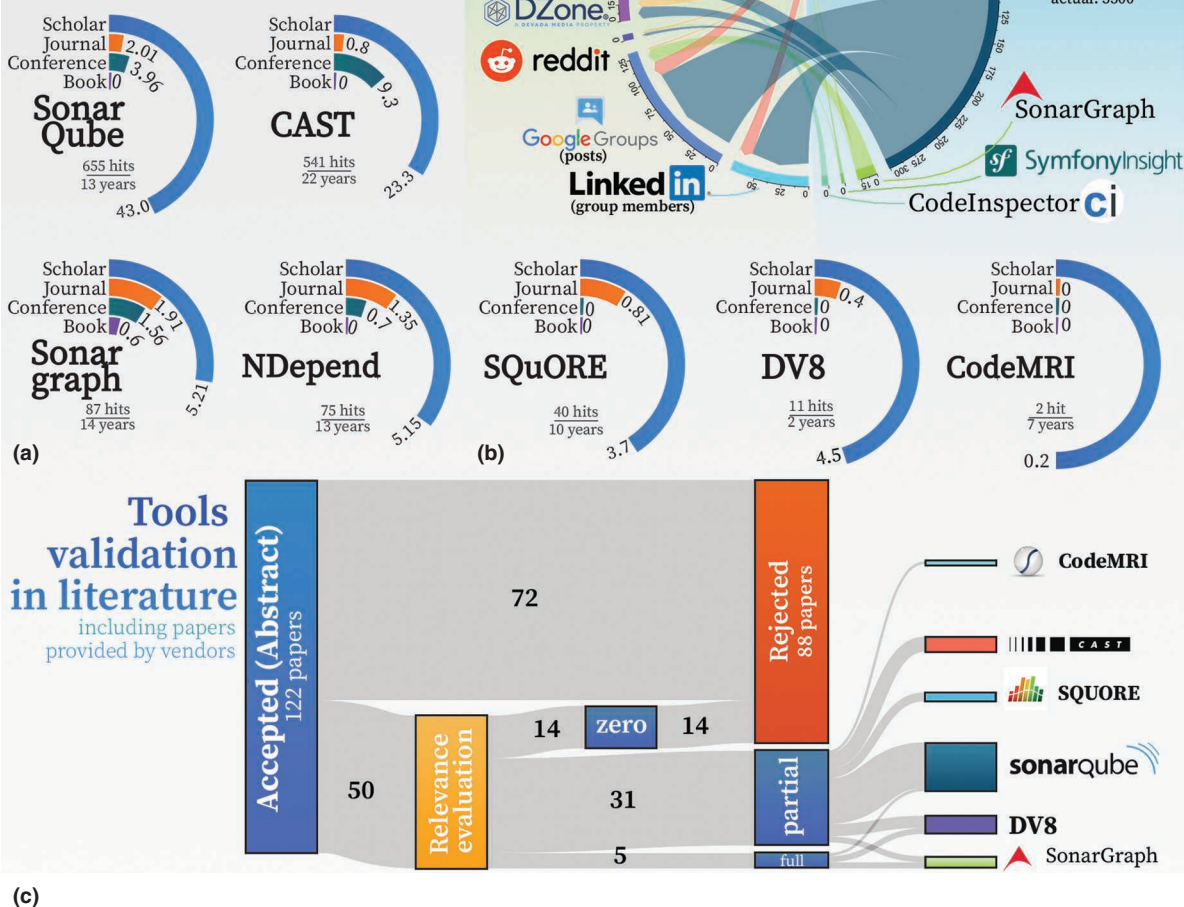


FIGURE 1. An infographic depicting the (a) popularity in the scientific literature (radial bar charts normalized per tool), (b) popularity on the web (chord chart), and (c) empirical validation of TD tools in the literature (Sankey diagram). All values in the radial diagrams and chord chart are in number of hits divided by the number of years.

have been considered more in empirical evaluations, followed by DV8 and CAST. However, regarding the accuracy of the TDI, only DV8, SonarGraph, and SonarQube have been considered in empirical studies. Based on these results, we find that TD quantification in units of

effort is still lacking empirical validation in terms of its accuracy; this may lead to practitioners not having full confidence in the remediation effort and order proposed. However, we argue that the existing tools can be safely used for TD refactoring, since they are able to identify

TD items in some meaningful (and actionable) way.

Discussion

How to Select a Tool

There is no clear “winner” that is the best option for all uses and

organizations—different tools better fit various purposes. We provide some tips on how teams can select a tool according to their needs.

First, it is important to think whether the measurement of TD principal and interest (or at least their proxies) is required to perform TD analysis. Some teams may simply require tools that analyze their codebase to find code smells and calculate quality metrics; numerous tools serve this purpose.¹⁰ If, however, principal and interest are a “must have,” as indicated in recent studies in several companies,¹¹ one should restrict the selection to the tools reported in this article. The tools listed in Table 1 calculate principal and interest differently; we advise teams to choose tools based on what helps them the most to prioritize refactoring.

Next, individual developers usually need tools that measure code debt only, but when the analysis involves larger or multiple teams, then tools analyzing the architectural debt are highly recommended. Other contextual factors that are useful to narrow down the selection of a tool include languages, IDEs, platforms, the license, and the architecture (server or client side). Finally, the involvement of tools in research articles might provide practitioners with further insights on the reliability of the studied tools—in some cases, supported by empirical evidence.

Which Tools Are Best for What?

All tools (but one) calculate principal, but only four of them calculate interest—NDepend, CAST, DV8, and CodeMRI—so these should be the tools of choice for developers interested in estimating the extra maintenance effort required in future iterations. For practitioners

interested in security, both CAST and SonarQube offer support, although CAST analyzes a higher number of security rules. Changeability and, more generally speaking, maintainability are considered by all of the tools; however, the front-runners are CAST, NDepend, and SQuORE, which offer elaborated functionality to manage maintainability at multiple levels through advanced features, such as custom component dependency violation, dependency graph analysis, and control flow analysis.

For detailed architectural analysis, CAST, NDepend, and SonarGraph provide several features that aid the user in gauging whether the intended architecture of the system matches the actual one. Users who manage code bases with a plethora of programming languages should definitely consider SonarQube, which is able to analyze the largest number of languages (26). DV8 takes into account not just the source code of a specific version but also version history and issue trackers. Such an approach renders the analysis richer by using more sources of data to measure evolutionary coupling (coupling discovered via co-changes in different snapshots) and its interest in terms of penalties incurred during bug fixing; using historical data also strengthens the reliability of its results.

Which Tools Are Popular Among Practitioners and Researchers?

We observed that the communities behind the analyzed tools differ significantly. In particular, SonarQube and NDepend are the only tools discussed in the Stack Overflow community, with SonarQube being by far the one with the most questions asked and answered. The organization behind SonarQube seems to

invest in supporting the TD community by creating posts, tags, and answers to users’ questions. However, in the majority of cases, the posts are not explicitly related to TD but more related to setting up and customizing the tool.

Examining the communities on LinkedIn, numerous members discuss SonarQube and, to a lesser extent, CAST, while SonarGraph seems to have a small community in Google groups. However, the presence in these communities can be seen both as a sign of popularity but also as a way for the tools to create visibility for marketing purposes. In summary, SonarQube seems to have a strong community behind the tool, while NDepend and CAST are present in selected channels as is, to a lesser extent, SonarGraph. The remaining tools do not seem to have an online community supporting them. Although popularity cannot be considered a quality index per se (less precise tools can become more popular due to better marketing), we believe that a tool that is widely used by practitioners inherently gives them some value, or it would not be used and discussed at all.

What Is Still Missing?

First, all analyzed tools quantify the level of maintainability issues (i.e., the principal), but not all tools focus on the consequence of these issues (i.e., the interest). This weakens the use of TD as a communication medium: practitioners can communicate the existence of the problem (principal), but they do not have numbers on extra maintenance costs (interest) nor the probability of additional maintenance (interest probability) to argue about repaying TD. It is crucial that all dimensions of the TD metaphor are represented.

Second, all analyzed tools but one (DV8) consider only static analysis in their TD calculation models. However, current software development practices entail additional rich sources of information (e.g., version history, issue trackers, email exchanges, and so on); these can be exploited for improving the accuracy of indicators or providing different perspectives.

Third, all tools focus on a limited set of types of TD: they work predominantly on code TD, to a lesser extent on design debt, and in a rather limited sense on architectural debt. This is not a coincidence: code and design debt are the easiest types to detect and, usually, repair. However, we argue that architectural debt has a much larger impact on maintenance efforts than other types.¹²

Last but not least, there is no commonly agreed on and validated set of rules and metrics for measuring TD. Instead, each tool uses its own set of rules and metrics without detailed explanation or motivation. Thus, there might exist discrepancies among the tools regarding the rules and output remediation time, and this creates confusion about which rules are important and how to customize their severity to match one's needs.

Limitations

The results of this article are subject to some limitations. The first one is the narrow search string we applied. We are aware that using different synonyms or relaxing the search string might have yielded more results. However, we aimed at using the terminology adopted by the TD community. The choice of our inclusion/exclusion criteria also affected the selection of tools. We have aligned the inclusion criteria with the scope of the article; thus, only tools that directly or indirectly measure TD were included.

As for data extraction, different researchers collected the information for different tools and, therefore, possibly obtained information differently. We mitigated this threat by first assigning data collection per tool to at least two researchers with experience on that tool; any differences in opinion among them were discussed and resolved. Subsequently, the tool vendors were asked to inspect the results.

Furthermore, the online popularity of the tools could be biased by the activity of their respective communities: we compared the number of posts and not the number of tool users. Some tools may have very active but small communities; others may be widely used but not largely discussed online. In addition, for some tools, the discussion may happen elsewhere, such as in mailing lists or forums.


The results rely mostly on quantitative indicators to provide useful insights about the tools, but we warn against using such numbers as an absolute way to assess their quality. To mitigate this limitation, we have added an extensive discussion based on the researchers' qualitative interpretation gathered during the assessment procedure.

Finally, despite our best efforts, our personal experience using the analyzed tools might have biased the data analysis. Specifically, we have extensive experience with SonarQube (18 published articles) and some familiarity with CAST (3 article), SonarGraph (2 articles), and SQuORE (1 article); we had no background with the other tools. We mitigated this by collecting objective data instead of user opinions and by making all of the data for this study freely available online to allow other researchers to replicate this article.⁷

In this article, we highlighted the current state of the market for TD tools, focusing on those providing an estimation of TD principal and/or interest. These tools have been selected through a rigorous process and were analyzed regarding their offered features, popularity, and accompanying evidence.

The studied tools offer a comprehensive variety of functionalities that cover multiple languages, levels of analysis, and artifacts as well as different computations of TD principal and interest. They can, to some extent, identify, measure, and monitor TD as well as provide suggestions for repayment. More importantly, they support the communication of TD through monetary values, both horizontally among the technical teams and vertically between the technical and management teams.

Our analysis offers practitioners a clearer overview of the current landscape of TD tools and highlights their differences in offered features, popularity, and empirical validation as well as current shortcomings. Our results allow the tools to be compared against each other so that an informed choice can be made on which one best suits the needs of individual developers or their teams.

As a follow-up to this article, we plan to conduct a user study with practitioners to compare the tools based on concrete TD management tasks. This would complement the current study with information on the usability and usefulness of the tools. 

References

1. P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing Technical Debt in software engineering (Dagstuhl seminar 16162)," *Dagstuhl*



PARIS AVGERIOU is professor of software engineering at the University of Groningen, Groningen, 9747 AG, The Netherlands, where he has led the software engineering research group since September 2006. Avgeriou received a Ph.D. from the National Technical University of Athens, Greece. He is a Senior Member of IEEE. Further information about him can be found at <http://www.cs.rug.nl/~paris/>. Contact him at p.avgeriou@rug.nl.



DAVIDE TAIBI is an associate professor at Tampere University, Tampere, 33014, Finland. He is a Member of IEEE and the IEEE Computer Society. Further information about him can be found at <http://www.taibi.it>. Contact him at davide.taibi@tuni.fi.



APOSTOLOS AMPATZOGLOU is an assistant professor of software engineering in the Department of Applied Informatics at the University of Macedonia, Thessaloniki, 546 36, Greece. Ampatzoglou received his Ph.D. in software engineering from the Aristotle University of Thessaloniki in 2012. Further information about him can be found at <https://users.uom.gr/~a.ampatzoglou/>. Contact him at apostolos.ampatzoglou@gmail.com.



FRANCESCA ARCELLI FONTANA is a full professor at the University of Milano Bicocca, Milan, 20126, Italy. Arcelli Fontana received her Ph.D. in computer science from the University of Milano. She is a Member of IEEE. Contact her at francesca.arcelli@unimib.it.



TERESE BESKER received her Ph.D. in software engineering from Chalmers University of Technology, Gothenburg, 41756, Sweden, in 2020. Besides her research career, she has worked as a senior software engineer in the software industry for more than 15 years. She has published several peer-reviewed articles in journals as well as conference and workshop proceedings. She is a Member of IEEE. Contact her at terese.besker@gmail.com.



ALEXANDER CHATZIGEORGIOU is a professor of software engineering in the Department of Applied Informatics at the University of Macedonia, Thessaloniki, 546 36, Greece. Chatzigeorgiou received his Ph.D. in computer science from the Aristotle University of Thessaloniki, Greece, in 2000. Further information about him can be found at <https://users.uom.gr/~achat/>. Contact him at achat@uom.edu.gr.



VALENTINA LENARDUZZI is a researcher at LUT University, Lahti, 15210, Finland. In 2011 she was one of the cofounders of Opensoftengineering s.r.l., a spin-off company of the Università degli Studi dell'Insubria. She is a Member of IEEE. Further information about her can be found at www.valentinalenarduzzi.it. Contact her at valentina.lenarduzzi@lut.fi.



ANTONIO MARTINI is an associate professor at the University of Oslo, Oslo, Norway, and a part-time researcher at Chalmers University of Technology, Gothenburg, 0373, Sweden. Martini received his Ph.D. in software engineering from Chalmers University of Technology, Sweden, in 2015. Further information about him can be found at <https://www.mn.uio.no/ifi/english/people/aca/antonima/index.html>. He is a Member of IEEE. Contact him at antonima@ifi.uio.no.



ABOUT THE AUTHORS



ATHANASIA MOSCHOU is a Ph.D. student in computer science at the University of Macedonia, Thessaloniki, 54636, Greece. Moschou received her master's degree in 2018 in applied informatics from the University of Macedonia, Greece. Further information about her can be found at <https://www.linkedin.com/in/nasiamoschou/>. Contact her at nasiamoschou@gmail.com.



DARIUS SAS is a Ph.D. student at the Bernoulli Institute for Mathematics, Computer Science, and Artificial Intelligence, University of Groningen, Groningen, 9747 AG, The Netherlands. Sas received his master's degree in computer science in 2018 from the University of Milano-Bicocca, Milan, Italy. Contact him at d.d.sas@rug.nl.



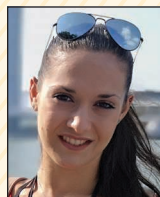
ILARIA PIGAZZINI is a Ph.D. student in computer science at the Department of Computer Science, Systems, and Communications, University of Milano-Bicocca, Milan, 20126, Italy. Pigazzini received her M.Sc. degree from the University of Milano-Bicocca in computer science in 2018. Contact her at i.pigazzini@campus.unimib.it.



SAULO SOARES DE TOLEDO is a Ph.D. candidate at the University of Oslo, Oslo, 0373, Norway. de Toledo received his master's degree in computer science from the Federal University of Campina Grande, Paraíba, Brazil. Further information about him can be found at <https://www.linkedin.com/in/saulostoledo/>. Contact him at saulos@ifi.uio.no.



NYTYI SAARIMÄKI is a Ph.D. student in software engineering at Tampere University, Tampere, 33014, Finland. Saarimäki received her M.Sc. in theoretical computer science in 2018 from Tampere University of Technology. Further information about her can be found at <https://nyyti.github.io/>. Contact her at nyyti.saarimaki@tuni.fi.



ANGELIKI TSINTZIRA is an M.Sc. student in the School of Electrical and Computer Engineering in the Aristotle University of Thessaloniki, Thessaloniki, 546 36, Greece, and a researcher at the Department of Applied Informatics of the University of Macedonia, Macedonia, 546 36, Greece. Tsintzira received her integrated master's degree in informatics and telecommunications engineering from the University of Western Macedonia, Greece. Contact her at angeliki.agathi.tsintzira@gmail.com.

Rep., vol. 6, no. 4, pp. 110–138, 2016. doi: 10.4230/DagRep.6.4.110.

2. C. Izurieta et al., “Perspectives on managing technical debt. A transition point and roadmap from Dagstuhl,” in *Proc. 1st Int. Workshop Tech. Debt Analytics (TDA). In Association 23rd Asia-Pacific*

Software Engineering Conf. (APSEC). Hamilton, New Zealand: Univ. of Waikato, Dec. 6–9, 2016, pp. 84–87.

3. N. Rios, M. G. de Mendonça Neto, and R. O. Spínola, “A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners,”

Inf. Softw. Technol., vol. 102, pp. 117–145, Oct. 2018. doi: 10.1016/j.infsof.2018.05.010.

4. Zv. Li, P. Avgeriou, and P. Liang, “A systematic mapping study on Technical Debt and its management,” *J. Syst. Softw.*, vol. 101, pp. 193–220, Mar. 2015. doi: 10.1016/j.jss.2014.12.027.

5. F. Arcelli Fontana, R. Roveda, and M. Zanoni, "Technical Debt indexes provided by tools: A preliminary discussion," in *Proc. 2016 IEEE 8th Int. Workshop Managing Technical Debt (MTD)*, Raleigh, NC, pp. 28–31. doi: 10.1109/MTD.2016.11.
6. R. Kazman et al., "A case study in locating the architectural roots of Technical Debt," in *Proc. 37th Int. Conf. Software Engineering (ICSE '15)*, 2015, vol. 2, pp. 179–188. doi: 10.1109/ICSE.2015.146.
7. P. Avgeriou et al., "An overview and comparison of Technical Debt measurement tools," 2020. Accessed: Feb. 27, 2020. [Online]. Available: <https://www.doi.org/10.6084/m9.figshare.12489290>
8. V. Garousi, M. Felderer, and M. V. Mäntylä, "Guidelines for including grey literature and conducting multivocal literature reviews in software engineering," *Inf. Softw. Technol.*, vol. 106, pp. 101–121, Feb. 2019. doi: 10.1016/j.infsof.2018.09.006.
9. V. Lenarduzzi, A. Sillitti, and D. Taibi, "A survey on code analysis tools for software maintenance prediction," in *Proc. Int. Conf. Software Engineering Defence Applications*. Cham: Springer-Verlag, June 2018, pp. 165–175. doi: 10.1007/978-3-030-14687-0_15.
10. "Source code analysis tools," OWASP Foundation, Bel Air, MD. Accessed: Feb. 27, 2020. [Online]. Available: https://owasp.org/www-community/Source_Code_Analysis_Tools
11. A. Martini, T. Besker, and J. Bosch, "Technical Debt tracking: Current state of practice: A survey and multiple case study in 15 large organizations," *Sci. Comput. Program.*, vol. 163, pp. 42–61, Oct. 2018. doi: 10.1016/j.scico.2018.03.007.
12. A. E. E. Neil, S. Stephany Bellomo, I. Ipek Ozkaya, L. Robert, R. L. Nord, and I. Ian Gorton, "Measure it? Manage it? Ignore it? Software practitioners and technical debt," in *Proc. 2015 10th Joint Meeting Foundations Software Engineering (ESEC/FSE 2015)*. New York: ACM, 2015, pp. 50–60. doi: 10.1145/2786805.2786848.



CALL FOR ARTICLES

IT Professional seeks original submissions on technology solutions for the enterprise. Topics include

- emerging technologies,
- cloud computing,
- Web 2.0 and services,
- cybersecurity,
- mobile computing,
- green IT,
- RFID,
- social software,
- data management and mining,
- systems integration,
- communication networks,
- datacenter operations,
- IT asset management, and
- health information technology.

We welcome articles accompanied by web-based demos. For more information, see our author guidelines at www.computer.org/itpro/author.htm.

WWW.COMPUTER.ORG/ITPRO



Digital Object Identifier 10.1109/MS.2021.3068504