Structural Quality & Software Evolution

Alison Major
Computer Science
Lewis University
Romeoville, Illinois, USA
AlisonMMajor@lewisu.edu

I. INTRODUCTION

This paper will investigate how well the Pylint score (more specifically, the Refactor score) measures a system's structural quality. To do this, we will analyze the correlation between the Refactor score and the ease of adding features to the system.

Good architecture takes into account maintainability. The effort to make a feature work should be easy and localized in maintainable projects. We will measure locality by the number of files edited in a commit. We will focus on commits that represent new features rather than bug fixes.

This paper will review how structural quality can impact software evolution.

II. WORKING WITH DATA

The work done by Dr. Omari and Dr. Martinez involves collecting a sub-set of Python projects that can be used for further research. The bulk of the effort that they have provided is in determining which classifiers to use to pare down the public set of Python systems into a reasonable set for further analysis [1]. This work will be used to select appropriate Python systems for review by collecting meta-data on these code systems. With this information, we hope to understand the impact that structure quality can have on the software evolution process.

In addition to data collection, many articles contain exciting information about this very topic.

First, in order to understand the scores we'll be working with, we must understand what Pylint itself is doing. Through the documentation of Pylint, we can understand how to use it an the scores it will provide [2]. We can also review the documentation to understand how the Pylint Score is calculated, as well as the various features that the Refactor Score takes into account [3]. Finally, in respect to Python, it is also important to understand PEP 8, as this is the default set of standards that Pylint uses to judge Python code [4].

A study conducted by Baishakhi Ray, Daryl Posnett, Premkumar Devanbu, and Vladimir Filkov begins by programmatically collecting a sample set of projects in GitHub that vary in languages. Then the set is appropriately culled, resulting in a final set used for review. The results are then a study of the impact that different programming languages may have on the code quality of a project [5]. The methods that this group used and ideas they have formed may be useful in creating direction and assumptions in our own research.

III. INSIGHTS INTO MAINTAINABILITY

Another paper, "Standardized code quality benchmarking for improving software maintainability," provides additional insights into how code maintainability is impacted by the technical quality of source code [6]. Within their paper, they seek to show four key points: (1) how easy it is to determine where and how the change is made, (2) how easy it is to implement the change, (3) how easy it is to avoid unexpected effects, and (4) how easy it is to validate the changes. The research that this group provides could provide useful insights in our project.

Reviewing the "Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models" maintained by ISO may also provide interesting insights [7]. Does the Pylint checker follow the quality models outlined here? Are there benefits or drawbacks to the models that the ISO/IEC:25010:2011 suggests? These could influence more thoughts on interpreting final data results.

Within the course's text book, "Software Architecture in Practice," chapter 18 provides some insight in documentation around architecture [8]. When reviewing code quality scores, it would be interesting to review some of the documentation around the highest scoring software systems and some of the lowest scoring software systems. Do these projects have adequate documentation? Does the level of documentation correlate to the ability to maintain a proper score? I would be curious to see if contributing developers are positively influenced by good documenation, or what other factors may contribute to maintainable code structures.

Another variable that may impact the maintainability of code is readability. In the article "How does code readability change during software evolution?" the authors have addressed this concern, and found that within the sample that they reviewed, most source code was readable and a minority of commits changed the readability [9]. This variable in the maintainability of a software system can influence how easy or difficult it is to make a change. Referencing the findings from these authors, as well as the guidelines they provide for maintaining readiability, could be useful when building conclusions from the data.

REFERENCES

[1] Safwan Omari and Gina Martinez. Enabling empirical research: A corpus of large-scale python systems, 2018. [Provided by Dr. Omari].

[2] Logilab and contributors. Pylint. Logilab, 2020. https://pylint.org/ [Online; accessed 7-December-2021].

[3] PyCQA Logilab and contributors. Pylint features. Logilab and PyCQA, 2021. https://pylint.pycqa.org/en/latest/technical_reference/features.html# reports-options [Online; accessed 7-December-2021].

[4] Python Software Foundation and contributors. Pep 8 – style guide for python code. Heroku Application, 2021. https://www.python.org/dev/ peps/pep-0008/ [Online; accessed 7-December-2021].

[5] Baishakhi Ray, Daryl Posnett, Premkumar Devanbu, and Vladimir Filkov. A large-scale study of programming languages and code quality in github. *COMMUNICATIONS OF THE ACM*, 60(10):91–100, 2017.

[6] Robert Baggen, José, Pedro Correia, Katrin Schill, and Joost Visser. Standardized code quality benchmarking for improving software maintainability. *Software Qual J*, 20:287—307, 2012.

[7] Technical Committees: ISO/IEC JTC 1/SC 7, Software, and systems engineering. Iso/iec 25010:2011: Systems and software engineering — systems and software quality requirements and evaluation (square) — system and software quality models. ISO, 2011. https://www.iso.org/ standard/35733.html [Online; accessed 7-December-2021].

[8] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice: Third Edition*. Addison-Wesley Professional, 2012.

[9] Valentina Piantadosi, Fabiana Fierro, Simone Scalabrino, Alexander Serebrenik, and Rocco Oliveto. How does code readability change during software evolution? *Software Qual J*, 25:5374—-5412, 2020.