An Overview and Comparison of Technical Debt Measurement Tools

Paris Avgeriou[1], Davide Taibi[2], Apostolos Ampatzoglou[3], Francesca Arcelli Fontana[4], Terese Besker[5], Alexander Chatzigeorgiou[3], Valentina Lenarduzzi[6], Antonio Martini[7], Nasia Moschou[3], Ilaria Pigazzini[4], Nyyti Saarimäki[2], Darius Sas[1], Saulo Soares de Toledo[6], Angeliki-Agathi Tsintzira[3]

[1] University of Groningen
[2] Tampere University
[3] University of Macedonia
[4] University of Milano-Bicocca
[5] Chalmers University of Technology
[6] LUT University
[7] University of Oslo

Keywords: Technical Debt, Tools, Source Code Analysis, Software Quality

ABSTRACT

There are numerous commercial tools and research prototypes that offer support for measuring technical debt. However, different tools adopt different terms, metrics, and ways to identify and measure technical debt. These tools offer diverse features, and their popularity / community support varies significantly. Therefore, (a) practitioners face difficulties when trying to select a tool matching their needs; and (b) the concept of technical debt and its role in software development is blurred. 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. Our findings can help practitioners to find the most suitable tool for their purposes, and researchers by highlighting the current tool shortcomings.

Introduction

Technical Debt (TD) has grown to be one of the most important metaphors [Avgeriou2016][Izurieta2016] to express development shortcuts, taken for expediency, but causing 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, technical debt remediation models, and definitions of the various technical debt 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 as technical debt 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 technical debt metaphor will be diluted and lose its value as a means that translates 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 of technical debt [Rios2018]. We considered 26 tools and filtered them to select 9 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 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 literature and 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

Technical debt 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*" [Avgeriou2016]. Technical debt expresses the development of an artifact: (a) in a 'quick and dirty' way for the sake of speeding up development; or (b) optimally, but later rendered sub-optimally because of 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 technical debt have been studied by researchers and academics, such as *Code*, *Architectural, Testing* and *Requirements* Debt [Li2015].

The technical debt 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 [Avgeriou2016]. Interest is the extra effort that developers spend when making changes because of the existence of technical debt, e.g. because of code smells or unnecessarily complex code [Avgeriou2016].

As related work, Arcelli et al., investigated in detail how TD indices are calculated by five tools [Arcelli2016], 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 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. For the first step (***identify 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 papers that cite 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 technical debt could be used in the search string, we opt not to broaden it using terms similar to Technical Debt symptoms or remediation actions, such as refactorings, code smells, anti-patterns, etc. This could lead to multiple, narrow-scoped tools that would be later on excluded because they do not aim at estimating the effort required to eliminate the identified inefficiencies.

In order to ensure we did not miss relevant tools, we manually cross-checked with: (a) the tool demo sessions of the 1st and 2nd International Conference on Technical Debt, in 2018 and 2019 respectively; (b) all tools mentioned in a tertiary study on technical debt management [Rios2018]. No additional tools were identified through cross-check. The complete list of tools from this step is available in the replication package.

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 technical debt 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. not being able to download or install it, lack of documentation for installation/deployment, inactive website.

The inclusion criterion ensures that the selected tools match the scope of the paper: 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. By *proxy* of TD principal and interest, we refer to any measure that does not directly represent TD principal or interest but is correlated to them. For example, DV8 does not provide a complete TD interest index, but an accompanying study [Kazman2015] explains how the extra time spent on fixing bugs due to the presence of TD was used as a proxy of TD interest. After applying the inclusion/exclusion criteria, nine tools were retained for data extraction (see Table 1).

For the third step (***tool assessment criteria***), we performed a focus group discussion (among the authors of this paper) 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 [ReplPackage] for the full set of 18 criteria). The authors worked in groups of either 2 or 3 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 tools. We evaluated popularity in terms of how much the tools are mentioned in public online sources. The following sources were investigated:

- *Online Media*: We investigated 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 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 papers, false positives or papers from different domains. In case of disagreement, a third author provided his/her opinion.

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

For the fifth step, (***empirical evidence on the accuracy of each tool***), we have performed a multivocal literature review [Garousi2019], including peer-reviewed (Scopus and Google Scholar) and grey 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 paper (peer-reviewed, grey 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: (a) the characteristics of the different TD indices, and (b) additional tool features (such as export, integration with other tools, and customizability).

For every index we look into *Interest*, *Principal*, and measurement method (which factors are used to compute the index value). Interestingly, not all the tools consider interest, but all (except CodeMRI) compute 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 in monetary form.

In general, every selected tool is able to inspect both sources and binaries of a given software project and to 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 tools own user interface, or in the IDEs that support plugins for six out of nine of the analyzed tools.

All tools have different degrees of customization. All 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 (SymphonyInsight) allows only customizing the thresholds, and two tools (Code Inspector and DV8) do not allow adding rules or customizing thresholds. Finally, all the tools, except NDepend and CodeMRI allow creating new plugins.

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

Table 1: Characteristics of TD indices and other features in the analyzed tools

CHARACTERISTICS OF TECHNICAL DEBT INDICES				
Name (Release Year)	**Type**	**Principal**	**Interest**	**Index**
CAST (1998)	Architectural, Design, Code	Time to remove issues	Yes	Violations * rule criticality * effort
Sonargraph (2006)	Architectural, Design	Computation of several metrics	No	structural debt index * minutes to fix
NDepend (2007)	Architectural, Design, Code	Estimated man-time to fix issues	Yes	Violations * fix effort
SonarQube (2007)	Code	Time to remove issues	No	Cost to develop 1 LOCe * Number of lines of code.
Squore (2010)	Design, Code	Time to remove issues	No	No
CodeMRI (2013)	Design	Not estimated	Yes	Interest - Not mentioned
Code Inspector (2019)	Architectural, Design, Code	Effort needed to avoid high TD	No	A function of violations, duplications, 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, Maven	API, GUI	Security, Efficiency, Changeability, Robustness, Transferability,	async
Sonargraph	Independent	Eclipse, Gradle, IntelliJ Jenkins, Maven, VS	GUI	Changeability	real-time
NDepend	Windows	Azure, Jenkins, VS	GUI	Changeability, Robustness, Testability	async
SonarQube	Independent	Eclipse, IntelliJ, VS	ALL(*)	Security, Reliability	real-time
Squore	Independent	No	API, GUI	Changeability, Reliability, Efficiency, Portability, Security, Testability	async
CodeMRI	Windows, Linux	No	CLI	Security, Efficiency, Robustness, Portability, Testability	async
Code Inspector	Independent	Github, Gitlab, Bitbucket Jenkins, Travis	API	Security, Changeability, Portability, Testability, Maintainability	async
DV8	Windows, MAC	Depends, Jenkins	GUI	Maintainability,Evolvability, Security	real-time
SymfonyInsight	Independent	No	GUI, CI	Security, Maintainability, Reliability	async

(*) ALL refers to API, GUI, Command Line Interface(CLI), and Continuous Integration (CI)

Findings on Popularity

In Figure 1 (in the chord diagram) we report the results related to the popularity of the tools in 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 [ReplPackage]. 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 the channels. In most cases, NDepend comes second, being present in all 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).DV8, CodeInspector, CodeMRI, SQuORE and SymfonyInsights are finally the least popular tools, with only a handful of posts.

As for the popularity in scientific literature (radial bar charts in Figure 1), SonarQube and CAST are clearly the most popular tools, matching the results reported earlier (see [Lenarduzzi2018]).

Combining the findings from 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.

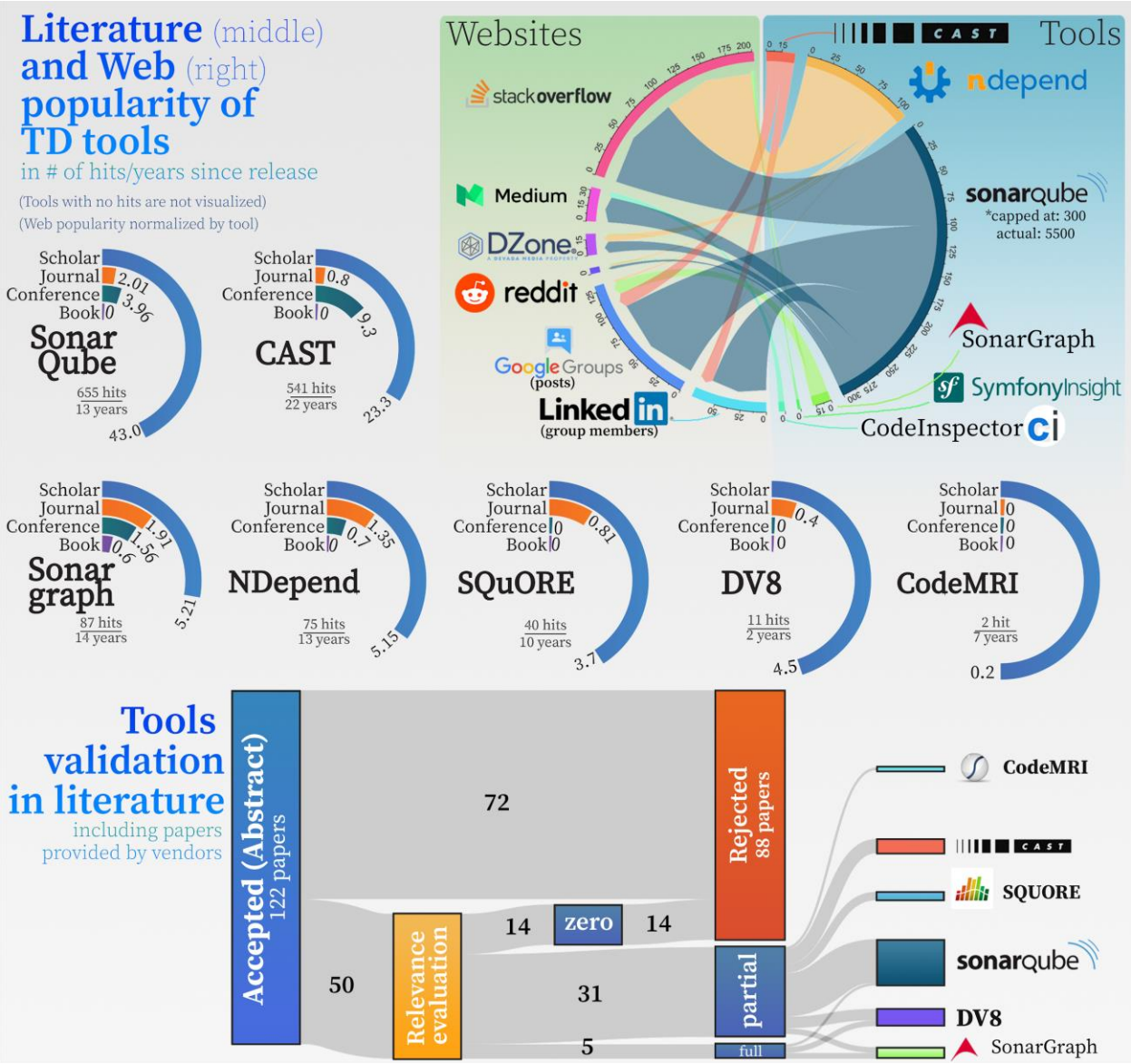


Figure 1: Infographic depicting popularity in the scientific literature (radial bar charts normalized per tool), popularity on the web (chord chart) and the empirical validation of TD tools in the literature (Sankey diagram). All values in the radial diagrams and chord chart are in # of hits divided by the number of years.

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 technical debt and in particular to the evaluation of the proposed indices for TD principal or interest, obtaining a list of 122 papers for 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 papers were classified based on the relevance of the empirical evaluation. A full relevance point was given to papers 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 papers that assess the relation of TD principal or interest index to other qualities (e.g., maintainability, reliability, etc.). This aligns with the scope of this paper, 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 (Sankey diagram at the bottom), SonarQube is the tool whose measures have been considered more in empirical evaluations, followed by DV8 and CAST. However, when considering the accuracy of the TD Index, 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 regarding 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 different purposes. We provide some tips on how teams can select a tool, according to their needs.

First, it is important to think whether 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 [OWASP]. If however principal and interest are a "must have", as indicated in recent studies in several companies [Martini2018], one should restrict the selection to the tools reported in this paper. 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: languages, IDEs, platforms, the license, and the architecture (server or client side).

Finally, the involvement of tools in research articles, might provide the 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, CodeMRI; so these should be the tools of choice for developers interested in estimating 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,* is considered by all the tools; however the front-runners are CAST, NDepend, and SQuORE, offering 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 that manage code bases with a plethora of programming languages, should definitely consider SonarQube, which is able to analyse 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 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 technical debt, 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 of 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, and to less 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, etc); 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 technical debt: they work predominantly on code technical debt, to a

lesser extent design debt and in a rather limited sense architectural debt. This is not a coincidence: code and design debt are the easiest types to detect and usually to repair. However, we argue that architectural debt has a much larger impact on maintenance efforts than other types [Ernst15].
Last but not least, there is no commonly-agreed and validated set of rules and metrics to measure Technical Debt. 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, the output remediation time, and creates confusion on which rules are important and how to customize their severity to match one's needs.

Limitations

The results of this work 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 paper, 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 requested 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. In addition, 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 papers), some experience with CAST (3 papers), with Sonargraph (2 papers) and with Squore (1 paper); we had no experience with the other tools. We mitigated this by collecting objective data instead of user opinions, and by making all the data for this study freely available online, so as to allow other researchers to replicate this work [ReplPackage].

Conclusions and future work

In this paper, 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, artifacts, as well as different computations of technical debt principal and interest. They can, to some extent, *identify*, *measure*, and *monitor* technical debt as well as provide suggestions for *repayment*. More importantly, they support the *communication* of technical debt through monetary values, both horizontally, between the technical teams, and vertically, between the technical and the management teams.

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

As follow-up of this work, 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 tools usability and usefulness.

References

[Avgeriou2016] Avgeriou, P., Kruchten, P., Ozkaya, I. and Seaman, C., 2016. “Managing technical debt in software engineering (dagstuhl seminar 16162).” In *Dagstuhl Reports* (Vol. 6, No. 4). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[Arcelli2016] Arcelli Fontana F., Roveda R. and Zanoni M., "Technical Debt Indexes Provided by Tools: A Preliminary Discussion," *2016 IEEE 8th International Workshop on Managing Technical Debt (MTD)*, Raleigh, NC, 2016, pp. 28-31.

[Ernst2015] Neil A. E.Ernst, Stephany Bellomo S., Ipek Ozkaya I., Robert L. Nord R.,L., and Ian Gorton I.. 2015. Measure it? Manage it? Ignore it? software practitioners and technical debt. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015). ACM, New York, NY, USA, 50-60.

[Izurieta2016] Izurieta C., Ozkaya I., Seaman C., Kruchten P., Nord R., Snipes W., Avgeriou P., "Perspectives on Managing Technical Debt. A Transition Point and Roadmap from Dagstuhl," 1st International Workshop on Technical Debt Analytics (TDA). In association with the 23rd Asia-Pacific Software Engineering Conference (APSEC), University of Waikato, Hamilton, New Zealand, December 6-9 2016.

[Garousi2019] Garousi V., Felderer M., Mäntylä, M.V. 2019. “Guidelines for including grey literature and conducting multivocal literature reviews in software engineering.” Information and Software Technology 106, pp. 101-121

[Lenarduzzi2018] Lenarduzzi, V., Sillitti, A. and Taibi, D., 2018, June. “A survey on code analysis tools for software maintenance prediction.” In *International Conference in Software Engineering for Defence Applications* (pp. 165-175). Springer, Cham.

[Li2015] Li, Z., Avgeriou, P. and Liang, P., 2015. “A systematic mapping study on technical debt and its management.” *Journal of Systems and Software*, *101*, pp.193-220.

[Martini2018] Martini, A., Besker, T., Bosch, J., 2018. Technical Debt tracking: Current state of practice: A survey and multiple case study in 15 large organizations. *Science of Computer Programming* 163, 42–61.

[Rios2018] Rios, N., de Mendonça Neto, M.G. and Spínola, R.O., 2018. “A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners.” *Information and Software Technology*, *102*, pp.117-145.

[OWASP] Online, Access 24/02/2020. https://owasp.org/www-community/Source_Code_Analysis_Tools

[ReplPackage] Online, Access 27/2/2020. https://www.doi.org/10.6084/m9.figshare.12489290

[Kazman2015] 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 Proceedings of the 37th International Conference on Software Engineering - Volume 2 (ICSE ’15). IEEE Press, 179–188.

Authors' Biographies

Dr. Paris Avgeriou is Professor of Software Engineering at the University of Groningen, the Netherlands where he has led the Software Engineering research group since September 2006. Before joining Groningen, he was a post-doctoral Fellow of the European Research Consortium for Informatics and Mathematics. He is the Editor in Chief of the Journal of Systems and Software, as well as an Associate Editor for IEEE Software. He also sits on the board of the Dutch National Association for Software Engineering (VERSEN) and the Dutch research school IPA. He has co-organized several international conferences such as ECSA, ICSA, and Tech Debt and served on their steering committees. His research interests lie in the area of software architecture, with strong emphasis on architecture modeling, knowledge, evolution, analytics and technical debt. He champions the evidence-based paradigm in Software Engineering research and works towards closing the gap between industry and academia. Contact him at p.avgeriou@rug.nl.

Davide Taibi is Associate Professor at the Tampere University, Finland. He obtained his PhD in Computer Science at the Università degli Studi dell'Insubria, Italy in 2011. His research activities are focused on software quality in cloud-based systems, supporting companies in keeping Technical Debt under control while migrating to cloud-native architectures. Moreover, he is interested in patterns, anti- patterns and "bad smells" that can help companies to avoid issues during the development process both in monolithic systems and in cloud-native ones. Formerly, he worked at the Free University of Bolzano, Technical University of Kaiserslautern, Germany, Fraunhofer IESE - Kaiserslautern, Germany, and Università degli Studi dell'Insubria, Italy. In 2011 he was one of the co-founders of Opensoftengineering s.r.l., a spin-off company of the Università degli Studi dell'Insubria. Contact him at davide.taibi@tuni.fi.

Dr. Apostolos Ampatzoglou is an Assistant Professor of Software Engineering, in the Department of Applied Informatics at the University of Macedonia (Greece). Before joining the University of Macedonia, he was an Assistant Professor in the University of Groningen (Netherlands). He holds a BSc in Information Systems (2003), an MSc in Computer Systems (2005) and a PhD in Software Engineering by the Aristotle University of Thessaloniki (2012). He has published more than 80 articles in international journals and conferences, and is/was involved in over 15 R&D ICT projects, with funding from national and international organizations. His current research interests are focused on technical debt, maintainability, reverse engineering, quality management, and design. Contact him at apostolos.ampatzoglou@gmail.com.

Francesca Arcelli Fontana had her Master degree and Ph.D. in Computer Science taken at the University of Milano. She is currently in the position of Full Professor at University of Milano Bicocca. The actual research activity principally concerns the software engineering field. In particular in software evolution, reverse engineering, managing technical debt, and software quality assessment. She is at the head of the Software Evolution and Reverse Engineering Lab at University of Milano Bicocca. Contact her at francesca.arcelli@unimib.it.

Terese Besker is a Ph.D. candidate in the Software Engineering at Chalmers University of Technology in Sweden. She is working in the research fields of technical debt management. Before becoming a Ph.D. student, she had worked as a senior software engineer in the software industry for more than fifteen years. She also has a bachelor's degree in software engineering and a master's degree in applied IT. She has published several peer-reviewed articles in journals, conference and workshop proceedings. Contact her at besker@chalmers.se.

Dr. Alexander Chatzigeorgiou is a Professor of Software Engineering in the Department of Applied Informatics at the University of Macedonia, Thessaloniki, Greece. Since 2017, he serves as Dean of the School of Information Sciences. He received the Diploma in Electrical Engineering and the PhD degree in Computer Science from the Aristotle University of Thessaloniki, Greece, in 1996 and 2000, respectively. From 1997 to 1999 he was with Intracom S.A., Greece, as a telecommunications software designer. His research interests include technical debt, software maintenance, and software evolution analysis. He has published more than 140 articles in international journals and conferences and participated in a number of European and national research programs. He is an Associated Editor of the Journal of Systems and Software and SN Computer Science. Contact him at achat@uom.edu.gr.

Valentina Lenarduzzi is a postdoctoral researcher at the LUT University in Finland. Her primary research interest is related to data analysis in software engineering, software quality, software maintenance and evolution, with a special focus on Technical Debt. She obtained her PhD in Computer Science at the Università degli Studi dell'Insubria, Italy, in 2015, working on data analysis in Software Engineering. She also spent 8 months as Visiting Researcher at the Technical University of Kaiserslautern and Fraunhofer Institute for Experimental Software Engineering (IESE) working on Empirical Software Engineering in Embedded Software and Agile projects. In 2011 she was one of the co-founders of Opensoftengineering s.r.l., a spin-off company of the Università degli Studi dell'Insubria. Contact her valentina.lenarduzzi@lut.fi

Antonio Martini is Associate Professor at the University of Oslo and is a part-time researcher at Chalmers University of Technology. The current focus of Antonio's research is on Technical Debt, Architecture, Technical Leadership and Agile software development. Antonio's experience covers Software Engineering and Management in several contexts: large, embedded software companies, small, web companies, business to business companies, startups. HIs expertise ranges from technical programming to software architecture and software quality, to Agile ways of working and software business. Antonio Martini has worked as Principal Strategic Researcher at CA Technologies for a co-financed project for technology transfer related to Technical Debt and Architecture by the H2020 Marie Skłodowska-Curie grant of the European Union. Antonio has collaborated with several large companies such as Ericsson, Volvo, Saab, Axis, Grundfos, Siemens, Bosch and Jeppesen. He has also started his own consultancy company and has run projects with large companies in north- and central-Europe to manage and visualize Technical Debt. Antonio has been employed as a Postdoc Researcher at Chalmers, after having obtained a PhD in Software Engineering at Chalmers University of Technology, Sweden in 2015. Contact him at antonima@ifi.uio.no.

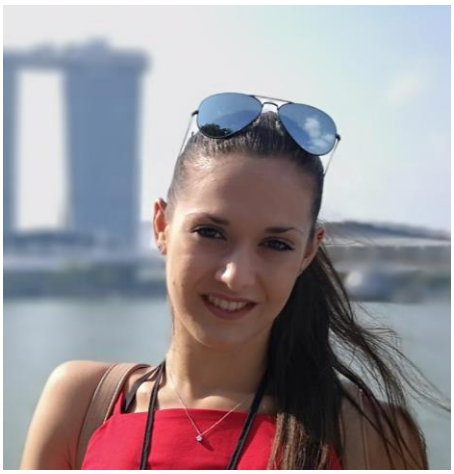
Athanasia Moschou is a Java Software Engineer at Intrasoft International. She received her Bachelor's degree in 2011 in Economic Science, her Bachelor's degree in Applied Informatics in 2016 and her Master's degree in 2018 in Applied Informatics from the University of Macedonia. Her technical interests lie in software development, code quality, and technical debt. Contact her at nasiamoschou@gmail.com.

Ilaria Pigazzini is currently a Ph.D. student in computer science at the Department of Computer Science, Systems and Communications, University of Milano-Bicocca. She has received her B.Sc. and M.Sc. degrees from the University of Milano-Bicocca in Computer Science in 2016 and 2018, respectively. Her research interests include reverse engineering, architectural smell detection and refactoring of Object-Oriented systems. Contact her at i.pigazzini@campus.unimib.it.

Nyyti Saarimäki is a software engineering Ph.D. student at Tampere University, Finland. She received her B.Sc. in mathematics in 2016 and M.Sc. in theoretical computer science in 2018 from Tampere University of Technology. Her main research interests include data analysis and adapting observational study methodologies from epidemiology to empirical software engineering. Contact her at nyyti.saarimaki@tuni.fi.

Darius Sas is a Ph.D. student at the Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, the Netherlands. He received his Master's degree in Computer Science in 2018 from the University of Milano-Bicocca, Milan, Italy. He then started his academic career as a Ph.D. student in the Software Engineering and Architecture (SEARCH) group lead by Paris Avgeriou at the University of Groningen. His Ph.D. project focuses on architectural technical debt elimination in embedded systems and is part of a European project (SDK4ED) that focuses on the interplay between technical debt, energy efficiency, and dependability. Contact him at d.d.sas@rug.nl.

Saulo Soares de Toledo is a Ph.D. candidate at University of Oslo, Norway. His research topics are mainly related to Technical Debt, Microservices and SOA. Before becoming a Ph.D. student, he worked for private companies for more than 13 years, carrying out roles from development to technical leadership and software architecture. He obtained a master's and a bachelor's degrees in Computer Science from the Federal University of Campina Grande (UFCG), Brazil, and a Licentiate degree in Computation by the State University of Paraíba (UEPB), Brazil. Contact him at saulos@ifi.uio.no.



Angeliki-Agathi Tsintzira is a MSc student in School of Electrical and Computer Engineering in the Aristotle University of Thessaloniki, in Greece and a Researcher at the Department of Applied Informatics of the University of Macedonia in Greece. Her research interests include technical debt management, refactorings, software quality and metrics, and software architecture. She previously worked as Researcher at Centre for Research and Technology Hellas (CERTH). She holds an Integrated Master in Informatics and Telecommunications Engineering from the University of Western Macedonia in Greece. Contact her at angeliki.agathi.tsintzira@gmail.com.