A Curated Benchmark Collection of Python Systems for Empirical Studies on Software Engineering

Conference Paper · October 2015

DOI: 10.1145/2810146.2810148

5 authors, including:

Matteo Orrù
Università degli Studi di Milano-Bicocca
31 PUBLICATIONS **96** CITATIONS

SEE PROFILE

Michele Marchesi
Università degli studi di Cagliari
298 PUBLICATIONS **6,732** CITATIONS

SEE PROFILE

Roberto Tonelli
Università degli studi di Cagliari
129 PUBLICATIONS **1,189** CITATIONS

SEE PROFILE

Giuseppe Destefanis
Brunel University London
60 PUBLICATIONS **528** CITATIONS

SEE PROFILE

A Curated Benchmark Collection of Python Systems for Empirical Studies on Software Engineering

Matteo Orrú[1], Ewan Tempero[2], Michele Marchesi[1], Roberto Tonelli[1], and Giuseppe Destefanis[3]
[1]DIEE, University of Cagliari
{matteo.orru,michele,roberto.tonelli}@diee.unica.it
[2]The University of Auckland, New Zealand
e.tempero@auckland.ac.nz
[3]CRIM, Computer Research Institute of Montreal, Canada
giuseppe.destefanis@crim.ca

ABSTRACT

The aim of this paper is to present a dataset of metrics associated to the first release of a curated collection of Python software systems. We describe the dataset along with the adopted criteria and the issues we faced while building such corpus. This dataset can enhance the reliability of empirical studies, enabling their reproducibility, reducing their cost, and it can foster further research on Python software.

CCS Concepts

•Software and its engineering → Object oriented languages; Software libraries and repositories; Empirical software validation;

Keywords

Python, Empirical Studies, Curated Code Collection

1. INTRODUCTION

Empirical research in software engineering often lacks of reproducibility and presents ambiguity of results due mainly to inaccuracies and uncertainties on the empirical data set used for the analysis. As a consequence, the usefulness of empirical research conducted on such data, both for scholars and practitioners, is largely reduced because it is difficult to compare the findings based on different datasets. The research efforts can be significantly improved by using a representative collection of software systems taken from real world as a benchmark, in order to enable reproducibility and comparison of the results. Presently there exists a curated collection of software, the Qualitas Corpus [18], but it is limited to Java software systems. With regards to Python systems, there is nothing to support the research this way, albeit Python is a programming language of wide adoption both in academia and industry [4, 12, 14]. Python has a wide community and it is becoming more and more popular. TIOBE index[1] and Popularity of Language Index (PYPL) [2] provide a measure of the popularity of this programming language. StackOverflow[3] and GitHub[4] witness how Python is widely used among developers. Scopus bibliographic database reports an increasing number of scientific works that deals with some aspects of Python language or present software systems developed in Python. OpenHub[5], a popular public directory of Free and Open Source Software, presently tracks 68.000 Python projects, with a positive trend along the time, confirming the increasing interest towards this language from the open source community.

The aim of software engineering is to provide the technologies which apply an engineering approach to the development and support of software products and processes. Software engineering activities includes managing, planning, modeling, analyzing, designing, implementing, testing and maintaining. To achieve this goal, measurement plays a key role in understanding, controlling and improving software development processes and products. Measurement is fundamental during each step of software development, and having the possibility to perform analysis on a well-defined corpus of systems is definitely an added value.

In this work we present a dataset of metrics taken from a curated collection on Python systems. After having downloaded 51 popular software systems, we computed several metrics. Additionally we investigated the internal structure of each system and collected meta-data in order to provide additional information to allow reproducibility of the results. Our final goal is to create a curated corpus of Python software similar to the Java Qualitas Corpus (JQC) [18]. In the meantime we are releasing the dataset of metrics associated to the software collected so far and we are confident that this dataset might be useful for researchers interested in conducting empirical studies on Python systems. This dataset and its documentation is available at the following url:
http://agile.diee.unica.it/PROMISE2015/
as are details for acquiring the corpus. This paper is structured as follows. In Section 2 we report the related works.

[1]http://www.tiobe.com
[2]http://pypl.github.io/PYPL.html
[3]http://stackoverflow.com
[4]https://github.com
[5]https://www.openhub.net

ACM ISBN .
DOI:

Then, in Section 3, we illustrate the process of construction of the dataset along with some issues we faced along the way. Next, in Section 4 we thoroughly describe our dataset. We then discuss some limitations of our dataset in Section 5 and then, in Section 6, we outline some of the research opportunities provided by our dataset. Finally we draw some conclusions in Section 7.

2. RELATED WORK

Reproducibility, reliability and applicability of results or findings can be significantly improved by the use of datasets of software systems. Several empirical studies have already been conducted on code written in different languages. For example, Succi et al. [17] conducted a study on two large datasets of 100 Java and 100 C++ software systems and Collberg at al. worked on 1132 Java systems [2]. Several studies have also been conducted on Python code [5, 16]. However, the datasets used for these studies are often not publicly available, and for this reason it is difficult to reproduce the results.

In general, there are actually many available datasets: the Mining Software Repositories conference (MSR), since 2013 [22], hosts a data-showcase session where researchers are called to illustrate and share their datasets. Additionally, many datasets of the Mining Challenge hosted in the MSR conference are available, for example in the Promise website [15]. The same Promise conference website [6] hosts a number of software engineering research datasets. Moreover, there are also some infrastructures for empirical studies on software engineering. The DaCapo benchmark [1] and the New Zealand Digital Library project [21] collect open source Java software. The Software-artifact Infrastructure Repository (SIR) [7] contains software systems written in different languages (Java, C, C++, and C#). To the best of authors' knowledge, presently the only *curated* corpus of software aimed at enabling reproducible empirical studies on software engineering is the Java Qualitas Corpus (JQC) [18]. Initially inspired by SIR, the JQC is now a mature project that provides not only code on request, but also information, in form of meta-data, on the curated software systems. More recently there is also the Qualitas.class project [19], whose main goal is to assist researchers in the effort of analyzing compiled Java code taken from the JQC.

With the exception of JQC and the related Qualitas.class, the main issue of these datasets is that they are not *curated corpora*, meaning that, in general, there is little in the way of organization beyond a collection of things, and there is little information about the contents. Moreover, with some exceptions such as the GHTorrent dataset [11] and the work of Farah et al. [8], as far as the authors know, they rarely contain Python systems. However, none of them is a curated collection or report metrics for the collected systems. This paper presents a work aimed at extending the JQC by including a curated collection of Python software systems.

3. DATASET CONSTRUCTION

The main goal of this work is to provide a dataset of metrics computed on a curated corpus of software written in Python, in order to enable researchers to perform empirical studies on Python systems. To collect the systems we followed the guidelines adopted for the JQC [18] relying also on our knowledge of the Python community. In general, we considered systems written in Python that are publicly available in order to allow researchers the access to their code. Moreover, we considered systems whose repositories are hosted on GitHub, to exploit some GitHub features for further studies. For each system of the corpus, we downloaded the latest available release from its official repository. In order to have a representative base of code, we considered systems with more than 50 Python files. Moreover, these files had to represent more than 50% of the system files. We faced two main issues when retrieving data for our corpus. The first one is related to the fact that the source code retrieved from repositories usually contains "infrastructure code", i.e., code that is related to the development, management, installation and test. Some of this code is sometimes included in the official release (e.g, examples, demo, etc.) whereas some other usually is not (e.g. test code, etc). This code is not a part of the system, but it is provided to help the user to get the most out of it. Consequently, taking into account this code while computing the metrics could bias the results. For this reason, we decided not to consider test code, examples, code associated to documentation and the like. The second issue is related to the fact that some systems were not written only in Python, but presented a significant part of the code written in another language. In order to deal with this not-Python code we made a distinction between not executable (XML, html, css, etc.) and executable code (e.g. C, javascript, etc.).

- When we found that a project had a significant part of its code written that is not executable, unless this amounts to the majority of the available code, we included it in the corpus.
- In case we found that a significant part of the executable code was written in languages different from Python, we discarded the system.

There is a third case, where we found that a significant part of the executable code is written in a language other than Python, but that code is somehow mandatory for the domain of the application. It is the case of scientific libraries like `matplotlib` that delegate some computations to most efficient C language functions. In this case we included the system in our corpus, unless the not-Python code was the vast majority of the available code.

4. DATASET DESCRIPTION

The presented dataset contains metrics associated to a corpus of 51 Python systems, belonging to different application domains as reported in Table 4. Along with several metrics, for each system we report meta-data to provide information about the systems. The meta-data are the following:

System: a unique identifier for the system.

Description: a description of the system.

Sysver: a unique identifier for the system version

Fullname: the full name of the system.

Domain: the application domain for the system.

Python Software Library Version (PSLv): which python standard library the system is compliant.

[6] http://openscience.us/repo/index.html

Domain	No
Additional Development Package	5
Applications	21
Graphic Framework	2
Math Library	4
Scientific Package	9
UI Framework	2
Web Application	1
Web Framework	7

Table 1: Application domain of collected systems

License: license under which the system was released.

LOC: number of line of code for the entire system.

NCLOC: number of line of code (excluding comments).

N_Files: number of files.

N_Classes: number of classes.

Url: url of the official site.

Repo_url: url of the official repository.

Download_date: date of the download from repository.

All this data are reported in a comma separated values (CSV) file. We considered three different kind of metrics:

- Size/Volume metrics;
- Complexity metrics;
- Object-Oriented metrics.

Depending on which kind of metrics is considered, it is possible to obtain information about the dimension of a software system, its complexity (e.g. McCabe Cyclomatic Complexity) and about object oriented properties (e.g. Chidamber and Kemerer metrics). For, example, we computed 13 size metrics - including Line of Code (LOC) and Comment Line of Code (CLOC). To calculate these metrics we used Understand 3.1 (build 766) [20]. For each system, we provided metrics for 3 different level of granularity: system, file and class level. Metrics are contained in files named following the pattern:
`<system_name>-<entity_name>-<metrics_kind>_metrics.csv`.
`<system_name>` is self explanatory whereas `<entity_name>` corresponds to the level of granularity analyzed and could be *File* or *Class*. On the other hand, `<metrics_kind>` corresponds to a category of metrics (volume, complexity and object oriented). For example, the object oriented metrics file for Zope is `Zope_file_oo_metrics.csv`. Global metrics at class and file level are reported in two files called respectively `class.metrics.global.selected.csv` and `file.metrics.global.selected`. System meta data are reported all at once in a file named `system.meta.data.csv`.

5. LIMITATIONS

In this paper we presented a dataset of metrics associated to the first release of a curated collection of Python systems that could be used to perform empirical research. Being the first release of an ongoing work (that is meant to be maintained along the years in the future as it happens for the Java Qualitas Corpus [18]), there are several limitations. The most important is that at the moment the dataset contains only the last version of the collected systems. A second limitation is that the systems have been retrieved from repositories, and we have not yet performed a thorough analysis of the internal dependencies with third party libraries hosted in repositories or released along with the source code. Thus, there is the possibility that the code base might differ from that available from other sources (i.e. Python Package Index [9]). At the moment, there are some application domains that are under-represented (e.g. Web Application): as future work we plan to put effort in order to increase the number of systems in these categories and, in general, enhance the corpus representativeness. Despite the mentioned limitations, we are confident that the provided data can be an interesting source of information for researchers interested in investigating Python systems properties and development.

Astropy Biopython BuildBot Calibre CherryPy Cinder Django Emesene EventGhost Exaile GlobaLeaks Gramps "Getting Things Gnome!" OpenStack - heat IPython Kivy OpenStack - magnum "GNU Mailman" OpenStack - manila Matplotlib Miro NetworkX OpenStack - neutron Nova NumPy Pathomx "Python Imaging Library" Pip Plotly Portage Pygame PyObjC Pyramid "PYthon Remote Objects" "Quod Libet" SABnzbd Sage scikit-image scikit-learn SymPy TurboGears2 Tornado Trac Tryton Twisted Veusz VisTrails VPython web2py wxPython Zope

Figure 1: List of the analyzed system

6. RESEARCH OPPORTUNITIES

Even if Python is a widespread programming language both in academia and in industry, there are few empirical studies on Python software compared to other languages, i.e., Java. For this reason there are many research opportunities for scholars, starting from replication studies that can be performed on Python systems, in order to assess if the already found results apply also to Python. The kind of studies that might be supported by this dataset are mainly those that involve a static analysis of Python code. In this perspective there are many specific features of Python language that are not available for Java software (e.g. dynamic binding, multiple inheritance, etc.) that could be investigated to understand how they are used by developers. A comparison of the usage of some Python features in a specific application domain with their counterpart on Java could lead to interesting results.

The provided metrics can be compared with other measurements of different nature [3] taken from the associated repositories (e.g., social metrics) or the corresponding Issue Tracking Systems (e.g. number of issues, etc.), or for patterns detection [6, 10]. Since we provided a categorization for the software systems in the corpus, software metrics can be also used to investigate specific differences among systems, depending on the application domain. Additionally, it could be interesting to investigate if and how metrics change

among different components of the same system. It could also be interesting to study different properties of systems that share the same application domain but are written in different languages.

7. CONCLUSIONS

In this work we presented a dataset of metrics associated to a curated collection of Python systems, describing the motivation that led us to build the corpus, the main issues involved in creating it, and providing a description of the dataset and its limitations. We also reported some suggestions about the use of this dataset for empirical studies of Python systems. The main goal of this corpus is to provide a benchmark for empirical studies of Python systems that allows reproducibility of results and lowers the cost of experiments. This corpus is intended to be a constant work in progress [13]. In the future we plan not only to to increase the number of systems available in the corpus, but also to include different versions of the same system to promote longitudinal studies. We are also considering to add information about the development process taken from the official repositories, along with information from issue tracking systems. This dataset is available for download at http://agile.diee.unica.it/PROMISE2015/

8. REFERENCES

[1] Stephen M. Blackburn, Robin Garner, Chris Hoffmann, Asjad M. Khang, Kathryn S. McKinley, Rotem Bentzur, Amer Diwan, Daniel Feinberg, Daniel Frampton, Samuel Z. Guyer, Martin Hirzel, Antony Hosking, Maria Jump, Han Lee, J. Eliot B. Moss, Aashish Phansalkar, Darko Stefanović, Thomas VanDrunen, Daniel von Dincklage, and Ben Wiedermann. The DaCapo Benchmarks: Java Benchmarking Development and Analysis. In *Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications*, OOPSLA '06, pages 169–190, New York, NY, USA, 2006. ACM.

[2] Christian Collberg, Ginger Myles, and Michael Stepp. An empirical study of Java bytecode programs. *Softw. Pract. Exper.*, 37(6):581–641, May 2007.

[3] Giulio Concas, Cristina Monni, Matteo Orrù, and Roberto Tonelli. A study of the community structure of a complex software network. In *4th International Workshop on Emerging Trends in Software Metrics, WETSoM 2013, San Francisco, CA, USA, May 21, 2013*, pages 14–20, 2013.

[4] Giuseppe Destefanis. Technical report: Which programming language should a company use? a twitter-based analysis. *CRIM - Technical Report*, 2014.

[5] Giuseppe Destefanis, Steve Counsell, Giulio Concas, and Roberto Tonelli. Software metrics in agile software: An empirical study. In *Agile Processes in Software Engineering and Extreme Programming*, pages 157–170. Springer, 2014.

[6] Giuseppe Destefanis, Roberto Tonelli, Ewan Tempero, Giulio Concas, and Michele Marchesi. Micro pattern fault-proneness. In *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on*, pages 302–306. IEEE, 2012.

[7] Hyunsook Do, Sebastian Elbaum, and Gregg Rothermel. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Softw. Engg.*, 10(4):405–435, October 2005.

[8] Gabriel Farah, Juan Sebastian Tejada, and Dario Correal. Openhub: A scalable architecture for the analysis of software quality attributes. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 420–423, New York, NY, USA, 2014. ACM.

[9] Python Software Foundation. Python package index, 1999.

[10] Joseph Yossi Gil and Itay Maman. Micro patterns in Java code. In *ACM SIGPLAN Notices*, volume 40, pages 97–116. ACM, 2005.

[11] Georgios Gousios. The GHTorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA, 2013. IEEE Press.

[12] Philip Guo. Python is now the most popular introductory teaching language at top us universities. *BLOG@ CACM, July*, 2014.

[13] Susan Hunston. *Corpora in applied linguistics.* Cambridge University Press, 2006.

[14] Johnny Wei-Bing Lin. Why Python is the next wave in earth sciences computing. *Bulletin of the American Meteorological Society*, 93(12):1823–1824, 2012.

[15] Tim Menzies, Bora Caglayan, Ekrem Kocaguneli, Joe Krall, Fayola Peters, and Burak Turhan. The PROMISE Repository of empirical software engineering data, June 2012.

[16] Sebastian Nanz and Carlo A. Furia. A Comparative Study of Programming Languages in Rosetta Code. *CoRR*, abs/1409.0252, 2014.

[17] Giancarlo Succi, Witold Pedrycz, Snezana Djokic, Paolo Zuliani, and Barbara Russo. An empirical exploration of the distributions of the Chidamber and Kemerer Object-Oriented Metrics Suite. *Empirical Softw. Engg.*, 10(1):81–104, January 2005.

[18] Ewan Tempero, Craig Anslow, Jens Dietrich, Ted Han, Jing Li, Markus Lumpe, Hayden Melton, and James Noble. Qualitas corpus: A curated collection of Java code for empirical studies. In *2010 Asia Pacific Software Engineering Conference (APSEC2010)*, pages 336–345, December 2010.

[19] Ricardo Terra, Luis Fernando Miranda, Marco Tulio Valente, and Roberto S. Bigonha. Qualitas.class Corpus: A compiled version of the Qualitas Corpus. *Software Engineering Notes*, 38(5):1–4, 2013.

[20] Understand. Scitools.com: https://scitools.com.

[21] Ian H. Witten, Sally Jo Cunningham, and Mark D. Apperley. The New Zealand digital library project. *New Zealand Libraries*, 48:146–152, 1996.

[22] Thomas Zimmermann, Massimiliano Di Penta, and Sunghun Kim, editors. *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, San Francisco, CA, USA, May 18-19, 2013.* IEEE Computer Society, 2013.