An Open Tool for Assisting in Technical Debt Management

Carlos Fernández-Sánchez*, Héctor Humanes†, Juan Garbajosa† and Jessica Díaz†
*Universidad Politécnica de Madrid, CITSEM, Madrid, Spain
Email: carlos.fernandez@upm.es
†Universidad Politécnica de Madrid, CITSEM and ETSISI, Madrid, Spain

***Abstract*—Technical debt monitoring is one of the activities that have to be performed in technical debt management. To do that, there are different techniques that can be used to estimate technical debt and different tools that implement those different techniques. This paper presents *TEDMA Tool*, a tool for monitoring technical debt over the software evolution and that it is open to integrate third party tools. TEDMA is based on the analysis of source code repositories and is useful for researching using empirical data extracted from software projects. Currently, it is been used to analyze big projects in the execution of several case studies. The expected evolution of TEDMA will make the tool useful for software development industry.**

I. Introduction

Technical debt (TD) monitoring is one of the activities in technical debt management (TDM) [1]. To monitor TD, it is necessary to analyze the change of TD indicators over the software evolution. For TDM, it is also needed to take into accout several requirements, including different points of view [2], [3]. This implies to obtain data from several sources and to apply several techniques. There are many possible metrics and techniques to be used to support TDM requirements [4]. This variability of techniques is extended to the approaches for TDM [5]. Therefore, it is compelling to have means to combine metrics, techniques, and management approaches implemented by different tools, considering also that current research in TD [3] is resulting in new metrics and techniques, that should have to be considered when they become available. Also, there are some tools that provide different TD indexes [6]. It must be noted that it is not completely clear in literature which of the current metrics must be selected, or how they must be combined, since they might be indicators of different types of TD [7].

Thus, there is a necessity of tools that integrate different techniques and other tools for analyzing changes on TD over the software evolution. This would facilitate the research activity using empirical data obtained from software projects. This kind of tools are necessary because they allow the collection of evidences to support research [8]. Software development organizations could also take advantage of such tools to define models for TDM and analyzing the impact of TD in their projects.

Tools used to manage TD are focused on specific requirements of TDM [4]. Existing tools are mainly focused on obtaining metrics, implementing some specific techniques for TDM, or providing their own models for TDM [4] and, as described in [3], each individual tool does not support all the required elements for TDM and, consequently, it is necessary to integrate them. As discussed in Section III, current tools are often standalone and not devised to integrate other tools.

Though, it is challenging to integrate tools that are not thought to be integrated with other tools. Special attention has to be paid to the often different tool analysis-strategies. For example, a tool can be focused on extracting data from the last version of the software while other tool can consider all changes in the history of the project. Ignoring these differences will lead to wrong conclusions when results are compared.

TEDMA is an open tool created to assist in TDM by means of the execution of TDM metrics and techniques implemented by third party tools, and taking into account the historical evolution of the software. It is open in the sense of being designed to integrate third party tools and developments. TEDMA allows to analyze the evolution of the metrics over the software evolution. It facilitates the integration of new tools and has been designed to support different TDM model implementations. The information that it generates is stored in databases that can be exploited by external tools. This includes to access specific information using query languages. Data are organized following the software evolution, and therefore TEDMA facilitates the implementation of efficient algorithms to traverse the history of files and revisions. As described in following sections, a first version of TEDMA has been implemented and has been tested analyzing some large open source projects. TEDMA analyzes release by release, starting from the beginning of the project, to create a graph database with the evolution of each file. Additionally, it gathers metrics of the files and revisions. This information can be enriched by the execution of analyzers. An analyzer is an abstraction of any tool or technique used to obtain relevant data for TDM. When the analyzer is executed, its outputs are linked to a graph database as it is described in Section II.

II. TEDMA Tool Description

A. Overall View

To understand the architecture of the TEDMA tool it may help to, firstly, describe the process to analyze a project. Figure 1 depicts the life cycle of a project in TEDMA. A project has to be added to the tool by providing a name, a description, and the location of the source code repository

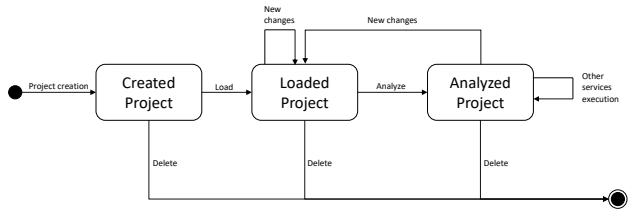

Fig. 1. Life cycle of a project in TEDMA

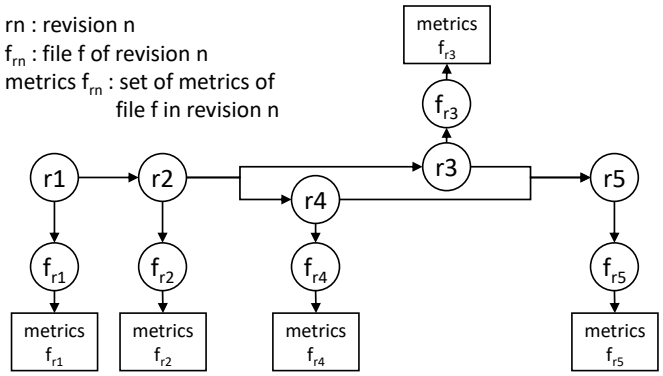

Fig. 2. Basic data structure

(local or remote). After that, the next step is to load the basic project data into the tool's database. This information is mainly source code data about changes in files over the software evolution. With this basic information, the project can be analyzed by any of the available analyzers. When new changes are uploaded to the repository, those changes can be loaded to the tool and analyzed. At any moment a project can be removed from TEDMA for releasing resources.

Figure 2 depicts how the information is stored in TEDMA. TEDMA stores information for each revision and for each file in each revision. For each file several metrics are stored. Example of metrics are basic size metrics of each file as size in bytes, number of lines, and, if the file has changed, the type and size of the change. Depending on the analyzers executed, different metrics can be stored. Each analyzer prescribes how its output is stored. For example, the current implementation of the PMD analyzer [9], one of the tools integrated, includes a problem for each file in which PMD detects problems.

Additionally, TEDMA ensures that any change or action over the files is considered just one time. This is important in merging revisions where changes could be considered twice if they are not carefully checked. Even when TEDMA is currently focused on file level, because of the flexibility of its data model, it can be extended to use other abstraction levels such as modules and methods. This flexibility facilitates the integration of tools focused on different types of TD as code TD and architectural TD.

B. Obtaining information from projects

Currently, TEDMA can gather information about the evolution of each file in a project, and this information is mainly

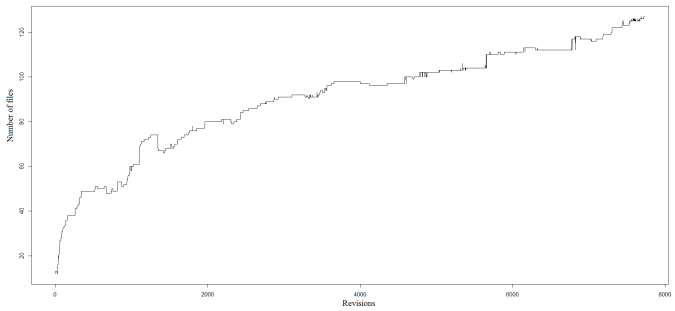

Fig. 3. Evolution of the number of files with cyclomatic complexity issues in Apache Log4j 2 project

obtained from Git repositories [10], PMD [9] detected code smells, and Findbugs [11] detected problems. Both, PMD and Findbugs analyzers are limited to analyze Java projects. In the case of Findbugs, each release has to be compiled, so it is necessary to have installed additional tools to build the project, for example Maven and Gradle. Other metrics have been directly implemented, for instance, the probability of change and expected size of change as are defined in [7]. Data are collected for all the releases of the software so that metrics evolution can be analyzed.

Figure 3 shows the evolution of one metric: the number of files with at least a method with a cyclomatic complexity issue, as it is detected by the default PMD configuration in Apache Log4j 2 project. This type of time series provides information about how some problems are evolving in the analyzed projects. In this case, it seems that the project is getting more complex over time. This type of analysis can help to identify potential TD problems over the software evolution. Figure 3 is generated, in that case, using R [12] to generate the graphical representation from the data exported by TEDMA. R is a free software environment for statistical computing and graphics.

In the following sections it is explained how R has been also used for other purposes in TEDMA.

C. Processing information from projects

Currently, the TEDMA core components are already implemented. TEDMA is being used to analyze open source projects on GitHub that are widely used in the software development industry. Examples of already analyzed projects are Spring-Framework, Karaf, Log4j 2, Hbase, and Hadoop. Thanks to these analyses, and that projects are really large, it was possible to test the capabilities of TEDMA. In this sense, TEDMA has proved helpful to investigate metrics and models for TDM. It provides empirical data obtained from real projects used in the software development industry. The results of these analyses will be published separately.

The tool provides access to data at different abstraction levels that can be used together when needed. The most abstract level is the entity level (project, revision, file, change, etc.). At that level of abstraction, the programmer can manage all this concepts without thinking in the source code repository

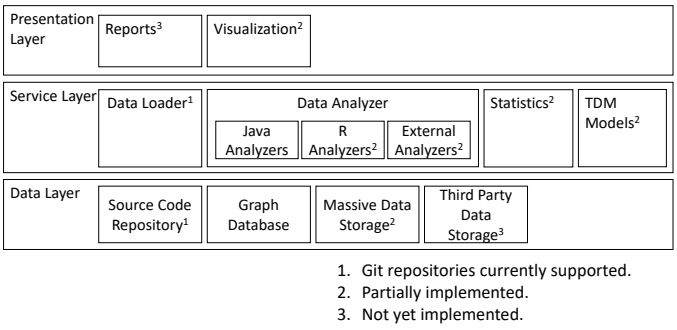

Fig. 4. Modules of TEDMA

or the database. If it is needed, the TEDMA API supports the direct usage of the Neo4j [13] API to manage the graph database and the JGit [14] API to manage the Git repository.

D. How TEDMA is built

In this section we describe the main modules of TEDMA(see Figure 4).

1) Data Layer: TEDMA uses several means to obtain and store projects data. The main data source is the source code repository. Currently, TEDMA only supports Git repositories. For each project, TEDMA clones the source code repository to be used in the whole life cycle of the project TDM.

After the source code repository, the most relevant data are stored is the graph database. For each project, TEDMA creates and maintains a graph database that represents the whole software evolution, including all the source files evolution. This graph allows to access the information following the evolution paths of the software. Therefore it can be traversed using releases, revisions, and files following any combination of the stored relationships of such nodes. In fact, the graph database acts as an index for the rest of the information. If any node requires large amount of data to be stored, an additional storage is used to avoid overcharging the graph with data that it is only required in specific analysis or reports. Additionally, it is planned to provide means to use third party data storage by including extensions for other data sources, for example, external metrics databases.

2) Service Layer: TEDMA provides a set of services that can be used in the TDM process. The first service is Data Loader. This service is in fact the service that incorporates a project to the tool to be analyzed.

The data analyzer service allows to implement and execute different analyzers in the projects. As it was said before, an analyzer is an abstraction of any tool or technique used to obtain relevant data for TDM. Analyzers can be implemented using Java and R. Currently, two analyzers are available to use PMD [9] and Findbugs [11] to analyze projects. Other metrics have been directly implemented, for instance, the probability of change and expected size of change as are defined in [7]. All these implementations were done to demonstrate TEDMA integration capacity.

The statistics service is based on the integration of R in the core of the TEDMA. This facilitates the usage of R scripts and incorporates all the power of R to perform statistical analysis over the collected metrics. Currently, only a basic version exists. R can be used into the TEDMA using two different approaches, Renjin [15] and Rserve [16].

Technical debt management models service addresses the goal of testing and developing TDM models that help decision making about TD. Models can be defined in Java or R languages. Currently, a model for TDM is implemented in R, but not completely integrated into TEDMA (the model will be published separately). To be used it is necessary to export project data from TEDMA using an adhoc export module, and after that, it is possible to use the model from R directly.

3) Presentation Layer: The reports service, which is not currently available, will be focused on the elaboration of automatic reports to show the results of the analyses. The reports will be oriented to different stakeholders roles needs.

The visualization service is in charge of showing dashboards and graphical representations of the results. It is related to the reports service but focused on interactive means of visualization of the information. This service is currently under development.

E. Integration of third party tools

So that it can be extended, TEDMA provides one API to manage all the information stored in the Data Layer. The API allows to work with Git repositories using internally JGit [14]. The graph database is implemented using Neo4j [13]. The API provided by TEDMA uses internally Neo4j and it has different levels of abstractions that allow to work with files, revisions, projects, etc., including the usage of queries using CYPHER language. This provides a large flexibility when new extensions have to be created. If it is needed, the TEDMA API allows to use the Neo4j and JGit APIs directly. This API can be used to extend any of the services provided by TEDMA or implement new ones.

F. TEDMA Tool Roadmap

The great advantage of TEDMA is that it can be integrated with other tools to obtain metrics from many different sources. At the moment TEDMA is used to analyze projects with thousands of revisions and thousands of files in each revision. This demonstrates that the core of TEDMA is ready for the analysis of real software to extract metrics over their evolution. The next planned integration is with SonarQube to take advantage of the great number of metrics that this tool can obtain. Additionally, SonarQube is widely used in software development industry, so this integration will help to use TEDMA in any development environment where SonarQube is currently been used.

Summing up, in the next months it is planned to extend TEDMA to implement all the modules described in Figure 4. The next specific features that will be included in TEDMA are: i) support for SVN source code repository; ii) integration with SonarQube; iii) full integration of R; iv) visualization

service; and v) report service. When all these features are implemented TEDMA will be released. At that point TEDMA will provide software development organizations with a high value by allowing them to use already implemented TDM models and by allowing them to implement their own models. The goal is to have a tool that can be used to generate reports and dashboards without a deeper knowledge of how TEDMA is working internally.

III. Related Work

To the knowledge of the authors there is not other tool with the goal of integration of TDM tools. The available tools are focused on implementing metrics, specific techniques for TDM, or both of them. Techniques and tools for TD have been analyzed in literature reviews previously [4], [1], [3]. The goal of this tools is not to be a quality tool that implements metrics to keep the code quality. TEDMA pursues to manage TD by integrating other available tools that implements those metrics or TD approaches.

One of the most used tool to analyze code is SonarQube [17]. This tool integrates many code metrics and is widely used in software projects. It is based on the static analysis of source code. The goal of the tool presented in this paper is to analyze the evolution of code, and to compare different techniques to analyze software. Therefore, one future work will include the integration with SonarQube to use it as another source of metrics for TEDMA. Other tool, Titan [18], which provides mechanisms to estimate TD and that also provides a framework to make decisions based on these estimations, could be also integrated into TEDMA. Titan [18] is mainly focused on the analysis of modularity violations, similar to Clio [19], and a model for TDM. As the goal of TEDMA is to integrate both metrics and management models, the integration of a tool as Titan could be a good example of the capabilities of TEDMA. Similar tools that could be integrated in the future within TEDMA are Structure101 [20], Sonargraph [21], Lattix [22], and Arcan [23].

IV. Conclusions

This paper presents TEDMA. TEDMA is a tool to analyze software projects with a perspective of evolution. It is thought to work with projects with thousands of revisions and thousands of files per revision. To do that, it supports the integration of third party tools for TDM in order to use them in the analysis of the evolution of TD of software. It can be extended to include additional tools or to implement specific metrics. With these capabilities, TEDMA is a useful tool for researching using empirical data extracted from software projects. Currently, it is been used to analyze large projects in the execution of several case studies.

TEDMA provides a way to experiment with different metrics required for TDM. TEDMA helps to analyze how these different techniques work in specific projects. This is important for software developers because it allows them to choose, from all the available tools that calculate metrics for TDM, those that are the most useful for their projects.

Using TEDMA, organizations will be able to manage the TD of their projects by selecting the tools and indicators that are the most important for them.

Acknowledgment

This work was partially sponsored by MESC DPI2013-47450-C2-2-R (Spain) and by CrowdSaving TIN2016-79726-C2-1-R (Spain).

References

[1] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *J. Syst. Softw.*, vol. 101, no. C, pp. 193–220, Mar. 2015.
[2] D. Falessi, M. Shaw, F. Shull, K. Mullen, and M. Keymind, "Practical considerations, challenges, and requirements of tool-support for managing technical debt," in *International Workshop on Managing Technical Debt (MTD)*, 2013.
[3] C. Fernández-Sánchez, J. Garbajosa, A. Yagüe, and J. Perez, "Identification and analysis of the elements required to manage technical debt by means of a systematic mapping study," *J. Syst. Softw.*, vol. 124, pp. 22 – 38, 2017.
[4] C. Fernández-Sánchez, J. Garbajosa, C. Vidal, and A. Yagüe, "An analysis of techniques and methods for technical debt management: a reflection from the architecture perspective," in *SAM 2015*, 2015.
[5] N. S. Alves, T. S. Mendes, M. G. de Mendona, R. O. Spnola, F. Shull, and C. Seaman, "Identification and management of technical debt: A systematic mapping study," *Information and Software Technology*, vol. 70, pp. 100 – 121, 2016.
[6] F. A. Fontana, R. Roveda, and M. Zanoni, "Technical debt indexes provided by tools: A preliminary discussion," in *2016 IEEE 8th International Workshop on Managing Technical Debt (MTD)*, 2016.
[7] N. Zazworka, A. Vetro, C. Izurieta, S. Wong, Y. Cai, C. Seaman, and F. Shull, "Comparing four approaches for technical debt identification," *Software Quality Journal*, vol. 22, no. 3, pp. 403–426, 2014.
[8] F. Shull, D. Falessi, C. Seaman, M. Diep, and L. Layman, "Technical debt: Showing the way for better transfer of empirical results," in *Perspectives on the Future of Software Engineering*, J. Münch and K. Schmid, Eds. Springer Berlin Heidelberg, 2013, pp. 179–190.
[9] Pmd web project. [Online]. Available: https://pmd.github.io/
[10] Git web page. [Online]. Available: https://git-scm.com/
[11] Findbugs project. [Online]. Available: http://findbugs.sourceforge.net/
[12] The r project. [Online]. Available: https://www.r-project.org/
[13] Neo4j web project. [Online]. Available: https://neo4j.com/
[14] Jgit web project. [Online]. Available: https://eclipse.org/jgit/
[15] Renjin web project. [Online]. Available: http://www.renjin.org/
[16] Rserve web project. [Online]. Available: https://rforge.net/Rserve/
[17] Sonarqube web project. [Online]. Available: https://www.sonarqube.org/
[18] R. Kazman, Y. Cai, R. Mo, Q. Feng, L. Xiao, S. Haziyev, V. Fedak, and A. Shapochka, "A case study in locating the architectural roots of technical debt," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2, May 2015, pp. 179–188.
[19] S. Wong, Y. Cai, M. Kim, and M. Dalton, "Detecting software modularity violations," in *2011 33rd International Conference on Software Engineering (ICSE)*, May 2011, pp. 411–420.
[20] Structure101 web page. [Online]. Available: http://structure101.com/
[21] Sonargraph web page. [Online]. Available: http://www.hello2morrow.com/products/sonargraph
[22] Lattix web page. [Online]. Available: http://lattix.com/
[23] F. A. Fontana, I. Pigazzini, R. Roveda, and M. Zanoni, "Automatic detection of instability architectural smells," in *IEEE International Conference on Software Maintenance and Evolution*, 2016.