Enabling Empirical Research: A Corpus of Large-Scale Python Systems

Safwan Omari and Gina Martinez
Department of Computer and Mathematical Sciences
Lewis University
Romeoville, IL, USA
omarisa@lewisu.edu, martingi@lewisu.edu

***Abstract*—The Python programming language has been picking up traction in Industry for the past few years in virtually all application domains. Python is known for its high calibre and passionate community of developers. Empirical research on Python systems has potential to promote a healthy environment, where claims and beliefs held by the community are supported by data. To facilitate such research, a corpus of 132 open source python projects have been identified, basic information, quality as well as complexity metrics has been collected and organized into CSV files. Collectively, the list consists of $36,635$ python modules, $59,532$ classes, $253,954$ methods and $84,892$ functions. Projects in the selected list span various application domains including Web/APIs, Scientific Computing, Security and more.**

***Index Terms*—Python, Corpus, Empirical Research, Quality Metrics, Complexity Metrics**

I. Introduction

The Python programming language has been picking up popularity over the past few years according to several indices that track and measure programming languages popularity [7], [13]. Python provides a rich set of evolving primitives and open-source frameworks that support various application domains such as Web/API, Data science, security and much more. The rise of Python has caught the attention of the research community and triggered a slew of research efforts addressing various aspects of the language and its evolution [3].

Among other research efforts, empirical research in software engineering has great potential in bridging the gap between Academia and Industry. Rooted in extensive measurements and use of sound statistical models, empirical research has the potential in supporting or refuting many of the unsupported claims often made and beliefs held sacred by various communities of developers. This promotes a more data-driven approach to best practices and well-informed decision making in Industry, which invites more of constructive discussions and less of ideological debates.

Statically typed languages such as Java and C have enjoyed the bulk of empirical research in software engineering [6], [10], [11], [14], we believe that Python exhibits unique internal characteristics [5]. This warrants a fresh wave of empirical research to address different aspects of the language and process related practices in the community [1], [2], [6], [8], [11]. Robustness of Statistical conclusions and quality of models presented in empirical research is largely impacted by use of consistent and representative software artifacts [12], [15]. In this paper, we build upon previous efforts [12], [15] and present a large-scale Python project data-set to further and facilitate empirical research in software engineering. Selection criteria is carefully defined to yield popular projects with long development history and many contributors.

Corpus consists of 132 Python projects, necessary meta-data including repository `URLs`, description, release tags, and several others are collected and organized in `csv` files. Furthermore, several quality and complexity metrics have been collected and included. Collectively, corpus consists of $36,635$ modules and more than 5.7 million Python lines of code. Average system is 4 years old with at least 26 releases. 50% of corpus systems exhibit a Pylint quality score of 8 out of 10 or higher, and 50% of projects showed an average McCabe's cyclomatic complexity score of 7.

The rest of the paper is organized as follows, Section II presents related work; corpus design goals, selection criteria and contents are discussed in Section III; Section IV presents several quality as well as complexity metrics and provides summary statistics; and finally paper is concluded and future work is discussed in Section V.

II. Related Work

Python's rise as one of the mainstream languages in Industry is attracting increasing interest from the research community, which studies all aspects of the language features and evolution [1], [2], [4], [6], [8], [9], [11], [13]. While more traditional languages such as Java and C/C++ has enjoyed the bulk of research community attention [6], [10], [11], empirical research on Python is still in its infancy. Similar to previous research efforts in [12], a corpus of Python systems is proposed to help advance and facilitate future empirical research. Proposed corpus builds on similar corpus proposed in [12], by constructing a larger set of Python systems, which are carefully selected to include large-scale systems. Furthermore, an extensive set of meta-data and metrics are collected and organized in `csv` files for easy access and replication. Finally, a thorough description of various system characteristics is presented, the goal is to provide as much context as possible around statistical results drawn based off the corpus.

```
ajenti ansible astropy autobahn-python aws-cli beets biopython blaze bokeh boto3 buildbot bup
calibre celery certbot ckan cobbler curator compose conda CouchPotatoServer cryptography cython
django django-allauth django-blog-zinnia django-cms django-extensions django-haystack django-
oscar django-rest-framework django-shop django-tastypie docker-py edx-platform elastalert
electrum erpnext eve fail2ban faker falcon flask fonttools gensim gevent glances google-cloud-
python headphones home-assistant hue ipython jinja jupyterhub kafka-python kivy livestreamer
luigi matplotlib mongoengine mezzanine mitmproxy mongo-python-driver mopidy mrjob munki mypy
networkx paramiko nova numba nupic OctoPrint orange3 pandas peewee pelican picard pika Pillow
pip platform_development play1 psutil pwntools pyethereum pyinstaller pymc3 pyramid pyzmq
qutebrowser ranger raven-python readthedocs.org redash you-get robotframework s3cmd saleor salt
scikit-image scikit-learn scipy scrapy securedrop security_monkey sentry Sick-Beard SiCKRAGE
spaCy spinnaker sqlalchemy sqlmap st2 supervisor swift sympy synapse Theano thumbor tornado
tribler troposphere twisted w3af wagtail WeasyPrint web2py aiohttp werkzeug youtube-dl zipline
```

Fig. 1. Python project names: total of 132 projects.

III. PYTHON CORPUS

A. Design Goals

Software engineering empirical research targets a wide range of statistical hypothesis. The type of hypothesis under study plays a primary role in data-set design criteria. For example, testing hypothesis about early stages of software development life-cycle, requires a data-set of artifacts in their early states of evolution, whereas, doing same for latter stages of software life-cycles, should be performed on a corpus of projects with several years of development history and multiple release cycles. We seek to facilitate and support empirical research on large-scale and mature Python Software Projects. Therefore, selection criteria in this work are crafted to include popular projects with long development history and multiple release cycles. To ensure easy access to the exact source code, all necessary project meta-data and information are collected and included in corpus. Furthermore, several complexity as well as quality metrics were generated and included. Having good understanding of corpus characteristics is essential to verify necessary assumptions and appropriateness of corpus for target study.

To meet stated goals, the authors chose to include open-source projects hosted in Github and collected needed meta-data to enable future researchers to obtain and reconstruct exact copy of software artifacts (i.e., `URL` and release tags). For quality and complexity metrics, Pylint and radon are used. The latter generates raw and object-oriented complexity metrics as well as McCabe's complexity, whereas, the former generates metrics about code style convention violations as well as code smells.

B. Selection criteria

The total number of Python projects in Github is upward of 608K projects. In this work, a two-phase approach is used to construct a corpus of projects that satisfy stated goals. In *Phase-I*, Github REST searching API is used to filter projects based on primary programming language, date of creation, total stars received and date of last commit to the project repository. Phase-I filtering is meant to define bare-minimum thresholds in terms of age and popularity. Parameters are set as follows: created before $= 2015-01-01$, stars $>= 600$ and date of last commit $>= 2018-07-01$. This resulted in a total of 928 Python projects. A manual investigation of the initial list revealed several repositories that were not in fact Python systems, these systems were eliminated. The following are some examples:

- Public lists (json): list of Members of the US Congress since 1789.
- Personal web-sites.
- Lists of frameworks and libraries: awesone-python, awesone-machine-learning.
- Documentation: PEPs, Raspberry Pi, Python packaging user guide, Ansible playbooks for Wordpress, scipy-lecture-notes, etc.
- Configuration lists (json).

To elaborate on the goal of selecting the most mature projects with a rich development history, in *Phase-II*, the following criteria are developed and projects that fall in top 50^{th} percentile are selected:

1) *Number of stars* received and *watchers*: project must be popular and of interest to a large population. 50^{th} percentile thresholds for stars received and number of watchers are found to be 1457 and 99, respectively.
2) *Number of commits*: project must have a long development and evolution history. 50^{th} percentile threshold is found to be 1067 commits.
3) *Number of contributors*: project must be developed by a large community of developers. This is essential for corpus population to capture and exhibit typical team dynamics characteristics. 50^{th} percentile threshold is found to be 22 contributors.
4) *Number of releases*: project must be mature and has gone through multiple release cycles and actively in use. 50^{th} percentile is found to be 26 releases.

C. Corpus contents

The total number of projects that satisfied initial Phase-I criteria and ranked in top 50th percentile came out to be 132 Python projects. For each project, the following meta-data is

Fig. 2. Word cloud based on topic keywords

TABLE I
SUMMARY CORPUS META-DATA

Project Age (years, months)			
Min	Max	Median	Mean
4y, 2m (elastalert)	10y, 3m (platform_development)	7y, 3m	7y, 5m
Number of commits			
Min	Max	Median	Mean
1261 (django-tastypie)	101,678 (salt)	10,255	6,784
Number of stars received on Github			
Min	Max	Median	Mean
1,463 (fonttools)	45,187 (youtube-dl)	6,863	4,016
Number of watchers on Github			
Min	Max	Median	Mean
103 (s3cmd)	2,238 (flask)	394	252
Number of contributors on Github			
Min	Max	Median	Mean
49 (WeasyPrint)	443 (matplotlib)	207	192
Number of releases			
Min	Max	Median	Mean
26 (django-tastypie, flask, zipline)	1114 (edx-platform)	132	74

collected: *name, description, topics, age, repository URL, number of commits, number of watchers, stars received, number of releases, release tag. Using repository URL and specifying release tag*, future researchers should be able to an clone exact copy of source code for project in corpus (`git clone -b release-tag URL`). Fig. 1 shows the final list of repositories. Table I shows basic summary statistics of corpus projects.

As depicted in the word cloud in Fig. 2, projects span a wide array of application domains. Word cloud is based on topic keywords stated for each one of the projects. It is found that 44 projects were lacking any topic keywords on Github, for those, topic keywords were compensated with keywords from project description. Keywords were obtained by applying stemming and removing of stop words from full description text. Word cloud is generated using Pro Word Cloud add-ins plugin in MS Word. Web, data science as well as security applications and frameworks are dominant themes. Top frequency words are: django: 14 times, web: 9 times, frameworks: 7, library: 7 times, data science: 5 times, http: 5 times

IV. Internal Characteristics

In addition to basic project information, several internal source code metrics have been collected. Metrics are grouped into two categories: quality and complexity metrics. Such metrics shed light on internal code quality of data-set and help future researchers assess the appropriateness of corpus for the problem at hand. Python module (file) is used as a unit for collecting quality and complexity metric. Modules that are not part of the core system are excluded from quality and complexity calculations, including unit tests, scripts and example code, if any. We believe that code in these modules exhibit different quality and complexity characteristics from the core system. Module-level metrics are aggregated in a corresponding project-level metric. With the exception of McCabe's cyclomatic complexity and Pylint score, project-level metric is simply calculated as the sum of all corresponding module-level metrics. For example, the total number of convention violations at the project-level equals to the sum of module-level convention violations for all modules in the project. Details on McCabe's complexity and Pylint score are presented later in the section.

A. Quality metrics

TABLE II
SUMMARY CORPUS QUALITY METRICS

Number of refactor violations			
Min	Max	Median	Mean
3 (munki)	10,149 (ansible)	386	872
Number of style violations			
Min	Max	Median	Mean
6 (home-assistant)	375,317 (calibre)	1,951	10,235
Pylint score			
Min	Max	Median	Mean
-10.3 (fonttools)	9.9 (home-assistant)	7.9	6.9

We use Pylint to generate several quality metrics of selected projects. Pylint is a static analyzer that identifies five classes of quality concerns: **R**efactor, **C**ode Style, **W**arnings, **E**rrors and **F**atal. In this study, we focus our attention on Refactor and Code Style as primary quality metrics. Refactors are concerned with locating code smells and measuring how often it occurs in Python code, whereas, code style reports on project's adherence to a well-accepted set of code conventions and standards. Pylint recognizes and reports on 34 unique code style violations and 24 code smells. Example convention violations include bad variable names, missing docstring and using `type()` rather than idiomatic `isinstance()` for type checking. Example refactor violations include too many ancestors and too few public methods.

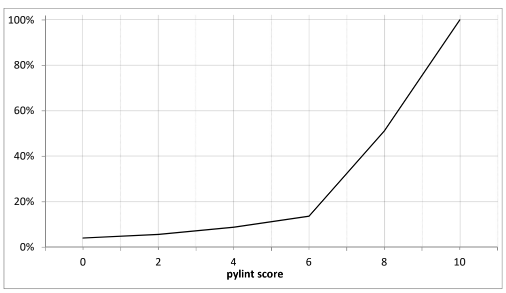
Besides raw number of **R**efactor and **C**onvention violations in a Python module, Pylint calculates module-level score as $10 - (C + R)/S * 10$, where C represents number of convention violations, R represents number of refactors and S represents number of logical Python lines of code in the module. Project-level score applies same equation using total C, R and S across all modules in the project. Scripts has been developed to run and collect Pylint metrics per module and calculate project-level score. Table II provides summary quality statistics of corpus.

B. Complexity metrics

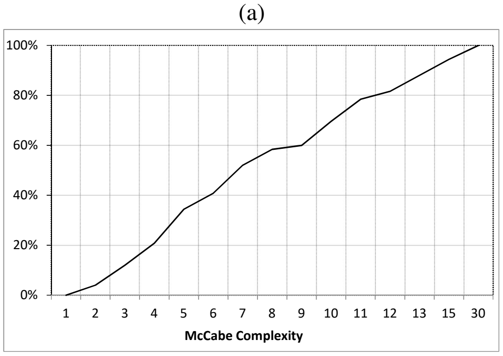
Radon is used to obtain raw and complexity metrics including source lines of codes, number of classes, number of methods, number of functions and McCabe's cyclomatic complexity per Python

TABLE III
SUMMARY CORPUS COMPLEXITY METRICS

Source lines of code			
Min	Max	Median	Mean
934 (raven-python)	723,480 (ansible)	17,465	45,400
Number of classes			
Min	Max	Median	Mean
14 (securedrop)	5,893 (hue)	241	476
Number of methods			
Min	Max	Median	Mean
37 (securedrop)	26,842 (hue)	910	2,031
Number of functions			
Min	Max	Median	Mean
11 (pika)	15,337 (salt)	268	679
McCabe's cyclomatic complexity			
Min	Max	Median	Mean
1.3 (django-allauth)	29.0 (WeasyPrint)	6.8	7.9



(a)



(b)

Fig. 3. Cumulative distribution function, (a) Pylint score, (b) McCabe's cyclomatic complexity.

module. With the exception of McCabe's complexity, project-level metric is calculated as sum of corresponding module-level one. For example, number of classes in a project is calculated as sum of classes in all modules of the project. Radon reports McCabe's complexity score per methods and functions. We define module-level McCabe's complexity as the maximum complexity among all methods and functions defined in the module. We believe the module's complexity is dictated by its most complex function or method. Furthermore, Project-level McCabe's complexity is calculated as average McCabe's score among all modules in the project. Table III reports summary statistics of complexity metrics for projects in corpus.

Using pylint score and McCabe's complexity score as proxies for project quality and complexity, respectively, Fig. 3 depicts an empirical cumulative distributive function for Pylint score in Fig. 3(a), and McCabe's cyclomatic complexity in Fig. 3(b). Almost 50% of projects in corpus have a Pylint score of 8 or below. 80% of projects have a complexity score of less than 11.

V. SUMMARY AND FUTURE WORK

A corpus of 132 Python projects are constructed and organized into `csv` files. Collectively, corpus consists of $36,635$ python modules, $59,532$ classes, $253,954$ methods and $84,892$ functions. To provide future researchers with insights into corpus internal characteristics, several quality as well as complexity metrics has been aggregated and included in the corpus. For future work, the authors are planning to pursue construction of specialized corpora to facilitate empirical research on niche Python systems populations, in addition to collection and inclusion of more metrics such as Halstead metric, Maintainability Index, etc.

REFERENCES

[1] An empirical study of dynamic types for python projects. In *8th International Conference (SATE))*, 2018.

[2] Beatrice Akerblom and Tobias Wrigstad. Measuring polymorphism in python programs. In *Proceedings of the 11th Symposium on Dynamic Languages*, DLS 2015, 2015.

[3] Carol V. Alexandru, José J. Merchante, Sebastiano Panichella, Sebastian Proksch, Harald C. Gall, and Gregorio Robles. On the usage of pythonic idioms. In *Proceedings of the 2018 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2018, 2018.

[4] Z. Chen, W. Ma, W. Lin, L. Chen, and B. Xu. Tracking down dynamic feature code changes against python software evolution. In *2016 Third International Conference on Trustworthy Systems and their Applications (TSA)*, 2016.

[5] G. Destefanis, M. Ortu, S. Porru, S. Swift, and M. Marchesi. A statistical comparison of java and python software metric properties. In *2016 IEEE/ACM 7th International Workshop on Emerging Trends in Software Metrics (WETSoM)*, 2016.

[6] Giuseppe Destefanis, Steve Counsell, Giulio Concas, and Roberto Tonelli. Agile processes in software engineering and extreme programming. chapter Software Metrics in Agile Software: An Empirical Study.

[7] Philip Guo. Python is now the most popular introductory teaching language at top u.s. universities. https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext, 2014.

[8] W. Lin, Z. Chen, W. Ma, L. Chen, L. Xu, and B. Xu. An empirical study on the characteristics of python fine-grained source code change types. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2016.

[9] B. A. Malloy and J. F. Power. Quantifying the transition from python 2 to 3: An empirical study of python applications. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2017.

[10] Nachiappan Nagappan, Thomas Ball, and Andreas Zeller. Mining metrics to predict component failures. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, 2006.

[11] S. Nanz and C. A. Furia. A comparative study of programming languages in rosetta code. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, 2015.

[12] Matteo Orrú, Ewan Tempero, Michele Marchesi, Roberto Tonelli, and Giuseppe Destefanis. A curated benchmark collection of python systems for empirical studies on software engineering. In *Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering*, PROMISE '15, 2015.

[13] The software quality company. Python is TOIBE's programming language of the year 2018!, 2019.

[14] Hoh In Taek Lee, Jung Been Lee. A study of different coding styles affecting code readability. *International Journal of Software Engineering and its Applications*, 7(5):413–422, 10 2013.

[15] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble. The qualitas corpus: A curated collection of java code for empirical studies. In *2010 Asia Pacific Software Engineering Conference*, 2010.