Capstone Project Report

Automated Detection and Analysis of Common Preambles and Their Meanings in Source Code Identifiers

Research Advisor
Dr. Christian D. Newman

Software Engineer MS Student
Henry Keena

Date
May 5th, 2023

Automated Detection and Analysis of Common Preambles and Their Meanings in Source Code Identifiers

Henry Keena
Software Engineering MS, GCCIS, Rochester Institute of Technology
htk4363@rit.edu

Advisor: Dr. Christian D. Newman

ABSTRACT

The interpretation of identifier names is a significant problem in software development. Programmers must interpret identifier names before performing any software development or maintenance task, code search and analysis techniques use identifier names to provide useful services to developers. Having high quality identifier names reduces the amount of time developers spend reading code and increases the accuracy of techniques that use natural language information to support developers. Preambles are a particular type of token found in identifiers. Unlike typical tokens, Preambles add no new information to the meaning of am identifier's name—but instead specify certain types of behavior (e.g., pointers) or help namespace (e.g., in the case of the C programming language) identifiers to a specify module. Because preambles add no new information, they should be removed from, or at least identified in, identifiers before code analysis tries to interpret identifier name meaning. The goal of my project is to validate and augment the types of preambles described in prior work and create a technique that can automatically detect preambles.

KEYWORDS

Preamble, Natural Language Processing (NLP), Socio-Technical Grounded Theory, Hungarian Notation, Underscore, Upper Case, Lower Case, Mixed Case

1. INTRODUCTION

Code comprehension and interpretation is a critical component of modern software engineering. New engineers who join existing development teams, be it for private projects or open source software, take significant time to adjust and become familiar with the existing code base. As a result, it has become imperative for developers to become familiar with a code base as quickly as possible, so as to optimize development time and effort on part of the developers. This project attempts to tackle one of the issues facing this greater scheme problem.

Among the components of code stylization that is of considerable attention is the preamble. Preambles are tokens present in source code identifiers that do not add additional information regarding the meaning or purpose of a given identifier, only specify limited information regarding type or behavior. Because in modern software development this kind of identifier token is redundant, there becomes a need to either remove them from the identifiers, or augment them in manner so as to make them efficiently understandable by developers.

This project aims to provide a tool that can effectively, and efficiently, detect, process, and analyze source code identifiers for preambles, then catalog and categorize them by type and function. This project achieves those goals by implementing a methodology that appropriately utilizes natural language processing techniques alongside socio-technical grounded theory.

2. METHODOLOGY

In order to properly identify, collect, and analyze preambles from source code identifiers, a comprehensive methodology was devised:

2.1 Collection of Raw Data

Test data for this project came in the form of source code identifiers from 40 open source project code bases. Using a set of preexisting SCANL tools, the source code identifiers were gathered from all source files in each project. Open source projects to be used in the development of this project were determined based on two factors: programming language, and size.

Programming language was a critical consideration of the development of this project. Programming languages selected for this project were all members of the C programming language family, namely: C, C++, C#, and Java.

Additionally, test project size was a consideration. Selected projects for testing the implementation of the preamble collection tool needed to hold a significant quantity of source code identifiers. This was determined by selecting the largest projects of a given language, by source file size/count, from GitHub.

2.2 Assemble Preamble Heuristics Set

The next step in the methodology of this project, was to create an appropriate taxonomy and heuristics set to analyze a source code identifier against. These heuristics and taxonomies would consist of available characteristics of preambles as defined in previous research. Additionally, from each non single character source identifier, in every test project, the first term was collected(ie. the first term from 'strVar1' would be 'str'). These first terms were then manually reviewed to determine if they contained determined characteristics of preambles.

2.3 Conduct Identifier Preamble Analysis

Finally, once the taxonomy of preamble characteristics and first terms had been created from the test dataset, every source code identifier would be parsed, processed, and analyzed to determine if they contained a preamble. The results would then be logged into an output CSV file.

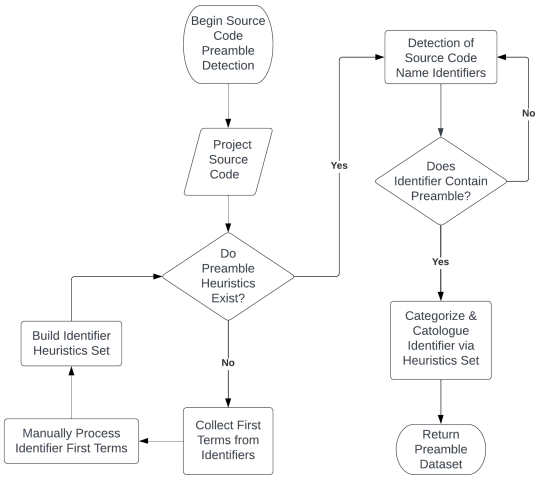

Figure 1. Project Flowchart

This project was implemented in Python(v3.11.2). Preamble detection logic was written natively, identifier collection utilized automation of a number of existing dependencies and packages.

```
USAGE:  python3 preamble_tool.py [OPTION]

OPTIONS:

Display Help/Options            --help  -h
Collect Project Data            --collect-project-data  -cpd
Collect First Term Data         --collect-first-terms   -cft
Analyze Project Data            --analyze-project-data  -apd
```

Figure 2. Preamble Collection Tool Usage Dialogue

3. Results & Discussion

The results of running the project tool implementation across the 40 project dataset resulted in the detection of roughly 3,749,701 unique identifiers. From these unique identifiers, a total of 3,636,252 were filtered and determined to not contain preambles. This left a total of 113,449 identifiers detected to contain identifiers, or roughly 0.03% of the original count of identifiers.

```
Original Number of Identifiers: 3749701
Total Number of Removed/Filtered Identifiers : 3636252
Remaining Identifiers: 113449
Generating List of Detected Identifiers with Preambles...


CSV Preamble Identifier File Written...
```

Figure 3. Preamble Collection Tool Analysis Results

From these results, further statistical analysis was conducted to determine the distribution of preambles by programming language, as well as distribution of preambles by preamble type.

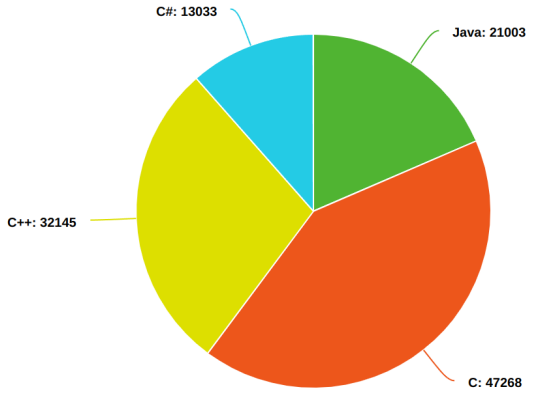

Figure 4. Distribution of Preambles by Programming Language

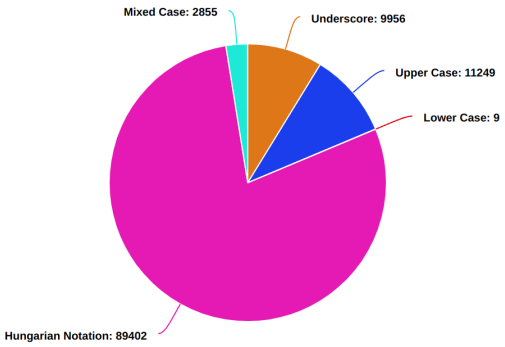

Figure 5. Distribution of Preambles by Preamble Type

4. CONCLUSIONS & FUTURE WORK

While the results of testing the preamble collection tool showed that only 0.03% of the original number of source code identifiers, this would still be considered to be high accuracy on the part of the project itself. There are a number of factors for this reasoning. Firstly, preambles are not a "standard" token, rather they could be considered akin to a code smell, or are simply poor practice, when it comes to naming or stylization conventions. Additionally, the code bases used in this project are highly extensive, but not of equal weight individually. Some of the projects used range in size from megabytes to gigabytes in scale of sheer source file size. Many of these projects are also maintained by professional developers with many years of software development experience. It can be logically extrapolated that the developers of such projects would be less likely, but not never, to push code that contain preambles in their identifiers. The results from the dataset tests demonstrated not only was the tool able to detect and process identifiers in large code bases, but that it could also appropriately categorize and analyze said identifiers.

Future work would include a more comprehensive study using the preamble collection tool, and collections from code bases of equal size and number of unique identifiers, to not only create a more complete dataset on the distribution and prevalence of preambles in code bases. Additionally, the preamble collection tool may be updated in the future to not only rely on natural language processing based heuristics, but potentially from a machine learning based approach as well.

REFERENCED WORKS

Project GitHub:

- https://github.com/SCANL/preamble_collection_tool

Published Research & Works:

1. Hoda, R. (2021, September 9). *Socio-technical grounded theory for software engineering*. arXiv.org. https://arxiv.org/abs/2103.14235
2. Newman, C. D., AlSuhaibani, R. S., Decker, M. J., Peruma, A., Kaushik, D., Mkaouer, M. W., & Hill, E. (2020, July 15). *On the generation, structure, and semantics of grammar patterns in source code identifiers*. arXiv.org. https://arxiv.org/abs/2007.08033

Implementation Dependencies:

- https://github.com/SCANL/srcml_identifier_getter_tool
- https://github.com/srcML/srcML
- https://github.com/casics/spiral
- https://pyenchant.github.io/pyenchant/

Java Dataset Projects:

1. https://github.com/jenkinsci/jenkins
2. https://github.com/junit-team/junit4
3. https://github.com/google/auto
4. https://github.com/libgdx/libgdx
5. https://github.com/elastic/elasticsearch
6. https://github.com/spring-projects/spring-bo

ot
7. https://github.com/google/guava
8. https://github.com/spring-projects/spring-framework
9. https://github.com/google/guice
10. https://github.com/ReactiveX/RxJava

C Dataset Projects:

1. https://github.com/curl/curl
2. https://github.com/openssl/openssl
3. https://github.com/torproject/tor
4. https://github.com/nothings/stb
5. https://github.com/raysan5/raylib
6. https://github.com/ggerganov/ggml
7. https://github.com/pbatard/rufus
8. https://github.com/FFmpeg/FFmpeg
9. https://github.com/Klipper3d/klipper
10. https://github.com/betaflight/betaflight

C++ Dataset Projects:

1. https://github.com/bitcoin/bitcoin/
2. https://github.com/xbmc/xbmc
3. https://github.com/google/googletest
4. https://github.com/tensorflow/tensorflow
5. https://github.com/boostorg/beast
6. https://github.com/opencv/opencv
7. https://github.com/fmtlib/fmt
8. https://github.com/WerWolv/ImHex
9. https://github.com/cemu-project/Cemu
10. https://github.com/TheAlgorithms/C-Plus-Plus

C# Dataset Projects:

1. https://github.com/Azure/azure-functions-core-tools
2. https://github.com/dotnet/aspnetcore
3. https://github.com/PowerShell/PowerShell
4. https://github.com/veler/DevToys
5. https://github.com/microsoft/Power-Fx
6. https://github.com/Ryujinx/Ryujinx
7. https://github.com/miroslavpejic85/p2p
8. https://github.com/lars-berger/GlazeWM
9. https://github.com/microsoft/MixedRealityToolkit-Unity
10. https://github.com/files-community/Files