A Survey on Code Analysis Tools for Software Maintenance Prediction

Valentina Lenarduzzi[1], Alberto Sillitti[2], and Davide Taibi[1]

[1] Tampere University of Technology, Finland
{valentina.lenarduzzi, davide.taibi}@tut.fi
[2] Innopolis University, Russian Federation
a.sillitti@innopolis.ru

Abstract. Software maintenance is a widely studied area of software engineering that it is particularly important in safety-critical and mission-critical applications where defects may have huge impact and code needs to be checked carefully through the analysis of data collected using a number of tools developed to investigate specific aspects. However, such tools are often not available to practitioners preventing them from applying the most recent and advanced approaches to industrial projects. This paper is an initial investigation about code analysis tools used to perform research studies on software maintenance prediction. We focus on the identification of tools that are available and can be used by practitioners to apply the same maintenance approaches described in published academic papers.

Keywords: software maintenance, tools, software measurement.

1 Introduction

Software maintenance is a deeply studied area with hundreds of studies performed every year and published in top international conferences and journals. In many cases, these studies are empirical ones and based on data collected from a wide range of sources and then analyzed using a variety of mathematical techniques [5]. Moreover, most of the research activities and results presented in such studies are based on the usage of a wide range of software tools.

The code analysis tools used in these studies are very diverse and, in many cases, are ad-hoc developed prototypes used only to perform a single study [5]. In such cases, the developed tools are usually not maintained and are often not even released by the authors. This makes the replication of such studies very difficult and the adoption of the proposed approach nearly impossible for practitioners.

This is a clear limitation of the research activities in the software maintenance area that need to be more open to external validation of the performed researches to push ahead the state of the art and have a real impact on the practice.

As pointed out in a recent analysis of the evolution of the software maintenance research area [5], maintenance models have increased their complexity over the last 40 years using more sophisticated mathematical approaches for data analysis. However,

this increase of complexity does not result in an improvement of performance of the developed models and the external validation of the approaches is almost inexistent, also due to the lack of availability of the tools used to extract and analyze the data. For such reasons, researchers are almost reinventing the wheel in each study and do not leverage on the work performed by others.

To help researchers and practitioners in the identification of available code analysis tools that have been used to publish scientific studies, we have classified them with regard to their support for software maintenance activities. We performed a review of the available tools, reporting both on commercial and open source products. Since the tools cannot be reliably retrieved from bibliographic sources, we adapted the systematic mapping approach defined by Petersen [1] by adding a step specifically intended for the identification of tools. We considered code analysis tools supporting maintenance activities as defined in the ISO/IEC 25010:2011 SQuaRE [4] (modularity, reusability, analyzability, modifiability, testability, reliability).

Only a few previous works have compared existing code analysis tools for specific software maintenance areas [6] [7] [8]. Unlike our work, [6] reports an overview on search-based optimization techniques, proposing four tools for software modularization support. Unfortunately, none of these tools are available anymore.

In [7], the authors compare three static code analysis tools (Fortify SCA, Splint, and Frama-C). However, the comparison focuses on the point of view of detecting security vulnerabilities.

In [8], the authors highlight the issues of detecting Java concurrency bugs using static code analysis tools (FindBugs, CheckThread, RacerX, and RELAY).

Moreover, to the best of our knowledge, no previous works exists that systematically identifies existing and available code analysis tools for software maintenance.

The remainder of this paper is structured as follows: in Section 2, we introduce the adopted methodology; in Section 3 and 4, we present and discuss the results; in Section 5, we point out the threats to the validity of this study; finally, in Section 6, we draw the conclusions and sketch possible future work.

2 Methodology

To perform the survey in a systematic way and achieve results that can be considered valid by the scientific community, we have analyzed in deep the most popular approaches for performing reviews in a coherent and replicable way. In particular, we have analyzed the following techniques: Systematic Literature Review (SLR) [2], Multivocal Literature Review (MLR) [3], Systematic Mapping (SM) [1]. We have realized that none of them are suitable for our study for a number of reasons:

1. SLRs focus only on peer-reviewed publications. However, we have realized that many tools are not described in usual academic papers that are subject to peer evaluations.
2. MLRs give the same level of importance to peer and non-peer reviewed sources. Even if they fit better our goals taking into account all the tools, they

do not provide a specific procedure on how to integrate the results and weight the contributions.

3. SMs are very good for very initial studies and collect basic information in an unstructured way. However, they are too generic and do not provide a rigorous enough framework for the identification of the tools we are interested to include in this study.

Based on such considerations, we have realized that an extension of the SM approach was more suitable to address the problem we face in this investigation (Section 2.2).

2.1 Goal and Research Question

The goal of this study is to identify and classify code analysis tools for supporting software maintenance that have been adopted in industry and validated academically.

Therefore, we define our main Research Questions (RQs):

- RQ1: Which are the most commonly used code analysis tools to support software maintenance? (We aim at classifying the tools)
- RQ2: Have these tools been adopted in research? (We aim at identifying the most popular tools used to perform research activities)

2.2 Research Strategy

In our work, we follow a different procedure compared to the ones commonly adopted in systematic mappings, systematic literature reviews, and multivocal literature reviews since all of them have the limitations described at the beginning of Section 2.

Therefore, the process developed is based on the following steps:

Step 1: Tools Identification. We report the search strategy adopted for identifying the tools available on the web.

Step 1.1: Keywords definition. Based on our RQs, we define the search terms applied for the identification of the tools, as presented in Table 1. We adopted the PICO structure (Population, Intervention, Comparison, and Outcome), skipping the Outcome and Comparison terms, since the focus of our research is a general investigation, as suggested by Kitchenham and Charters [2]. To retrieve a reliable set of tools, we expanded the term "maintenance" with the terms included in the maintenance sub-characteristics defined in the standard ISO/IEC 25010:2011 SQuaRE [4].

Table 1. The search terms.

P: Software maintainability	**P1 terms:** "software", "maintainability", "maintenance", "modularity", "reusability", "analyzability", "modifiability", "testability", "reliability"
I: tool	**I1 terms:** "tool*", "static analysis", "dynamic analysis"

Step 1.2: Bibliographic Sources definition. Differently from the traditional approach for systematic review, instead of searching in bibliographic sources suggested by Kitchenham and Charters [2], we applied the search terms in google.com. This

choice allowed us to broaden the search to non-scientific results to get the most comprehensive list of tools.

Step 1.3: Inclusion and exclusion criteria definition. We defined inclusion and exclusion criteria to identify the most relevant tools. We obtained the final criteria (Table 2) by means of refinements from an initial set of inclusion and exclusion criteria.

Table 2. The search terms.

Inclusion Criteria	Exclusion Criteria
Open source or commercial tool	Tool not available (e.g., not actively developed, never released, etc.) Tool does not directly support maintenance activities (e.g., IDEs) Tools that are not based on source code analysis (e.g., issue tracking)

Step 1.4: Search and Selection process. Two authors separately searched in the selected bibliographic source the tools according to the defined search keywords. They manually checked each retrieved tool by means of visiting the official web page and they applied the inclusion and exclusion criteria. In case of disagreement between the two authors, the third author was involved so as to apply the criteria. The search and selection process returned 25 tools.

From the retrieved 60 tools, we rejected 14 tools related to hardware maintenance, 18 research prototypes, including 14 not available anymore and 4 never mentioned in industrial case studies. Finally, we also rejected three IDEs tools.

Step 2: Popularity Assessment. The popularity of the tools identified in the previous step needs to be evaluated. This is an important step of the procedure, since it allows to consider differently tools that have been used or mentioned online very rarely. The results of the adoption are very important for practitioners: most of the companies cannot risk adopting a prototype or a tool with limited support and/or a too narrow community.

For this purpose, we have considered the trend of searches in the last 5 years using Google Trends and the number of hits reported in google.com. Even if the numbers provided are not absolute, they help us in investigating the turnover.

These two values can be used as proxy for assessing the adoption of a tool by practitioners. Beside a number of Google results does not imply the same number of installation, we can assume that if tool “A” is mentioned 100K more than tool “B”, the tool “A” is used and adopted much more than “B”.

The trend reported in Google Trends, when paired with the number of results, can be a good sign of adoption trend of a tool.

The result of this step is the number of results reported in google.com and its 5-years trend reported in Google Trends (Decreasing, Increasing, Constant, No-Trend).

Step 3: Papers reporting experience using the selected tools. We identified papers reporting the usage of the identified tools to prove the adoption of the tools reported in the scientific literature. Moreover, we investigated existing case studies reporting the usage of the tools in industrial contexts.

To investigate the popularity of the tools in the research community, we adopted the traditional approach based on citations in scientific publications. We analyzed the tool adoption by identifying peer-reviewed publications reporting their usage. The number of reported citations of the tools can be considered as a proxy-measure for assessing the suitability of the tool to support repeatable research investigations that could be performed by advanced practitioners.

Step 3.1: Keywords Definition. In our case, the search terms needed to be defined by combining the search terms identified in Table 1 and the list of tools retrieved.

Step 3.2: Identification of bibliographic sources. A search process can be conducted automatically or manually across specific journals and conferences. To better address this step, we decided to combine both procedures.

Step 3.3: Inclusion and exclusion criteria definition. The results obtained from the search were then filtered by applying the inclusion and exclusion criteria to the title and the abstract. In this step, we excluded research prototype tools identified in the previous step that were never mentioned in the industrial case studies reported in the selected papers, even if they are available and even if they have existing citations in the scientific literature that does not include an industrial case study.

Step 3.4: Search and Selection process. Two authors separately searched in the selected bibliographic source the tools according to the defined search keywords. They manually evaluated each retrieved paper and they applied the inclusion and exclusion criteria. In case of disagreement between the two authors, the third author was involved so as to apply the criteria. The results of this step are reported in Table 3.

2.3 Data Extraction

In this step, we extract the relevant data from the tools that passed the inclusion and exclusion criteria.

3 Results

In the Step 1 of the Search Strategy approach (Tools identification), we retrieved 60 tools. Thanks to the inclusion and exclusion criteria we rejected 35 tools, resulting in 25 selected tools for this review (Table 3).

About the rejected tools, they are all research prototypes and in most of the cases executables or source code are no longer available. Unfortunately, some research prototypes that are also mentioned in industrial case studies are not available anymore (PROM [11] [13] [14] and RIGI [12]), while the four available tools are rarely mentioned in research works and never in industrial case studies.

Table 3. The tools classification.

Tool	Goal	Supported Language	License	Cit.	Google Results	Trend
CodeSonar	Code review	Java, C, C++	Comm.	347	56,500	=
Findbugs	Bug detection	Java	LGPL	346	656,000	↓
Coverity	Bug detection/Testing	Java, C, C++	Comm.	315	235,000	=
PMD	Bug detection/Testing	Java	LGPL	238	502,000	=
Polyspace	Run-time errors	C, C++	Comm.	212	110,000	=
Checkstyle	Coding standards	Java	LGPL	155	438,000	↓
Klocwork	Safety, reliability	Java, C, C++, C#	Comm.	132	79,400	↓
Parasoft Jtest	Testing	Java. C, C++, .NET	Comm.	114	548,000	=
Squale	Code review	Java, C, C++, .NET, PHP, Cobol	LGPL	110	291,000	↑
IBM AppScan	Testing, Security flaws	Java	Comm.	86	364,000	=
JLint	Code Review	Java	LGPL	86	141,000	=
SonarQube	Code review	All Comm. Lang.	LGPL + Comm.	74	417,000	↑
Lattix	Technical debt, Modularity, Reusability	Java, C, C++	Comm.	69	93,700	↓
Fortify Static Code Analyzer	Code analysis, vulnerability detection	All common lang.	Comm.	50	581,000	=
ConQAT	Code review	Java, C#, C++, ABAP, ADA	Comm.	37	25,800	=
Ndepend	Dependency analysis	.NET	Comm.	35	85,600	↓
CAST	Code review	All Comm. languages	Comm.	24	134,000	↑
Structure101	Architecture analysis	Java, .Net	Comm.	24	131,000	=
LDRA testbed	Dynamic code analysis	java	Comm.	20	348,000	=
Axivion Bauhaus Suite	Reverse engineering and architecture recovery	Ada, C, C++, C# and Java	Comm.	3	1,710	=
source meter	Code review	Java, C/C++, C#, Python and RPG	Freeware and Comm.	2	262,000	=
Jarchitect	Structural code analysis	Java	Comm.	2	65,700	=
Imagix	Structural code analysis	Java, C, C++	Comm.	1	399,000	=
Codacy	Code review	JavaScript, Scala, Java, PHP, Python, CoffeeScript, CSS, Ruby, Swift, C/C++.	Comm.	1	83,100	↑
Parasoft dotTEST	Code review	C/C++, Java, .NET	Comm.	1	25,200	↓

3.1 RQ1. Which are the most commonly used tools to support software maintenance?

The most mentioned tools in google.com are Findbugs, Fortify, JTest, and PMD, all above 500K citations. However, analyzing the trend of the results in Google, no significant changes emerged or, at least, the tools with an increasing trend still have a very low number of results compared with the remaining tools. The only partial exception is SonarQube with over 400K citation and an increasing trend. This result also supports the importance of the tools reported.

To understand the purpose of this tools, we classified them based on the maintenance activities they can support.

Code Review is one of the most often considered activities by the selected tools, with 33% related tools, mainly for code understandability, support for code inspection, and error prediction. Reverse Engineering, Structural Code Analysis, and Testing are also popular activities, with 12.5% of the tools supporting each of them. Moreover, other tools focus on specialized activities such as Dynamic Code Analysis, Analysis of Technical Debt, Modularity and Reusability, and Security Flaws Analysis.

Taking into account the license of the selected tools, 83% of them are distributed with a commercial license, while the remaining ones are freely available with open source licenses (mainly LGPL).

Considering the supported programming languages, Java is the most frequently considered one, immediately followed by C and C++.

3.2 RQ2. Have these tools been adopted in research?

The selected tools have an average of 90 citations. However, the average is highly pushed up by the first three tools (CodeSonar, FindBugs, and Coverity) with more than 250 citations, which together account for nearly half of all citations. Fifteen tools have a number of citations below the average, with six tools having fewer than three citations.

Several academic works adopt tools with a very low number of results in google.com. As example CodeSonar and Polyspace are used by several research works but they are not well known or with a very low number of Google results. Moreover, these tools are mainly applied in non-industrial case studies.

This confirms the low level of adoption of several tools in academic works, including industrial case studies. In 60% of the cases, the interest in such tools (according to Google Trends) is almost constant, while in 16% of the cases is increasing and in 24% of the cases is decreasing. Therefore, we can conjecture that in most of the cases users could be satisfied with the tools they use or there are no better alternatives they can consider.

4 Discussion

Focusing on the two research questions, we have found out that the tools commonly used by practitioners and by researchers are very different, with only a single common tool among the most popular in the two cases: FindBugs.

We have also analyzed if there is a correlation between then citations in google.com and the ones in the scientific literature but the results were not significant. Even accepting an unusually low level of confidence, the correlation is very weak (0.35) showing that the two sets of citations are almost completely unrelated from each other.

These facts could be due to several reasons that need further investigation (e.g., difficulty of usage, different objectives, etc.). However, it is quite clear that the practitioners' and the researchers' communities value different tools to perform their work.

In any case, the most popular tools in research have a constant or negative trend for practitioners, the same happens for the most popular tools for practitioners. This is a clear evidence that neither set of tools are actually able to satisfy practitioners and there are no clear alternatives. The increasing trends are very limited and focused on tools with a low level of popularity, this could be related to the fact that they are being tested by practitioners, but such tools are not very satisfying and/or being able to be adopted by a large set of users.

We can also notice that almost all the tools identified are commercial and the availability of open source ones is very limited. In particular, the most used tools in research are available for free (as open source or with a free license for a subset of the features) while most of the tools used by practitioners are commercial. Since most researchers have limited budgets, it may happen that the features and the quality of the tools could be considered of less importance compared to the free availability. This may affect their usage in research projects and their citation in research papers. However, this is an aspect that needs further investigation.

Another aspect that needs to be considered is the lack of reliable maintenance models developed by researchers 5. Nearly all the models available are specifically developed to address a very limited set of projects (in many cases, such projects are proprietary and results cannot be replicated) and they lack of external validation, preventing practitioners from applying such models in their specific contexts.

Such lack of reliable results from the research produces also a lack of tools that practitioners can really use in their daily work. Looking at the excluded tools, we can notice that nearly all of them have a very low number of citations. This means that almost no research is based on them and they have been developed and used only for a limited number of projects (often just one) with the objective of writing a paper and not to create a tool able to support the adoption of a model by the community.

5 Threats to validity

As suggested by Yin [9], we defined Internal validity, Construct validity and Reliability. Since we do not draw any conclusions about mapping studies or systematic literature reviews in general, external validity threats are not applicable.

Moreover, Petersen et al. [10] suggest assessing the quality of a study by means of profiling an objectively checklist. We reached an excellent score of 72%, higher than the average (33% - 48%) of similar studies [10].

Internal validity: Our study does not draw cause-effect relationships. Moreover, since our analysis only uses descriptive statistics and basic correlation, the threats are minimal. However, we understand that the identification of the citations based on scientific literature might only reflect a portion of the adopted tools, while the analysis of gray literature could have provided different results, somehow promoting more tools with a bigger community or with a longer history, without considering their quality.

Construct validity: The measure of popularity could be of this threat. At the best of our knowledge there is no way to collect the exact number of users that use a specific tool, and therefore the measures we considered as proxy measures, could have affected the results. The terms adopted are well known and stable enough to be used as search strings.

Reliability: We defined search terms and applied procedures that can be replicated by others. However, we are aware that the same search string applied in google.com could return a slightly different set of results that can become relevant in a few years or even in a few months. It could be interesting to monitor the evolution over time.

6 Conclusions and future work

This paper provides an initial analysis and classification of the static and dynamic code analysis tools for supporting and predicting software maintenance that have been adopted in industry and validated academically.

The study focuses on tools that are not just research prototypes and that are available to support studies in academic and industrial settings. In particular, we have found out that most of the tools used in industrial case studies were no longer available at the time of writing, preventing other researchers or practitioners from easily adopting the same approaches and analyses by using already existing and tested tools.

Moreover, most of the widely-adopted tools are commercial ones and there are almost no open source communities that are able to build and support such kinds of tools. Research prototypes often have a large number of citations in scientific literature but are never adopted in industrial case studies. In most of the cases (60%), the interest in such tools is quite constant over the years providing a resistance in the dismissal of tools or in the adoption of new ones.

An interesting side result is that a lot of tool prototypes for reliability prediction have been developed in the past by researchers, but their usage in industry is very limited. Moreover, no commercial tools for reliability prediction were identified from the selected toolset. The results of the classification of the selected tools reported in Table 3 can be used by researchers to select the most frequently used tools to obtain reliable results for their research, to conduct empirical studies, or to perform other work. Moreover, results can be beneficial for industry practitioners in to easily access to a classification of existing tools used to perform maintenance research studies.

Another aspect is that almost all the selected tools have been designed to support developers in performing some kind of activities (e.g., code reviews, testing, static analysis, etc.) but they do not include maintenance models to help users in making estimates/predictions. On the contrary, the rejected tools include several ones that implement such models. Moreover, the tools implementing a model are the least cited ones (almost no citations at all). This means that the research activities performed are not basing their work on existing results and tools, but new ones are developed. This is a sign of immaturity of the field and the need of further research in this area to create reusable models and tools that can be adopted by practitioners.

References

1. K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson, “Systematic Mapping Studies in Software Engineering”, International Conference on Evaluation and Assessment in Software Engineering, 2008
2. B. Kitchenham, S. Charters, “Guidelines for Performing Systematic Literature Reviews in Software Engineering,” Version 2.3, 2007
3. V. Garousi, M. Felderer, M. V. Mäntylä. “The need for multivocal literature reviews in software engineering: complementing systematic literature reviews with grey literature.” 20th International Conference on Evaluation and Assessment in Software Engineering, 2016.
4. ISO/IEC 25010:20111 SQuaRE. Available online: https://www.iso.org/standard/35733.html.
5. V. Lenarduzzi, A. Sillitti, D. Taibi, “Analyzing Forty Years of Software Maintenance Models”, 39th International Conference on Software Engineering (ICSE 2017), Buenos Aires, Argentina, 2017.
6. G. Bavota, M. Di Penta, R. Oliveto. “Search Based Software Maintenance: Methods and Tools”, Evolving Software Systems.
7. M. Mantere, I. Uusitalo, J. Röning, “Comparison of static code analysis tools”, 3rd International Conference on Emerging Security Information, Systems and Technologies, SECURWARE, 2009.
8. N. Manzoor, H. Munir, M. Moayyed, “Comparison of static analysis tools for finding concurrecy bugs”, 23rd IEEE International Symposium on Software Reliability Engineering Workshops, 2012
9. R.K. Yin. Case Study Research: Design and Methods. SAGE Publications, 2009.
10. K. Petersen, S. Vakkalanka, L. Kuzniarz, ”Guidelines for conducting systematic mapping studies in software engineering: An update”, Information and Software Technology, 64, 2015.
11. A. Sillitti, A. Janes, G. Succi, T. Vernazza, “Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data”, 29th Euromicro Conference, 2003.
12. H.M. Kleine, H.A. Muller, “Rigi-An environment for software reverse engineering, exploration, visualization, and redocumentation”, Science of Computer Programming, 75(4), 2010.
13. I. Coman, A. Sillitti, “An Empirical Exploratory Study on Inferring Developers’ Activities from Low-Level Data”, *19th International Conference on Software Engineering and Knowledge Engineering (SEKE 2007)*, Boston, MA, USA, 9 - 11 July 2007.
14. I, Coman, P. N. Robillard, A. Sillitti, G. Succi, “Cooperation, Collaboration and Pair-Programming: Field Studies on Backup Behavior”, *Journal of Systems and Software*, Elsevier, Vol. 91, No. 5, pp. 124 - 134, May 2014.