Department of Electrical, Computer, and Software Engineering

Part IV Research Project

Project Compendium Report

Project Number: 67

Evaluating Identifier Quality

Jack Chu

Jafar Maash

Associate Professor: Ewan Tempero

19/10/2022

Declaration of Originality

This report is our own unaided work and was not copied from nor written in collaboration with any person other than my project partner.

Names: Jack Chu, Jafaar Maash

ABSTRACT:

Project 67's aim was initially set out to evaluate identifier quality in source code, applying rules and guidelines in order to potentially quantify how meaningful any identifier name was. After a literature review, this evolved into evaluating how guidelines were holistically followed in source code. This project resulted in the creation of a Java/Python tool which was used to evaluate nearly 50 different source code repositories and versions against approximately 13 identifier naming guidelines, presenting statistics for each; detailing the number of fields and methods in every repository, as well as the number of identifiers which violated every rule. Future work for the project could include implementing additional guidelines which were omitted in the interest of time or efficiency, evaluating further repositories to analyse trends on a more macroscopic scale, investigating relationships between guideline adherence and other factors of interest such as code quality, and potentially using all the data to postulate an authoritative set of guidelines based on correlations between existing evaluated guidelines' adherence and code quality, readability, etc.

1. Introduction

The comprehensibility of programs is "an essential part" of a software's evolution and maintenance, and with approximately 70% of a code system being made up of identifiers, creating meaningful identifiers is a significant part of writing understandable code. There are many guidelines in the literature for how to name identifiers, and studies have shown that following them is associated with higher quality and also more comprehensible code. Past research has shown that identifiers in practice can often violate these guidelines. However, the guidelines used in this research often have included outdated rules or haven't addressed important aspects of identifiers, such as the fact that they can commonly be made up of abbreviations, not only dictionary words. No research has also evaluated the same set of identifiers by guidelines from multiple studies.

This research project evaluates how closely open-source Java projects follow identifier-naming guidelines established in the literature, combining rules from three different studies to assess identifiers against. This report discusses our project journey, including research and implementation methods. The complete compendium presents the source code for two tools which work in unison to achieve this goal: a JavaParser tool for extracting identifiers out of any Java source code repository, and a Python tool for evaluating these identifiers against semantic naming guidelines laid out in the literature. In addition to these tools, this compendium contains instructions for how to use them, links to every repository/project tested, and data pertaining to the nearly 50 different repositories/projects and versions tested, describing the proportion of identifiers in each which violated every rule.

2. Project Journey

2.1 Literature Review

This was the first hurdle in our project, as we had little to no knowledge of this field of study. We started by looking within the references of a paper given to us by our supervisor, slowly expanding our collection of papers to be used within our literature review. We found many papers regarding identifiers and identifier quality, and these studies were done mainly using either technical or experimental empirical analysis. The majority of our papers were about evaluating the identifiers metrics or identifier naming through following guidelines. The gap in research we found was whether or not developers were actually following said guidelines. This eventually led to our research intent being “how closely do developers follow naming guidelines.”

2.2 Research Methods

After our literature review, we started our research by looking into easier-to-evaluate guidelines to implement. This included violations related to length, where identifiers should not be over 20 characters long, and other various underscore violations. This was a good way for us to explore the functionality provided by JavaParser, which allowed us to gather all identifiers from a Java repository. From here, we started checking for casing violations for camel case usages, where all identifiers that did not follow camel casing would be counted as a violation. However, we quickly realised that this meant single-word identifiers would also count as a violation, and therefore we had to change our code.

We initially tested on smaller repositories, some given by our supervisor, and some were taken from our own coding assignments. This helped us easily identify any bugs or issues in our code, since we could also manually perform the checks ourselves on the smaller

repositories. Once we were confident in our code, we moved on to finding open-source Java projects to evaluate. Aside from ApacheAnt, which was taken from the Qualitas Corpus, all the repositories we used can be found on the internet and they are all included in our project ReadMe file.

We started looking into libraries that we could use to split our identifiers, which was very useful throughout the project. We found a python library called ‘spiral,’ by casics, which could help split our identifiers into separate words. This library was chosen because it included splitting of programming-related vocabulary, such as “utf8” or “getInt,” which is something many other libraries lacked. From our evaluation, this library was very accurate with its splitting, and we ran into very few issues after it was set up.

Aside from spiral, we also used another python library called spaCy, which has a Natural Language Processing (NLP) module called ‘ronin.’ It uses tokenisation for Part-Of-Speech (POS) tagging, which allows us to implement checks for POS related naming guidelines. The guidelines we chose were mostly the easiest to implement within our given time frame, and they came from the Java Language Specification, and studies done by Binkley et al. and P. A. Relf.

2.3 Implementation

We started developing our tool using an old repository given to us by our supervisor, which included almost all usages of different JavaParser functionalities. We selected the ones that were relevant, and added it to our code as a base. We then refactored the code slightly, making it easier to switch between repositories that we wanted to test. From there, we could easily extract all the identifiers from a given repository, and we used a simple character count

and RegEx for length and underscore violations respectively. Moving on to case violations, we use RegEx again to compare each identifier to, however as mentioned previously, single-word identifiers would be counted as a violation so we had to implement a script to help split our identifiers.

The spiral library was relatively simple to implement, and we could easily count true camelcase violations by only counting the identifiers that would be split into more than one word. This way, single-word identifiers are exempt from being counted incorrectly as a violation, and we had an accurate display of camel case violations. Later on in the project, after we had expanded our evaluation of different case types to four different ones (camel, pascal, kebab, and snake, all evaluated using RegEx), we found that all the programs we tested our tool on used camel case as their primary case type, however there would often exist other case types which counted as an inconsistency, but we did not add this to the number of case violations. The evaluated total for case violations include all identifiers that did not belong to one of our selected case types and could be split into more than one word.

The ronin module from the spaCy library was implemented in a separate python script file, where we also used the splitter code from our case violation evaluation, but we followed different POS guidelines to implement using the tags that were given to identifiers by ronin. We added each of these guideline violations to their own separate count, and used prints to detect violations or any bugs that may have occurred. In the beginning, we also used prints to see our outputs, which we would then manually add to our spreadsheet of results. However, once the number of violations we were evaluating increased, this became rather tedious so we

automated this process slightly by writing our results to a csv file, which could then be easily copied into our results spreadsheet.

This covers our entire project timeline and workflow, please see our ReadME file for instructions on how to run the repository. Any resources or libraries mentioned above can be found in the ‘Links.docx’ file, also found in this compendium.

2.4 Conclusion

Our literature review captured not only what had been established in this field of study, but also what has not. This led us to investigate the closeness to which developers follow identifier naming guidelines. Our investigation utilises JavaParser, spiral from casics, and ronin from spaCy to gather identifiers, split them, and then tokenise them for POS evaluation. We then write our results to a csv file, which is finally added to our data spreadsheet, which can be found inside this compendium.

3. Challenges

Our biggest challenge was to try and mitigate the effects of spaCy’s poorn NLP algorithm, which would often tag words incorrectly due to capitalisations from camel casing, or even simply giving a word the wrong tag because it could not correctly identify the context in which the word was being used. This led to us making several band-aid fixes during the project, but we did specify in our final report that this will be an issue moving forward.

Other issues include setting up our Python environments correctly, as there was an issue with spiral where we had to edit the source files of the library itself to make it run. Other issues would appear during the project, such as literal ‘bad’ files made to create errors in some

repositories, but we managed to fix all the major problems, e.g. by throwing exceptions when needed.

4. Reflection

Overall, our project was a very good learning experience, and we have our supervisor, Ewan Tempero, to thank for it. We learned a great deal about researching into a mostly unexplored topic, and how to evaluate metrics to gather data and find trends. Initially, our workflow was staggered, and it would take some time to get through certain hurdles or increase our efficiency in certain areas, but over time it became a much more streamlined process. Our results for the time being are satisfactory, but they can definitely be improved through not only increasing the amount of evaluation, either through more metrics and guidelines, or by improvement of the NLP algorithm used to tag our identifiers, however this was much further outside of the scope of our project.

5. List of Files

Inside src/main/java

Analyser.java - Maps paths to an Abstract Syntax Tree (AST) for the source file the paths identify.

JavaParserIdentifiers.java - Extracts identifiers from classes

TypographyGuidelines.java - This file checks identifiers against typographic naming guidelines.

Utility.java - Finds all the Java source files reachable from within a specified source code repository.

Inside python_parsing

pos_checker.py - Splits every identifier into its constituent words, evaluating them against semantic identifier naming guidelines.

pos_tagger.py - This file is not relevant to the testing/evaluation process of repos. It was used to examine how spaCy processed individual words, i.e. for internal testing purposes.

splitter.py - This file splits identifiers which don't superficially follow a casing convention into their constituent words, in order to evaluate whether or not they committed casing violations.

Non-code-related

Links.docx - List of links to all the repositories we tested, as well as resources such as the libraries we used.

P67_Data.xlsx - Results spreadsheet, includes all evaluated metrics on all repositories.

Lit Review Seminar Slides.pptx - Powerpoint slides for our literature review seminar

Mid Year Video Slides.pptx - Powerpoint slides for our Mid Year Video.

Poster_67.pdf - Our A1 poster for Display Day.