Towards Studying the Evolution of Technical Debt in the Python Projects from the Apache Software Ecosystem

Jie Tan
University of Groningen
Groningen, The Netherland
j.tan@rug.nl

Mircea Lungu
IT University of Copenhagen
Copenhagen, Denmark
mircea.lungu@gmail.com

Paris Avgeriou
University of Groningen
Groningen, The Netherland
paris@cs.rug.nl

***Abstract*—The topic of technical debt has gained significant attention from researchers in recent years since its management has significant impact of software development. Several studies that analyze technical debt evolution from different perspectives; however since most of these studies are done for Java very little is known about the evolution of technical debt in software ecosystems consisting of projects written in other languages.**

In this paper we run a study across nine Python open-source software projects belonging to the Apache Software Foundation to investigate the amount of technical debt that is paid back. To measure technical debt we use one of the standard tools in industry: SonarQube. We investigate the impact of using the 28 default rules of SonarQube for Python versus using an extended set of 208 rules to detect instances of technical debt.

***Index Terms*—Software Evolution, Technical Debt, Software Ecosystems**

I. Introduction

Technical debt is a metaphor introduced by Ward Cunningham to explain the unavoidable interests developer pay while getting the short-term benefits [1] [2].

There have been several studies that analyzed technical debt for developing techniques and tools to detect specific types of technical debt, such as code smells [3] [4] or self-admitted technical debt [5] [6] [7] [8].

Olbrich et al. [9] investigated two code smells and analyze historical data over several years of development of two large scale open source systems.

Software ecosystems are groups of projects that co-evolve together in the same environment [10]. Although there has been a flurry of research activity in studying software ecosystems, there has been little work focusing on analyzing the evolution of technical debt at the ecosystem level. In one of the few such works, Digkas et al. [11] studied the evolution of technical debt in 66 Java projects over a period of 5 years. They used SonarQube to investigate how technical debt evolved and what types of issues included. In a follow up work, they selected 57 Java projects from the Apache ecosystem, focusing on the amount of technical debt that is paid back and the issues that are fixed [12].

However, to the best of our knowledge, there is no study focuses on technical debt evolution in a Python ecosystem. Given the pre-eminence of the Python programming language at the current moment and the lack of technical debt studies for systems written in the language, we decide to extend the previous work of Digkas which analyzed the Java projects from the Apache ecosystem [12] by analyzing the Python projects from the same ecosystem.

II. Study Design

Various techniques have been proposed for detecting technical debt. However, since our work does not focus on the detection aspect, we use SonarQube, a third party tool for this purpose. The tool is one of the most popular industrial tools for source code technical debt measurement and reporting and has been used also in the study of the Java systems in the Apache ecosystem.

The goal of our study is to investigate the types of issues that have been fixed during the evolution of Python projects in the Apache ecosystem and the differences between Java and Python, also comparing the types of technical debt between default rules and extended rules (rules for Python debt that must be manually enabled in SonarQube). To this end, we ask two research questions:

1) *What is the difference of technical debt types between default rules and extend rules?* This RQ aims to find the change results after using SonarQube plugins, and to explain why we should use the extended rules.
2) *What is the fixing prevalence of different issue types?* This RQ investigates how prevalent the technical debt is in open source systems.

A. Project Selection

The GitHub repository of the Apache Software Foundation contains projects in more than 30 languages. 38 out of 1574 Apache projects are written in Python. We use two elements to select Python projects from the Apache ecosystem:

1) *Size:* at least 100 classes
2) *Evolution:* at least two years and 1000 git commits

Table 1 shows the nine projects that are left after applying these inclusion criteria:

TABLE I
SELECTED PROJECTS

Project Name	Classes	Commits	Last Commit	First Commit
allura	1,195	9,026	Feb 2018	Oct 2009
qpid-python	385	1,000	Mar 2018	Sep 2006
cassandra-dtest	460	5,074	Mar 2018	Sep 2011
infrastructure-puppet	122	8,863	Mar 2018	Jul 2015
libcloud	1,319	5,981	Mar 2018	Jul 2009
incubator-superset	296	2,846	Mar 2018	Jul 2014
incubator-mxnet	638	6,755	Mar 2018	Apr 2015
bloodhound	1,068	1,238	Feb 2018	Jan 2012
incubator-airflow	638	6,755	Mar 2018	Oct 2014

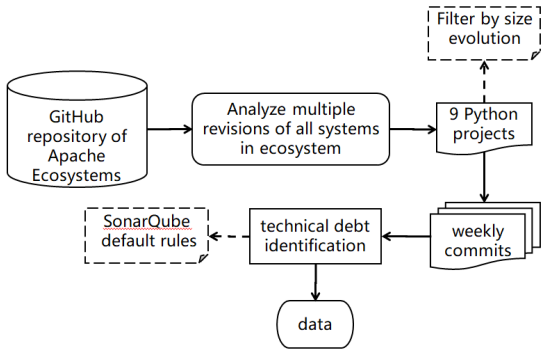

Fig. 1. Data extraction process

B. Data Extraction

To study the evolution of a large number of systems over an extended period, one must make a trade-off between the granularity of the analysis and the feasibility of the analysis (e.g. analyzing all the versions of all the nine selected systems with SonarQube could easily take many months since the tool has no possibility of performing an incremental analysis, but instead, with every commit the system is fully analyzed as a whole). In this study, we analyze weekly versions of each project.

Figure 1 shows the main steps of the data extraction process that we follow in this work:

1) We clone the GitHub repositories of the Python projects in the Apache Ecosystem and analyze the version repository of all the cloned systems
2) We use the two criteria of size and evolution to select Python projects from the Apache ecosystem. We are left with 9 projects.
3) We analyze the entire change history of each project with a weekly resolution by using SonarQube with the default rules (28 rules)
4) We also analyze the history of every project a second time with an extended set of all the possible rules for Python (208 rules).

In this context we want to study the evolution of technical debt fixes.

III. PRELIMINARY RESULTS

Analyzing multiple versions of a project with SonarQube is a very time consuming activity, since SonarQube can not reuse results from the analysis of a previous version, so the time required for analysis is proportional to the number of versions that are being analyzed. This slows down the speed with which the analysis can be done. This is why in this remainder of this paper we only look at one project: qpid-python but we plan to update the report as the results from the other systems are flowing in.

A. What is the difference of technical debt types between default rules and extend rules?(RQ_1)

Figure 2 depicts the relationship between three parts of the SonarQube rules:

- The grey part shows 434 (extended) Java rules
- The blue part shows 208 (extended) Python rules
- The red part shows 28 default Python rules

When we used the extra plugins of SonarQube and added all the corresponding rules, the number of rules that appear both in Python and Java increased to 27. Thus, using the SonarQube plugin will increase the comparability and we encourage other researchers to do it.

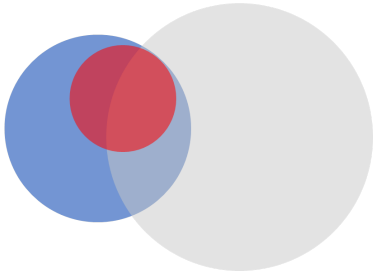

Fig. 2. Data extraction process (grey part shows 434 extra rules for Java; blue part shows 208 extended rules for Python; red part shows 28 default rules for Python

TABLE II
DEFAULT AND EXTRA RULES

Project Name	Unresolved		Fixed	
	Default	Extra	Default	Extra
allura	1k	8.2k	4.3k	106k
qpid-python	622	6.9k	2.5k	26k
cassandra-dtest	775	17k	2.8k	68k
infrastructure-puppet	418	4k	853	10k
libcloud	916	11k	2k	24k
incubator-superset	135	1.8k	629	8k
incubator-mxnet	1.7k	17k	2.6k	107k
bloodhound	1.1k	9.6k	1.9k	11k
incubator-airflow	895	6.2k	1.9k	23k

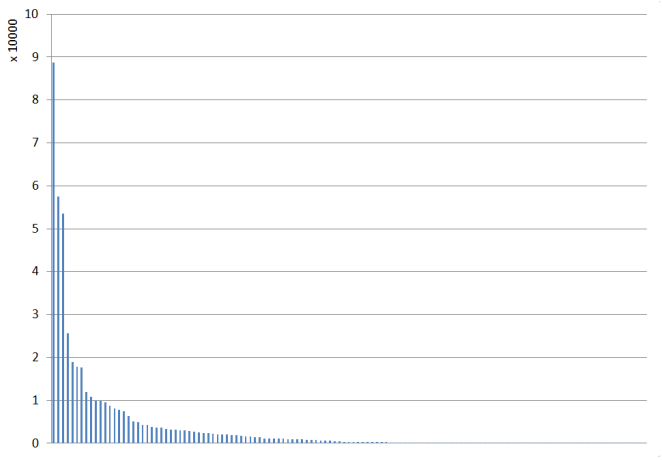

Fig. 3. Distribution of issues by issue type for the 208 types of issues

Table II presents the unresolved and fixed issues for nine Python projects, and compares the number of issues between default rules and extended rules for each project.

B. What is the fixing prevalence of different issue types?(RQ_2)

To quantify how much TD exists and investigate how prevalent the TD is in both all projects and each project, we use the 208 extended rules of SonarQube. A subset of 126 out of 208 rules are found in the nine selected Python projects.

To answer this RQ, we sum up all the issues, without differentiating among projects. The histogram of Figure 3 presents the distribution of the number of issues for each type: each vertical bar is an issue type, and its height is proportional to the number of issues of that type. The figure shows a strongly skewed distribution where some of issue types are introduced frequently but more than half of the issue types are introduced rarely.

The strongly skewed distribution of fixed issues types for is the same for both Python and Java.

C. Conclusions and Future Work

We have seen that SonarQube has different issue types for Java projects and Python projects. Out of these 27 rules appear both in Python and Java. Even if the majority of the issues are different, the distribution of issue types that are being fixed is similar: a strongly skewed distribution with several issue types being introduced frequently and more than half of the issue types being introduced very rarely.

In the future work, we plan to explore in more detail the differences and similarities between Python and Java from the point of view of technical debt evolution. Especially we plan to investigate the types of issues that have been fixed during the evolution of Python projects and the amount of TD that is paid back.

References

[1] W. Cunningham, "The wycash portfolio management system," in *Addendum to the Proceedings on Object-oriented Programming Systems, Languages, and Applications (Addendum)*, ser. OOPSLA '92. New York, NY, USA: ACM, 1992, pp. 29–30. [Online]. Available: http://doi.acm.org/10.1145/157709.157715

[2] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *IEEE Software*, vol. 29, no. 6, pp. 18–21, Nov 2012.

[3] F. Palomba, G. Bavota, M. D. Penta, R. Oliveto, D. Poshyvanyk, and A. D. Lucia, "Mining version histories for detecting code smells," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 462–489, May 2015.

[4] N. Moha, Y. Gueheneuc, L. Duchien, and A. L. Meur, "Decor: A method for the specification and detection of code and design smells," *IEEE Transactions on Software Engineering*, vol. 36, no. 1, pp. 20–36, Jan 2010.

[5] A. Potdar and E. Shihab, "An exploratory study on self-admitted technical debt," in *2014 IEEE International Conference on Software Maintenance and Evolution*, Sept 2014, pp. 91–100.

[6] G. Bavota and B. Russo, "A large-scale empirical study on self-admitted technical debt," in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, May 2016, pp. 315–326.

[7] E. d. S. Maldonado, E. Shihab, and N. Tsantalis, "Using natural language processing to automatically detect self-admitted technical debt," *IEEE Transactions on Software Engineering*, vol. 43, no. 11, pp. 1044–1062, Nov 2017.

[8] M.Yan, X. Xia, E. Shihab, D. Lo, J. Yin, and X. Yang, "Automating change-level self-admitted technical debt determination," *IEEE Transactions on Software Engineering*, pp. 1–1, 2018.

[9] S. Olbrich, D. S. Cruzes, V. Basili, and N. Zazworka, "The evolution and impact of code smells: A case study of two open source systems," in *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, Oct 2009, pp. 390–400.

[10] M. Lungu, "Reverse engineering software ecosystems," Ph.D. dissertation, University of Lugano, Nov. 2009. [Online]. Available: http://scg.unibe.ch/archive/papers/Lung09b.pdf

[11] G. Digkas, M. Lungu, A. Chatzigeorgiou, and P. Avgeriou, "The evolution of technical debt in the apache ecosystem," in *Software Architecture*. Springer International Publishing, 2017, pp. 51–66.

[12] G. Digkas, M. Lungu, P. Avgeriou, A. Chatzigeorgiou, and A. Ampatzoglou, "How do developers fix issues and pay back technical debt in the apache ecosystem?" in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, March 2018, pp. 153–163.