TEDDY: AN AUTOMATED TOOL FOR DETECTION AND SUGGESTION OF PYTHONIC IDIOM USAGE

เท็ดดี้ เครื่องมือสำหรับการตรวจจับและแนะนำการใช้งานสำนวนรหัสภาษาไพธอน

BY

MR.	**PURIT**	**PHAN-UDOM**	**5988023**
MR.	**NARUEDON**	**WATTANAKUL**	**5988053**
MS.	**TATTIYA**	**SAKULNIWAT**	**5988098**

ADVISOR
DR. CHAIYONG RAGKHITWETSAGUL

CO-ADVISOR
ASST. PROF. DR. THANWADEE SUNETNANTA

A Senior Project Submitted in Partial Fullfillment of the Requirement for

THE DEGREE OF BACHELOR OF SCIENCE (INFORMATION AND COMMUNICATION TECHNOLOGY)

Faculty of Information and Communication Technology
Mahidol University

2019

ACKNOWLEDGEMENTS

We would like to give gratitude to everyone who has been involved with the development and the success of this project and thank them for their assistance throughout the entire period of this project. Dr. Chaiyong Ragkhitwetsagul, our senior project advisor, Asst. Prof. Dr. Thanwadee Sunetnanta, our Co-advisor, and Dr. Pisit Paiwattana, for the help in the technical part of the project. We would also like to thank the Faculty of Information and Communication Technology, instructors, staff, members for the support given to us. Lastly, we would also like to thank our families for giving us support while making this project. The project would not have been this successful without those mentioned and their support that was offered.

Mr. Purit Phan-udom
Mr. Naruedon Wattanakul
Ms. Tattiya Sakulniwat

TEDDY: AN AUTOMATED TOOL FOR DETECTION AND SUGGESTION OF PYTHONIC IDIOM USAGE

MR. PURIT	PHAN-UDOM	5988023 ITCS/B
MR. NARUEDON	WATTANAKUL	5988053 ITCS/B
MS. TATTIYA	SAKULNIWAT	5988098 ITCS/B

B.Sc.(INFORMATION AND COMMUNICATION TECHNOLOGY)

PROJECT ADVISOR: DR. CHAIYONG RAGKHITWETSAGUL

ABSTRACT

In the present day, software development has been one of the main focuses for Thailand 4.0 with the aim to integrate technology and industrialization. In this field, many programming languages are being used and each language has its unique format called “coding idioms”. When programmers start learning a new language or switch between many programming languages, there might be some unfamiliarity that causes the produced program to drop in quality, or be difficult to read and understand.

With the problems stated above, we are inspired to create a software tool, named “Teddy”, that can help in checking the quality of idiom usage during development of Python programs. Python is a popular programming language and it is well-known for its unique set of coding idioms. Teddy helps programmers detect coding idioms within their source codes and report the usage of those idiomatic code snippets. Teddy is designed to offer two modes of operation - prevention and detection mode. The prevention mode supports real-time idiomatic code detection during code review time in GitHub pull requests, while the detection mode runs a thorough scan of idiomatic and non-idiomatic code over historical commits.

A series of tests and evaluation has been performed, using real software projects, to assess the performance of the tool. It has been found that Teddy has high precision for detecting idiomatic and non-idiomatic Python code usage. The visualization can provide developers insightful information regarding the evolution of idiomatic and non-idiomatic Python code in their projects. We hope this tool will allow the programmers to understand more about idiomatic Python usage and also help naive programmers in their learning of Python language.

KEYWORDS: PYTHON, CODING IDIOMS, AUTOMATED ANALYSIS

79 P.

เท็ดดี้ เครื่องมือสำหรับการตรวจจับและแนะนำการใช้งานสำนวนรหัสภาษาไพธอน

นาย ภูริช พันธุ์อุดม 5988023 ITCS/B

นาย นฤดล วัฒนกูล 5988053 ITCS/B

นางสาว ทัตติยา สกุลนิวัฒน์ 5988098 ITCS/B

วท.บ. (เทคโนโลยีสารสนเทศและการสื่อสาร)

อาจารย์ที่ปรึกษาโครงการ: ดร. ชัยยงค์ รักขิตเวชสกุล

บทคัดย่อ

ปัจจุบันการเขียนซอฟต์แวร์เป็นหนึ่งในการตอบรับนโยบายประเทศ 4.0 ที่มีเป้าหมายในการพัฒนาประเทศโดยการประยุกต์ใช้เทคโนโลยีร่วมกับภาคธุรกิจและอุตสาหกรรม ในส่วนของการใช้เทคโนโลยีนั้นมีภาษาที่ใช้ในการเขียนซอฟต์แวร์ต่าง ๆ อยู่หลายภาษา แต่ละภาษานั้นจะมีรูปแบบการเขียนที่แตกต่างกันเรียกว่า สำนวนรหัส หรือ coding idioms เมื่อนักพัฒนาซอฟต์แวร์เริ่มต้นเรียนรู้ภาษาใหม่หรือเปลี่ยนจากการเขียนภาษาโปรแกรมหนึ่งไปยังอีกภาษาหนึ่งอาจจะมีความไม่คุ้นชินที่ส่งผลให้คุณภาพของซอฟต์แวร์ที่เขียนขึ้นมาลดลง หรืออาจจะทำให้การทำความเข้าใจโปรแกรมนั้นมีความยากลำบากเพิ่มขึ้น

จากปัญหาที่ได้กล่าวมาข้างต้น ทางทีมพัฒนาจึงมีความต้องการที่จะสร้างเครื่องมือในการช่วยเหลือการตรวจสอบและประเมินคุณภาพการเขียนโปรแกรมด้วยภาษาไพธอนพร้อมทั้งแนะนำรูปแบบที่เหมาะสมโดยอัตโนมัติ ซอฟต์แวร์ดังกล่าวมีชื่อว่า Teddy โดย Teddy นั้นจะสามารถช่วยให้ผู้เขียนซอฟต์แวร์ด้วยภาษาไพธอนสามารถตรวจสอบเพื่อค้นหาปัญหาจากการเขียนโปรแกรมและนำเสนอตัวอย่างที่ถูกต้องให้นักพัฒนาโปรแกรมเข้าใจรูปแบบการเขียนที่ถูกวิธี ทำให้การเขียนซอฟต์แวร์นั้นเป็นไปได้อย่างมีประสิทธิภาพมากขึ้น โดย Teddy นั้นได้ถูกออกแบบให้มีระบบการทำงานแบ่งเป็น 2 แบบ ได้แก่ แบบที่ 1 คือแบบป้องกัน (prevention mode) ในรูปแบบการทำงานนี้ Teddy จะทำการหาการใช้ coding idiom ในเวลาจริงจาก pull request ที่เกิดขึ้นใหม่ และแบบที่ 2 คือแบบตรวจจับ (detection mode) ที่จะทำการค้นหาการใช้ coding idiom จากทุกเวอร์ชั่นตั้งแต่อดีตจนถึงล่าสุดของซอฟต์แวร์โปรเจคดังกล่าว

เพื่อทำการทดสอบและประเมินประสิทธิภาพของเครื่องมือที่ได้พัฒนา ได้มีการใช้โปรเจคซอฟต์แวร์จริงในการวัดระดับการทำงาน จากผลการทดสอบนั้นพบว่า Teddy สามารถตรวจจับการใช้ coding idiom ด้วยความแม่นยำที่สูง แผนภาพที่ถูกสร้างขึ้นจากเครื่องมือนั้นจะช่วยให้ผู้

พัฒนาทราบถึงข้อมูลเชิงลึกเกี่ยวกับการพัฒนาของโค้ดภาษาไพธอนทั้งแบบ idiomatic และ non-idiomatic ภายในโปรเจคนั้น ๆ โดยที่ผู้พัฒนาเองก็จะสามารถเข้าใจเกี่ยวกับ idiom ภาษาไพธอนให้ดียิ่งขึ้นและยังช่วยในการเรียนรู้เกี่ยวกับภาษาไพธอนอีกด้วย

79 หน้า

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF LISTINGS

Page

CHAPTER 1
INTRODUCTION

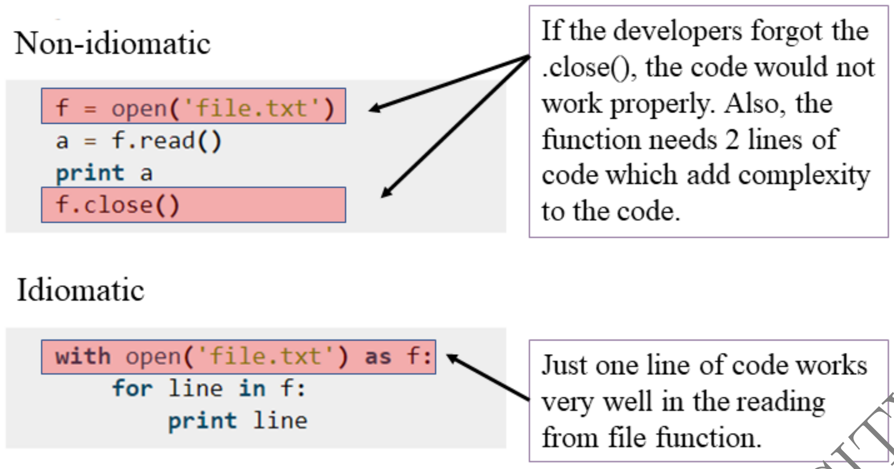
This chapter introduces the overview of this senior project report. It includes the motivation of the project, the problem statements that we tackle, the objectives of the project, the scope of the project, and the project's target users respectively. Lastly, the report structure is laid out for which each chapter contains the corresponding contents.

1.1 Motivation

Nowadays, Python is one of the most commonly used programming languages worldwide [2]. Python developers include those who are already settled in the community, the new ones who have just started learning how to write programs, and the experienced ones from other languages that are transitioning to Python. Each of them have different styles of writing their code.

The programmers who are strongly familiar with Python write their code in a so-called "Pythonic" way, which is a well-accepted way of writing and proven to be readable and efficient. The naive programmers who have just started writing programs may write in the most simple ways as much as possible. The proficient programmers, who are transitioning from other languages to Python, have the tendency to apply writing styles that are practical in their old programming language syntax, but considered to be unconventional in Python language.

When programmers write programs in Python language and do not adopt idiomatic Python into their coding, it *may* cause the program to be inefficient or difficult to read by the other Python programmers. One example of such usage is `f = open('file.txt')` and `f.close()` as displayed in Figure 1.1. These statements within the program can cause problems when programmers forget the to add in `f.close()`. As in many programs, not using the `f.close()` would cause memory leaks in which the overall usable memory to be left allocated to an unused part of the program. On the contrary, the proper Pythonic idiom for this scenario would be `with open() as f:`


Non-idiomatic
If the developers forgot the .close(), the code would not work properly. Also, the function needs 2 lines of code which add complexity to the code.
Idiomatic
Just one line of code works very well in the reading from file function.

```
f = open('file.txt')
a = f.read()
print a
f.close()
```

```
with open('file.txt') as f:
    for line in f:
        print line
```

Figure 1.1: Comparison between idiomatic and non-idiomatic Python coding style in `with open` case study

instead, which would help the program perform better [3].

In the software development cycle, using Pythonic way of writing code within the development phase gives the code reviewers a better understanding of the program and a smoother workflow. This is in regards that experienced reviewers would have a better understanding if the codes provided are Pythonic. If there is a software program which can assist the use of idiomatic Python coding style, the number of highly-readable Python code will be increased and so the effort of the Python code reviewers is reduced.

With the reasoning above, we have had the idea of creating "Teddy" as a software that would help in the detection and prevention of misusing idiomatic Python coding styles in software. The users of the software would be able to check their code quality in terms of idiomatic coding style. The tool also helps the new programmers to learn and understand what their flaws are when using Python as a programming language. We, as the developers of Teddy, believe that Teddy can enhance the standards for Python language and give the developers, both in Thailand and around the world, more understanding of Python, which is a worldwide and one of the most widely-used programming languages.

1.2 Problem Statement

This project tackles the following problems in nowadays' software development:

1. There are only a few studies on idiomatic Python codes during the evolution of a

software project.

2. Many naive Python programmers and transitioned experienced programmers from other languages to Python are not aware of idiomatic Python coding style. They also do not know when and how to use them in their day-to-day programming tasks.

3. The manual process of source code reviews by experts takes a lot of time and becomes a tedious routinely task.

While coding in Python, many programmers are not aware of idiomatic Python coding style, where it is actually considered to be a significant factor to the readability and efficiency of the program. The unawareness can potentially lead to certain problems such as unnecessary computational resource consumption, and functional flaws in the software program. By having a software that can guide the use of idiomatic coding style, source codes will be using a standardized format and give other developers the same understanding of the codes written.

1.3 Objectives of the Project

This project aims to satisfy the following objectives.

1. To create a software program that can detect idiomatic/non-idiomatic Python code with high accuracy.

2. To create a software program that can analyze software projects in an online source code version control system called GitHub and provide real-time results on idiomatic and non-idiomatic code statements.

3. To create a software program that can visualize the detected idiomatic/non-idiomatic Python code usage with a friendly graphical user interface.

1.4 Scope of the Project

The scope of this work is as follow:

1. The proposed tool and techniques allow the users to detect idiomatic Python within the source codes automatically.

2. The proposed tool allows the users to see information from GitHub through the software user interface.
3. The proposed tool is able to be integrated with GitHub and provide a full analysis report.
4. The proposed tool is designed to work with Python source code files only.
5. The proposed tool relies on GitHub API, and hence it only supports GitHub as the data source.

1.5 Target Users

The project provides numerous benefits within the programming field. Thus, our target users consist of programmers varying in their field of work. Our project would also provide benefits in the educational field. The tool would allow teachers and students to gain benefits in understanding idiomatic Python coding style. By giving an accurate, fast and automated code review, programmers would be able to obtain a preliminary review on their codes before sending it to a reviewer. Teachers would be able to have students use the tool to gain a better understanding in the usage of idiomatic coding style in Python.

1.6 Report Structure

This document consists of six chapters in total. Chapter 1 is this introductory chapter. It includes the motivation, problem statement, objective, scope and target users of our project, and the report structure. Next, Chapter 2 discusses the background concepts, related works, and tools and methods involved in this project. Chapter 3 then explains the design aspects of our works, with the system architecture, use case diagram and data flow diagrams. Chapter 4 is the detail of our implementation, followed by Chapter 5 being the result of testing and evaluation. Lastly, Chapter 6 summarizes and conclude the work on this project.

CHAPTER 2
BACKGROUND

With this project aiming to create a software for detecting and analyzing the usage of idiomatic Python coding style, the understanding of the related fundamental concepts is important. This chapter discusses the key topics related to the project, which includes definitions and keywords, fundamentals, related works, and tools and methods.

2.1 Definitions and Keywords

1. *Software Program*: A software program refers to a set of computer instructions that is designed to carry out certain tasks, which may or may not receive input or interact with human as the user of the program.

2. *Source Code*: Source code refers to a sequence of valid textual commands that can be compiled or assembled into an executable software program. Source code is written using a human-readable programming language.

3. *Coding Idiom or Idiomatic Coding Style*: It is stated in Stack Overflow, the online community of programmers, that idiomatic code means the code that following the conventions of the language [4]. Instead of using other knowledge from other languages that they have known, the programmers would write the code in the most effortless and acceptable way in that language's community. For example, in English, "a piece of cake" means something that is very easy which cannot be translated directly as an actual piece of cake but can depict the understanding of situation clearly. The explanation also works for coding idioms; sometimes what may have written code to perform a specific function in a short style that is allowed by the programming language syntax. For example, in C, to write *increment* of a variable, the developer can declare the statement `i=i+1` which can be written in idiomatic C code as `i++`.

4. *Idiomatic Python (IP) Code*: Python code written to execute a particular function

by following the Python language coding idioms that are well-accepted in the community [5]. The term “idiom”, in general, means a group of words or phrases that, altogether, have a particular meaning rather than a verbatim definition. Therefore, “idiomatic Python” refers to a set of syntax commands that, altogether, executes a particular function by following the principle of Python language.

5. *Non-Idiomatic Python (NIP) Code*: Python code written to execute a particular function but does not follow the Python language coding idiom principle, in which it can be replaced with a proper idiomatic Python code that yields similar functions and outputs.

6. *Online Repository (or Repository)*: An online repository refers to a website which provides online storage, sharing and managing of source codes as services. GitHub is a famous example of an online repository hosting website with its built-in `git` version control system.

2.2 Fundamentals

This section explains the concepts and background knowledge in relation to the project.

2.2.1 Python Programming Language

Python is a high-level programming language first appeared in 1990 [6]. It has been widely used in many fields of information technology research, such as deep learning, distributed computing, multimedia processing etc. In the last 10 years, Python has been one of the 10 most popular programming languages in the world, now ranked at number 2 [7]. Python language is an open-source code with its set of comprehensive libraries and add-on packages that facilitate different operations for different applications.

Python is practical with many programming paradigms - procedural, object-oriented, and functional. The design of Python syntax commands focuses on the inclusion of “significant white space”. Unlike other languages where the presence of white space is ignored or disregarded, white space such as indentation signifies a grouping of code, called a “block”, that is executed within an encapsulating declaration - function, conditional, iteration.

2.2.2 Benefits of Idiomatic Python Code

The usage of idiomatic Python code brings about benefits to the nature of software source codes and syntax's readability from programmers' point of view. In the study of Alexandru et al.[8], idiomatic code can improve the performance and readability of the source code.

The example of comparing between idiomatic and non-idiomatic Python code is shown in Figure 1.1 as `with open` is one of the most popular idiom used referenced in the referenced study. As it is stated in the paper, the three most used idiomatic Python from 1,000 GitHub repositories are (1) `list` comprehension with the usage of 866 repositories, (2) `with` statement with the usage of 848 repositories, and (3) `decorator` with the usage of 765 repositories.

2.2.3 Code Clones

Code clone refers to a code fragment — a sequence of source codes — that is similar to another code fragment by some given definition of similarity. The two similar code fragments then form a "clone pair". Clones occur as the result of programmers reusing or reproducing source codes from external sources, and apply those codes to their own programs [9].

Four different types of clones have been defined, in other words the definitions of how two code fragments are regarded as similar. Type-1 clones are identical code fragments which are different only by white spaces, layouts or comments. Definition Type-2 clone extends that of Type-1 by considering additional variation in identifiers, literals, and types. Type-3 takes into account Type-1 and Type-2 with added, changed, or removed statements. Lastly, Type-4 are syntactically different code fragments which yield the same computational results.

Consequential issues may follow code cloning without caution and knowledge about the duplicated syntax. Often the original code fragment contains flaws or errors. Once the *buggy* source code is reproduced, so are the errors and bugs hidden inside. The widely-spread errors now become difficult to track and correct, even when the original source of the code fragment gets fixed. Another problem of code cloning is that the duplicated codes may violate the copyright or license of those programs. As the result, code clone detection and prevention now plays a significant role in quality control of software development.

The process of code clone detection follows a sequence of steps. Pre-processing is the first step for which any irrelevant code segments are removed, and the definition of "comparison unit" is set. Next is the transformation step where the pre-processed codes are converted into some intermediate representation other than the original textual format, including extraction and normalization. The transformed code is then fed into a comparison algorithm as for the match detection step. After this, the list of resulting clone pairs is formatted to align with original textual code, parsed through post-processing and filtering by manual analysis or automated heuristics, and aggregation that combines clone pairs into clone classes.

Of all the code clone detection tools, many techniques and technical approaches are implemented. Four outlining groups of the approaches are

1. *Textual*: The source codes undergo little to no transformation before the actual comparison takes place. Often the raw source codes are used directly to find clone pairs.

2. *Lexical*: Also known as"token-based technique", the comparing source codes are converted into "tokens" first, using tokenizer and normalizer modules. Then the identical sub-sequence of tokens are scanned where matching units are returned as clones. This technique is more sensitive to minors changes in white spaces and formatting.

3. *Syntactic*: A parser is used to convert source codes into parse tree, or "abstract syntax trees" (ASTs). The generated trees are then processed to search for clones by either (1) tree matching (finding similar sub-trees) or (2) structural metrics (comparison of metric vectors).

4. *Semantic*: For the semantic approach, the source code is represented as a program dependency graph (PDG). The nodes suggest expression or statement, while edges

are control and data dependencies. Discovery of clones is by looking for "isomorphic sub-graphs" - sub-graphs containing the same number of nodes connected in the same way - from two comparing PDGs.

In Listing 2.1 is a snippet of bubble sort source code in Java language. From the raw source code, demonstration of its abstraction as other forms are shown. Figure 2.1 depicts the respective program dependency graph representation, where each node is connected by two types of dependency edge - data and control. From Figure 2.2, the source code is converted into an abstract syntax tree.

Listing 2.1: Java implementation of bubble sort algorithm

```
public int[] BubbleSort(int[] ar) {
        int temp;
        for (int i = ar.length-1; i > 0; i--) {
                for (int j = 0; j < i; j++) {
                        if (ar[j] > ar[j+1]) {
                                temp = ar[j];
                                ar[j] = ar[j+1];
                                ar[j+1] = temp;
                        }
                }
        }
        return ar;
}
```

In addition to above, some code clone detection tools utilize characteristics of two or more approaches mentioned, which is considered to be a hybrid combination. However, such trend is rather unpopular.

Since our work is centered around idiomatic Python detection, therefore it is logically relevant that the code clone detection pipeline is to be borrowed and appropriately adopted into the steps of coding idiom detection. In this project, we apply code clone detection to search for code snippets that are similar to our predefined set of idiomatic and non-idiomatic Python code snippets.

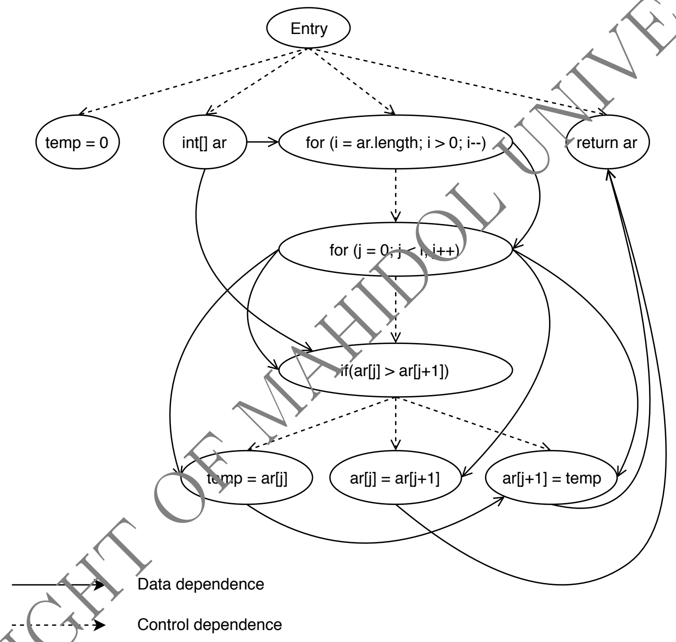


Figure 2.1: Program dependency graph representation of Java bubble sort code

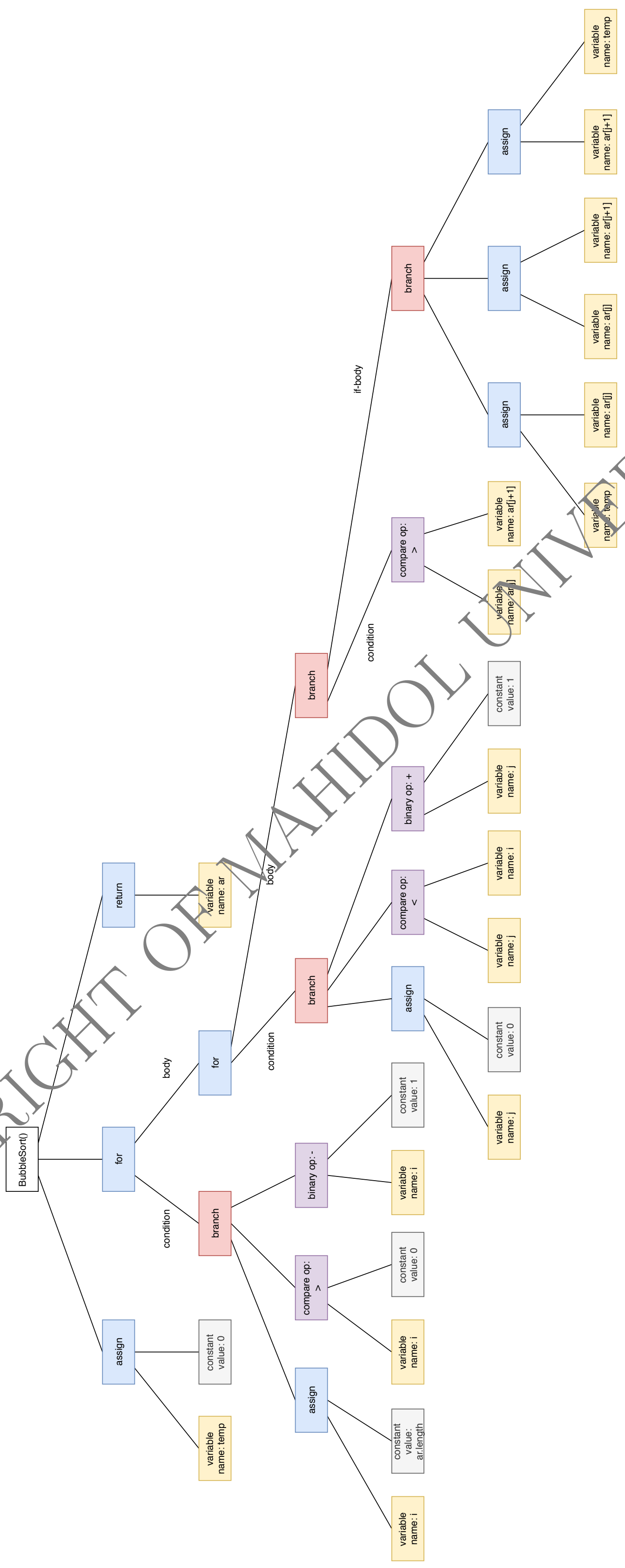


Figure 2.2: Abstract syntax tree representation of Java bubble sort code

2.3 Related Work

This section discusses the works closely related to the research of this project.

2.3.1 On the Usage of Pythonic Idioms

Pythonic way of coding is not concretely defined yet. In the book "Zen of Python" by Tim Peters [10], it is mentioned as the obvious way to do the preferable coding style. On the other hand, this paper by Alexandru et al. [8] states that Pythonic being the idiomatic style for developing Python. The researchers divided the experiment into two parts: interviewing Python developers on understanding the concept of IP and the study of the usage of IP over 1,000 GitHub repositories.

The result of the second part of the approach shows the popularity and improvement of performance or readability of idioms used in the selected 1,000 projects. The researchers of the paper listed and classified the idioms that are used in those 1,000 projects and also make the collection in the website [11]. The information would be the foundation for our tool to detect the usage of IP in a given project.

2.3.2 Diggit: An Automated Code Review Tool

Code reviewing can be seen as a common practice in many software development teams, this would also include the reviewing from communities as well. Code reviewing thus became a core practice within the software development cycle. With code reviewing becoming apparent, sometimes there are problems when many codes are being edited in different areas. Developers would use online version control services such as GitHub to control these problems.

Although GitHub can be seen as an answer to the coding review problems with a large number of contributors, the problem of verifying and going through the edits made is still considered to be tedious. Thus, the solution that Diggit uses is to have the GitHub Bot communicate directly with GitHub to gain access to changes or interactions made to the codes that are hosted on the GitHub service, allowing the manipulation and viewing from an automated system.

With the help from the GitHub API, Diggit has created a system that integrates with a version control system, in which it checks using association rules when activities have been made on the GitHub repository [12]. It then helps in the phase where code reviewers check on the edits made within the code and fasten the reviewing process. Our project uses the same method as Diggit, which is using an automated system to interact with a GitHub repository during a pull request and access the information within the pull request.

2.3.3 Toxic Code Snippets on Stack Overflow

With software development being used in many fields, more programmers are becoming more reliant on the usage of online platforms, such as Stack Overflow [13]. The code snippets that are found on such platforms are considered by many to be usable, but in reality there exists many flaws, outdated methods, or even licensing problems contained within the snippets [14]. These outdated codes with flaws can then be classified as toxic code snippets which are generally unwanted within the source codes.

Toxic code snippets are commonly seen posted in online forums, mostly in the commonly used platform Stack Overflow. Most of these snippets are not directly from Stack Overflow, but copied from other sources such as an open source software [14]. Even when there are many toxic code snippets located in Stack Overflow, and they are considered harmful to use, users themselves acknowledge their existences and are aware of these snippets [14].

In the research of Toxic code snippets on Stack Overflow, it focuses on the usage of codes that are considered to be harmful, bearing some similarities to our project that focuses on the usage of harmful non-idiomatic codes in Python.

2.4 Tools and Methods

The following section will contain the tools and methods we have used in the development of Teddy.

2.4.1 GitHub

GitHub [15] is an online platform which allows the users to be able to interact with their online storage called "repository". The repositories are used to store and organize source codes. Users are able to view, edit and give contributions to those repositories as a community. With over 40 million developers [16] on GitHub (as of September

30, 2019) and the increasing trend of creating new repositories being at 44 percent [17] more than the year 2018 (totaling in over 100 million repositories as of August 2019) [16], therefore we have chosen GitHub to be our target repository.

GitHub adopts a version control system called `git`, which was introduced during the time when source code version control was not widely known and used. GitHub started to attract developers' attention when the system allowed users to be able to work remotely and have a version control system that provides an ease of access to the developers.

2.4.2 Probot

Probot [18] is a framework created to help giving users flexible access to the usage of GitHub Apps by using Node.js. The framework aims to help users in using GitHub Apps by handling the sophisticated "webhooks" and GitHub's tedious authentication system. With an understandable and easy to write style of coding, it allows users to gain an understanding relatively fast.

2.4.3 Smee.io

Smee.io [19] is a webhook service provider, which was created by the Probot development team to allow users to create webhooks connecting to GitHub and sends those information back to the users. The webhook created by Probot is an access point which allows other systems to interact and trade information with each other using HTTP access. Smee.io can be used in a scenario such as when a system requests to gain access to information on pull requests from GitHub. The system would connect to the Smee.io webhook, which monitors the activity on GitHub and relays information to the system.

2.4.4 Elasticsearch

Elasticsearch [20] is a distributed, RESTful search and analytic engine, and a data storage itself, that relies on Apache Lucene infrastructure and library [21]. The exchange of user's queries and results is through REST API commands. It provides the schema-less JSON document formatting for the retrieved documents, through HTTP web service.

Inside Elasticsearch, data is stored as JSON documents. Documents which are related to each other are grouped and organized in a unit of “index”. First, the raw data is parsed, normalized and pre-formatted prior to indexation process. It utilizes “full-text search” that uses “inverted-index”, a data structure where for every unique word there is a corresponding list of documents whom the word appears in. And because Elasticsearch simultaneously updates the inverted-index as new documents are being indexed, therefore the search-able delay after indexing is very small, almost in real-time.

2.4.5 Git Version Control System

A version control system (VCS) is a software tool capable of managing edits and changes, made by a team of programmers, on the source code over time [22]. A VCS has the ability to keep track modifications, compare two versions of the same pieces code from two different times, and revert or merge versions of source code. One of the most popular VCS today is `git`. In `git`, a series of versions, where one is the result of modifying a preceding version, forms what is called a commits. `git` also provides “branches” which are separately working versions of the same software projects under VCS. Branches of the same software project can be worked on independently and synced when needed.

Master Branch

Master branch is the permanent primary branch for every source code whom a version control system is applied to. Source code with VCS is required to have at least this one branch. The master branch represents the root mainstream, where every version of it has passed testing, and all of the source codes are in a stable, ready-to-deploy state. Version updates on this branch should be not as frequent as other developing branches that serve as experimental or testing prototypes.

Developing Branch

Developing branch is a secondary branch that diverts from a pre-existing root branch. It can track version changes within its own branch independent from the root branch. The purpose of developing branch to isolate the experimental or unstable features from the more stabilized branches, or other unrelated branches. Once the development or changes of a developing branch ended, the final content can then be merged into the root branch or master branch, through a "pull request".

2.4.6 Siamese

Developed by Ragkhitwetsagul C. and Krinke J., "Siamese" is a code clone search tool with high degree of scalability via multiple code representation [1]. The tool is written in Java and supports clone detection in Java and Python language. It works with Elasticsearch 2.2.0 to index and retrieve source codes needed to perform clone detection process. Siamese is capable of detecting Type-1, Type-2, Type-3 and Type-4 by transforming raw source code into intermediate representation format as *tokens*, using *parser*, *normalizer* and *tokenizer* for the respective programming languages. The tool does not provide a graphic interface and can only be controlled via command-line interface. Figure 2.3 depicts the architecture of Siamese program, incorporating the small steps from indexing and retrieval, to outputting the results of clone search.

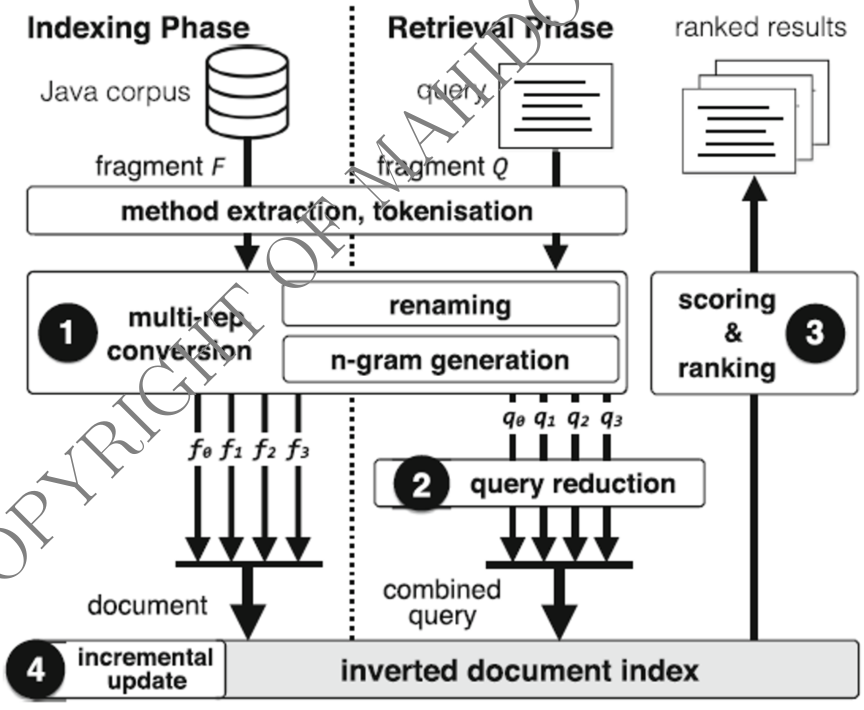


Figure 2.3: Siamese architecture [1]

CHAPTER 3

ANALYSIS AND DESIGN

This chapter covers the analysis and design including the explanation of system architecture, use case diagram, and data flow diagrams (level 0 and level 1).

3.1 System Architecture

The software aims to achieve two different processes, each handling one feature. As seen in Figure 3.1, one process handles "prevention mode". Another process is shown in Figure 3.2, which handles "detection mode". In "prevention mode" the task would be to detect IPs and NIPs before they are integrated into a software system by extracting the newly added code in a pull request and querying through a database of known IPs/NIPs. While in "detection mode", the task is to detect IPs and NIPs in the historal commits. The source code in each commit is used as a search database and queried by a set of known IPs/NIPs.

3.1.1 Detection Mode

In detection mode, Teddy is given a GitHub repository information and the tool acquires the source code to be analyzed. The source code is sent directly to Teddy through a set of bash scripts in which Siamese analyzes the source code for the usage of IPs and NIPs. After that the results are sent to a visualization module to show the users the IPs/ NIPs that have been detected. Figure 3.3 shows an abstract view of data flow in detection mode.

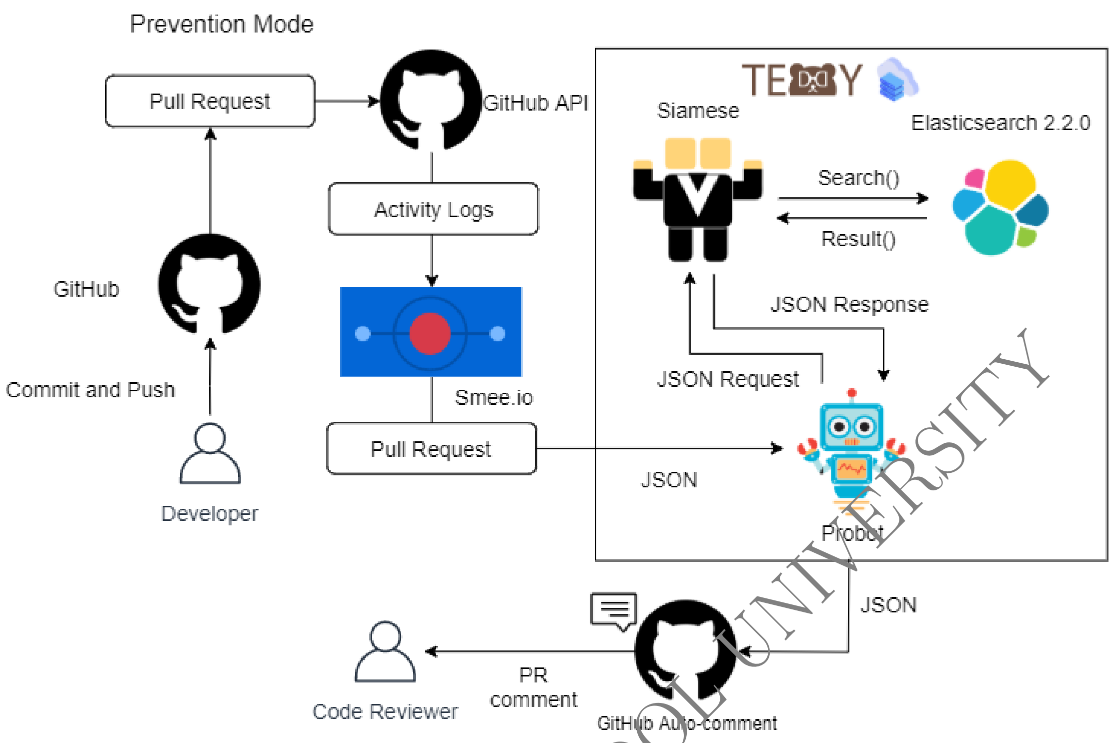


Figure 3.1: System architecture of Teddy prevention mode

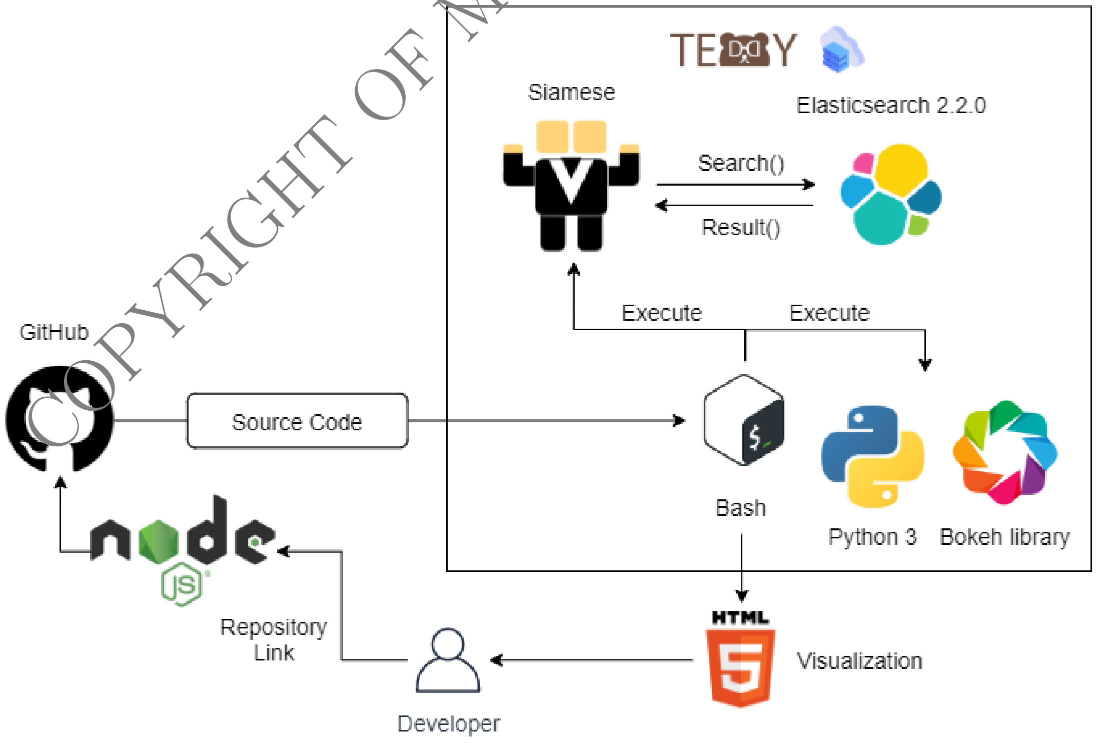


Figure 3.2: System architecture of Teddy detection mode

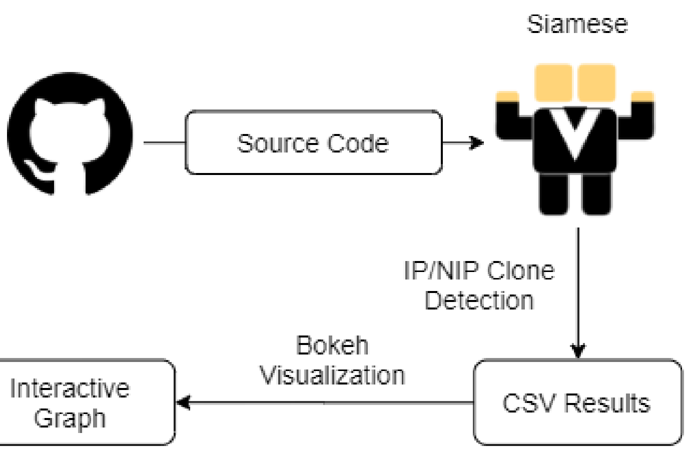


Figure 3.3: Simplified view of detection mode

3.1.2 Prevention Mode

In prevention mode, the software is designed to be active at all times and run on a real-time basis. Based on a simple overview in Figure 3.4, the prevention mode is also divided into two major parts which are the communication part with GitHub and the analysis part.

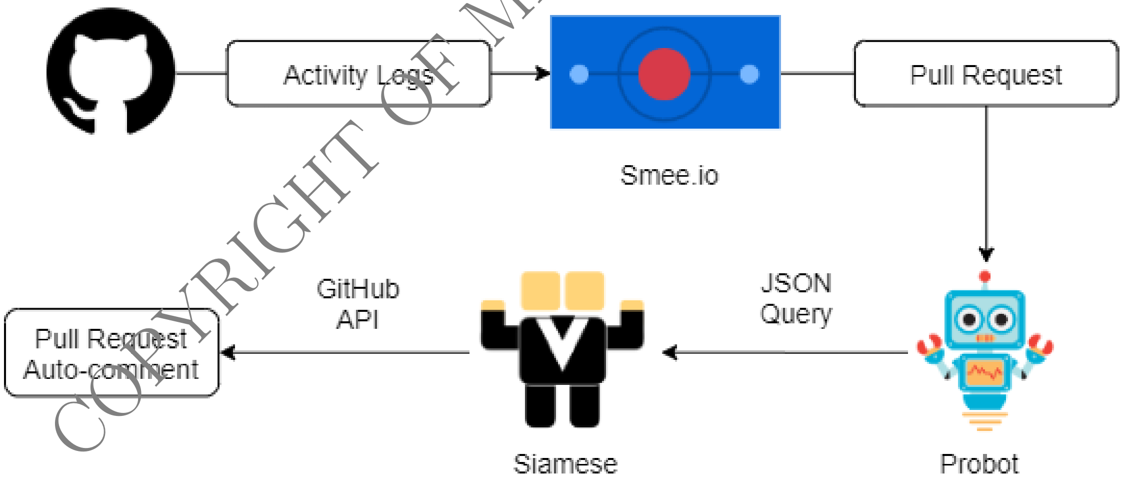


Figure 3.4: Simplified view of prevention mode

Communication with GitHub

In the first part will be the modules that works with external services outside of Teddy. Whenever GitHub has any activities, there will be information sent to Smee.io, and Teddy would then receive the information that was sent to Smee.io regarding pull requests and parse only the updated code snippets.

Analysis within Teddy

After the code snippets are extracted, Probot will send the code snippets as JSON files to Siamese. Siamese will then compare the code snippets and use it to obtain relevant *tokenized* Pythonic idioms within the database. Finally in a comment on the pull request, users will be able to see the what parts of the snippets are non-idiomatic Python code.

3.2 Use Case Diagram

In the use case diagram, Figure 3.5, Teddy interacts with three external actors and entities - GitHub, Developer, and Code Reviewer. There are three use cases.

The first use case, “Specify GitHub Repository”, is when the developers specifies a wanted repository from GitHub (both for the prevention and detection mode). GitHub is the secondary actor that receives the request from Teddy, and in return relays back the access to that repository.

Next, in “See Pull Request Analysis Report From Prevention Mode”, if the user chooses to use the prevention mode, a new pull request of the specified GitHub repository will be analyzed by the Teddy system. After that, it will generate a report and distribute the report to both the code reviewer and the developer themselves.

Finally, in “See GitHub Commit Visualization From Detection Mode”, the commits in the GitHub repository will be pulled into the system and analyzed for the usage of IPs/NIPs in every commit in its history. Then, the system will generate a visualization and display the output to the user.

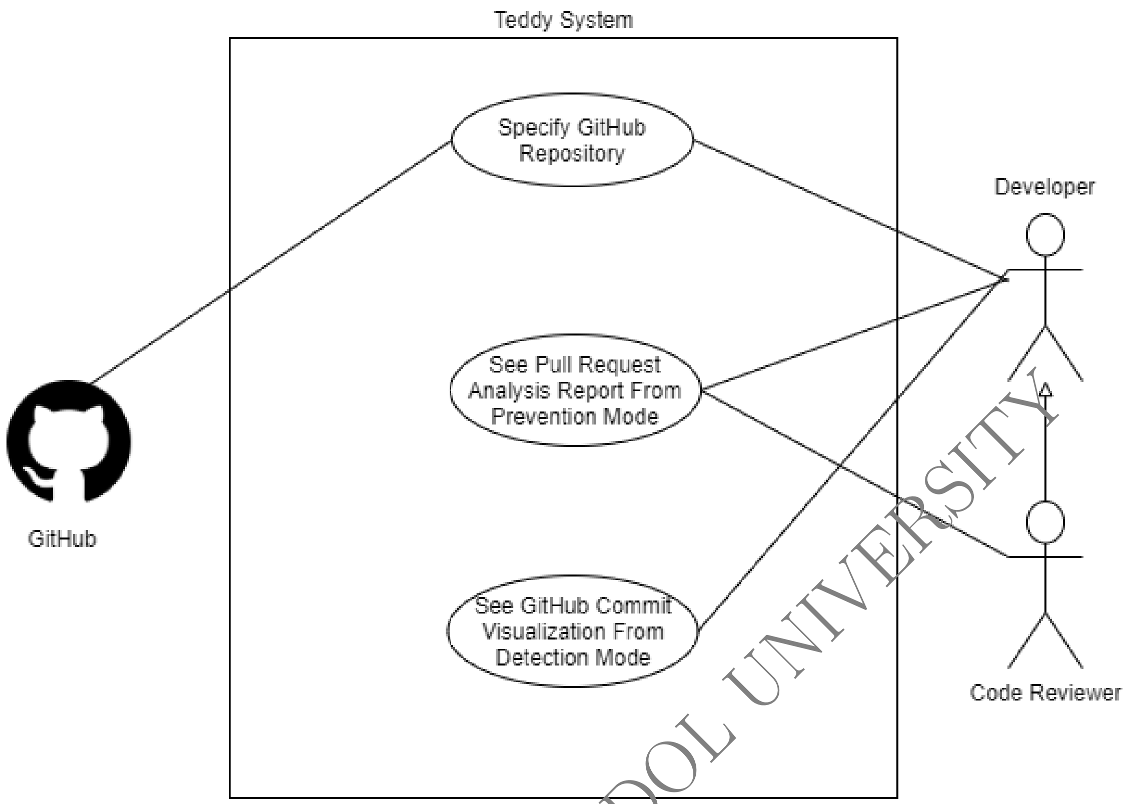

Teddy System
Specify GitHub Repository
See Pull Request Analysis Report From Prevention Mode
See GitHub Commit Visualization From Detection Mode
GitHub
Developer
Code Reviewer

Figure 3.5: Use case diagram of Teddy system

3.3 Data Flow Diagram

We analyze the Teddy system by using data flow diagram. The Data flow diagrams have been drawn up in accordance with the two operational modes - prevention and detection. Two levels of the data flow diagram, level 0 and level 1, have been made for each mode.

3.3.1 Level 0

In prevention mode, Figure 3.6, the data comes from three external entities: GitHub which is the repository, Code Reviewer, and Developer of the project. Once the developer chooses the source code to be analyzed by the tool, the request for source code will be sent to the repository, then the pull request log and the source code will be sent to the system from the repository. After that, the system will analyze the usage of IPs/NIPs in the pull requests and generate the report. Finally, the report will be sent to the code reviewer and the developer.

In detection mode, Figure 3.7, the external entities are only GitHub and Devel-

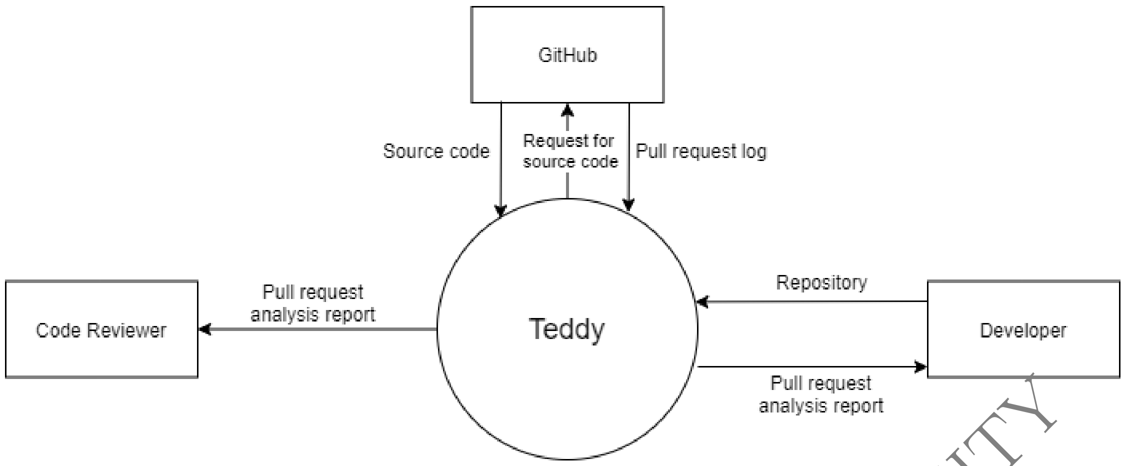


Figure 3.6: Data flow diagram level 0 of Teddy system in prevention mode

oper. Firstly, the developer would enter the GitHub URL to let Teddy send the URL to obtain the repository's source code. Then, Teddy extracts all of the historical commits to use in the analysis step. Finally, after getting the commits, Teddy would generate the visualization with IPs and NIPs and send the visualization back to the developer.

3.3.2 Level 1

According to Figure 3.8, the inner part of prevention mode composes of four main processes: 1. Get source code from repository 2. Extract code snippet 3. Search idiom 4. Generate report. Firstly, at process number 1, the external entity would select the repository to be analyzed by the system. The request for the pull request would be sent out to the repository and the pull request log, along with the source code, will be sent back to the system. Then, they will be passed on to the next process, extracting code snippet, which will extract out JSON snippet to be used in the next process. Then, along with the source code, the snippet will be sent to the search idiom process to search for idiom with the internal idiom database. The returned result is in the relevant JSON result. After the search, the result will be generated into the full detailed report, at process number four, and sent back to the external entity outside of the system.

For detection mode (Figure 3.9), the inner part of the system consists of three main processes: 1. Get GitHub repository, 2. Extract GitHub commits, 3. Search idiom, and 4. Generate visualization. For the first step, the system would receive GitHub URL as an input from the external entity, then pass the URL to GitHub to get hold of the

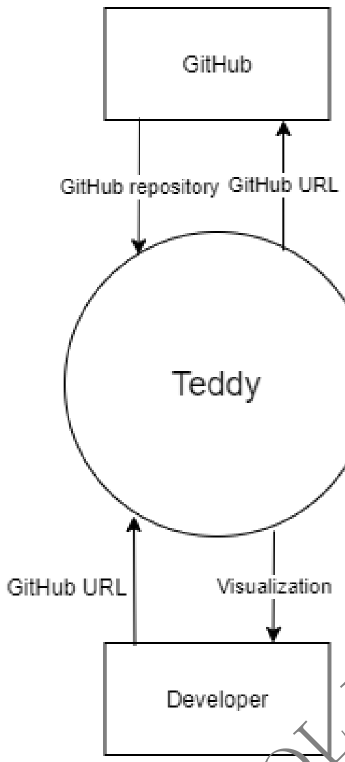


Figure 3.7: Data flow diagram level 0 of Teddy system in detection mode

GitHub repository. After that, the second procedure starts. The system would obtain commits from the repository and pass them to process number three, search idiom. In this process, the system will pull IPs and NIPs snippets from the idiom database then use them as queries to search for the usage of those IPs and NIPs in cloned repository database. After received the search result, the system will pass the result in the form of CSV file to step 4, which has the responsibility on generating the visualization. Finally, the system will generate the visualization in the form of a HTML page based on the information extracted from the commits.

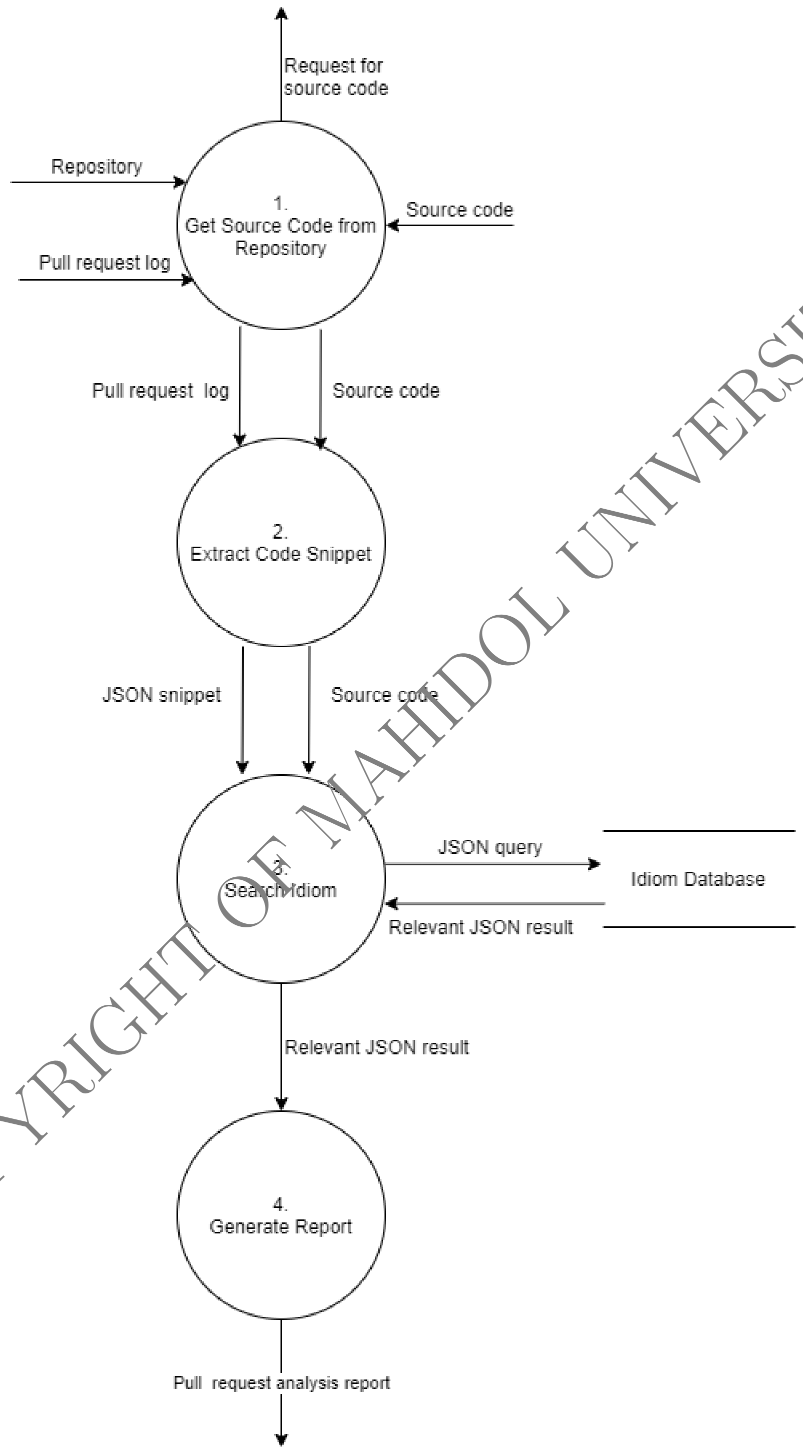


Figure 3.8: Data flow diagram level 1 of Teddy system in prevention mode

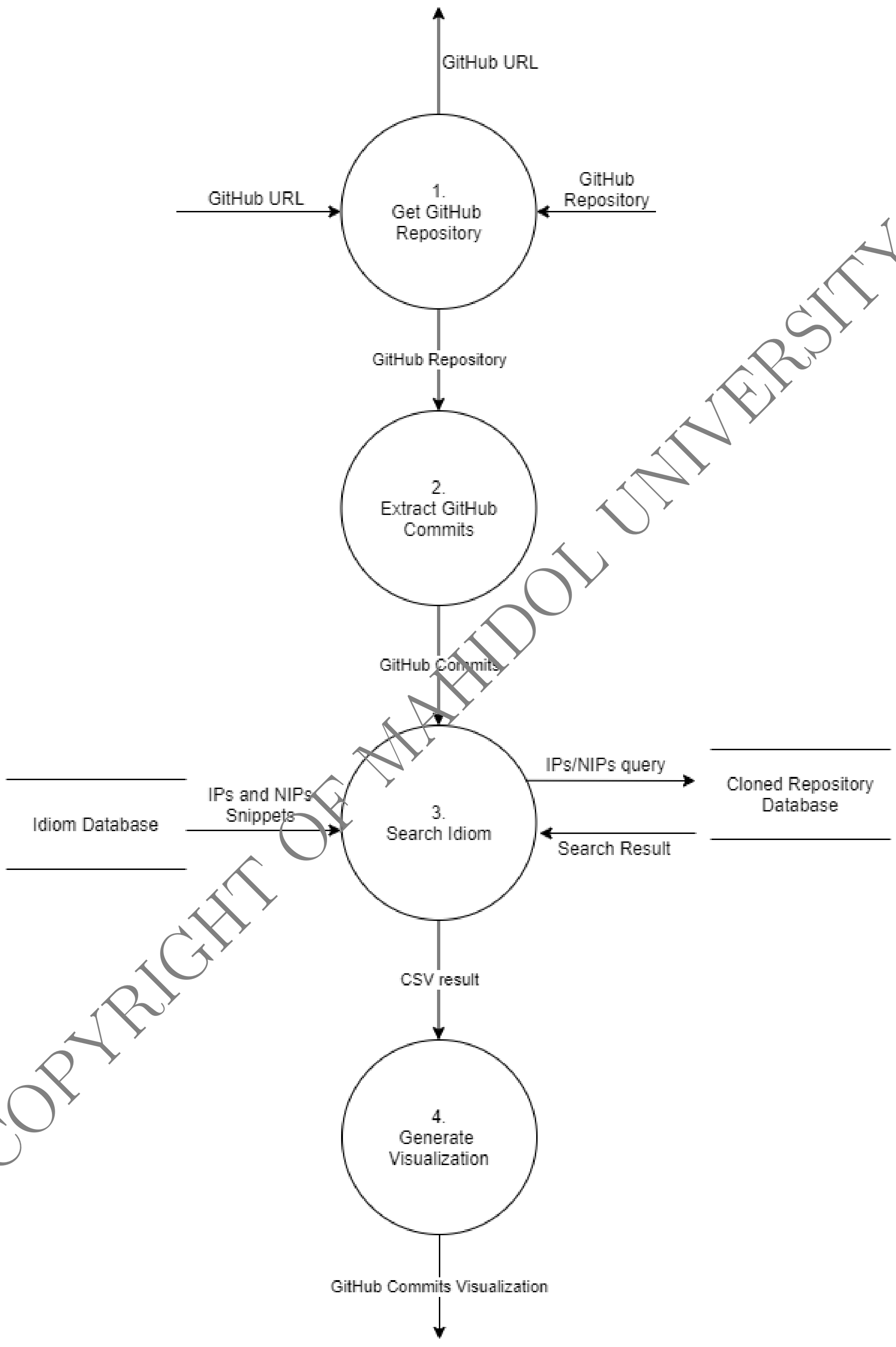
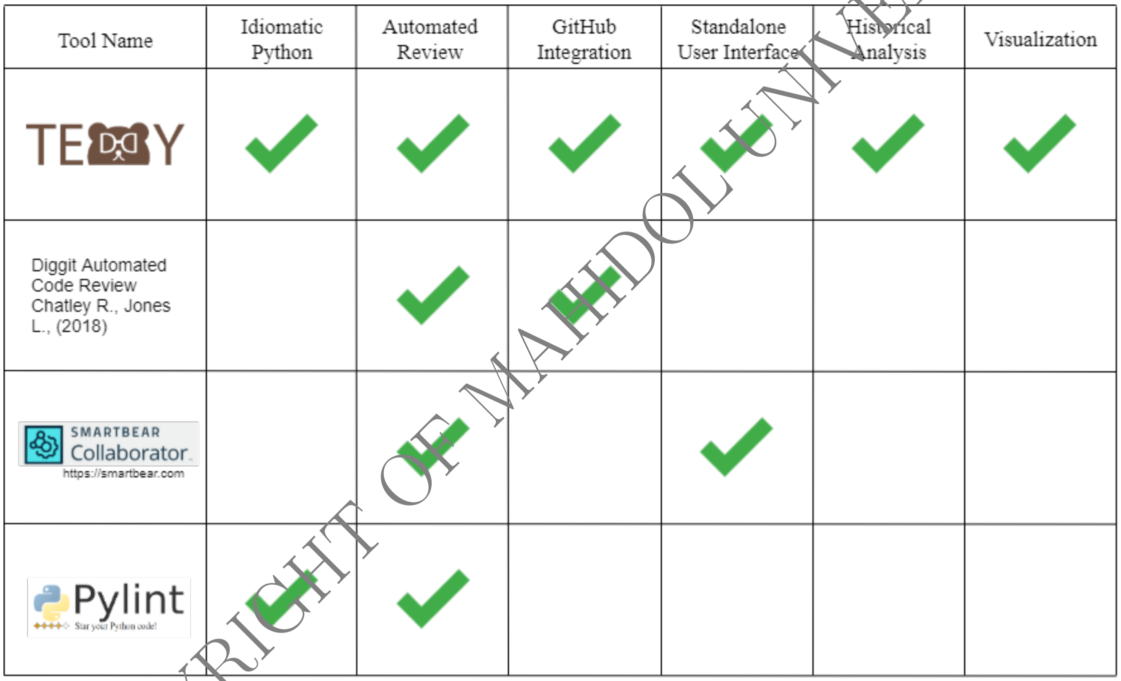


Figure 3.9: Data flow diagram level 1 of Teddy system in detection mode

3.4 Comparison to Relevant Tools

We have compared Teddy against three other tools that achieve similar objectives. Our criteria that is used to compare and show distinct features including IP, automated review, GitHub integration, standalone user interface, historical analysis and a visualization. The criteria were chosen based on the importance in regards to the objective of this project and the relevancy to the features that our tool can provide.

From Figure 3.10, we can see that Teddy has all the general features and unique features mentioned above, while other tools only have there some of the features but not all.



Tool Name	Idiomatic Python	Automated Review	GitHub Integration	Standalone User Interface	Historical Analysis	Visualization
TEDDY	✓	✓	✓	✓	✓	✓
Diggit Automated Code Review Chatley R., Jones L., (2018)		✓	✓			
SMARTBEAR Collaborator https://smartbear.com		✓		✓		
Pylint Star your Python code!	✓	✓				

Figure 3.10: Features comparison between Teddy and other relevant tools

CHAPTER 4
IMPLEMENTATION

This chapter contains the implementation details of this project which are divided into four sections: IPs and NIPs preparation, the prevention mode, the detection mode, and the user interface.

4.1 IPs and NIPs Preparation

From the literature review, we have decided to collect each type of idioms from the references that we have used [3][11]. Then, we extracted as many distinct coding patterns of both IPs ad NIPs as we could from examples that each source contains. 10 different types of IP and NIP, as shown in Table 4.1, were selected for the collection. After completing the collection, the code patterns were organized into different separate Python files. These are to be used later in pattern matching procedure with Siamese.

Table 4.1: Types of IPs and NIPs studied in this project

Type	Name	Description	Amount
IP	dictionary comprehension [3]	Declaration of `dict` variable and assigning its elements in a single statement	7
IP	enumerate	for-loop iteration using `enumerate` function	8
IP	file reading statement	Using `with open() as ...` to open a file	5
IP	list comprehension	Declaration of `list` data type and assignment of elements in a single statement	7
IP	if statement	Using implicit truthfulness for `if` condition statement	11
IP	string formatting	Concatenation of multiple string formatting statements, use of `.format()` with placeholder(s) in a static string	3
IP	set	Using `set` data type to create a unique collection	4
IP	tuple	Unpacking data for multiple assignment at once	4
IP	variable swapping	Using tuple to swap values between two or more variables	4
IP	code formatting	Proper use of indentation for code blocks and writing one statement per one line	2
NIP	dictionary comprehension	Separate declaration and for-loop element assignment of a `dict` variable	6
NIP	enumerate	for-loop iteration without `enumerate`	6
NIP	file reading statement	File opening without using `with open() as ...`	5
NIP	list comprehension	Separate declaration and for-loop element assignment of a `list` variable	8
NIP	if statement	direct comparison of variable with `True`, `False`, or `None`	12
NIP	string formatting	Sequence of one string formatting commands per one line, using '+' to concatenate static string and variable(s) together, or using '%' as string variable placeholder	7
NIP	set	Using for-loop to create a unique collection of item	4
NIP	tuple	Explicitly assigning variables with elements in a collection	4
NIP	variable swapping	Using a temporary variable to swap two variables' values	4
NIP	code formatting	Using ';' to put more than one statement in a single line	2

4.2 Siamese Configurations

Siamese allows for flexible customization of its search configuration with many different parameters in the setting options. From the array of over 40 individual parameters, the following are key parameters that are major factors to the result of this study.

From Table 4.2, there are three parameters included. They are originally designed to work with code clone detection, however they have been applied to perform NIP/IP detection in this particular scenario. Value assigned to each of them has significant effect to the outcome of IPs/NIPs search results. Details regarding how the parameters' values are set for Siamese to perform at the best possible degree is explained in Chapter 5.

Table 4.2: Key parameters of Siamese configuration

Name	Description	Value Options
Clone similarity computation method	Used to select the method of computing the numerical similarity value between a query method and an index method	none, fuzzywuzzy, tokenratio
Multi-representation similarity threshold	The percentage value that represent the threshold of the computed clone similarity for four code representations (representation 0, 1, 2, and 3)	0% - 100%
Multi-representation n-gram size	The length of consecutive tokens that form into a tuples to generate multi-representation formats (representation 0, 1, 2, and 3) of the original code	At least 1

4.3 Prevention Mode

Prevention mode, one of the two main features of this project, allows users to be able to get instant feedback of IPs and NIPs found in their pull requests from an automated process. The process involves the usage of GitHub's API and Bots to help show the results to the users.

4.3.1 Techniques and Tools Involved in the Implementation

1. GitHub API - An API provided by GitHub that is used to communicate and handle activities within GitHub, these APIs can be used to read the activities in a repository and make changes within the GitHub repository.

2. Node.js - Used as a server handler for Teddy, Node.js handles receiving activities from GitHub and send it to Siamese. It also handles the usage of GitHub's API.

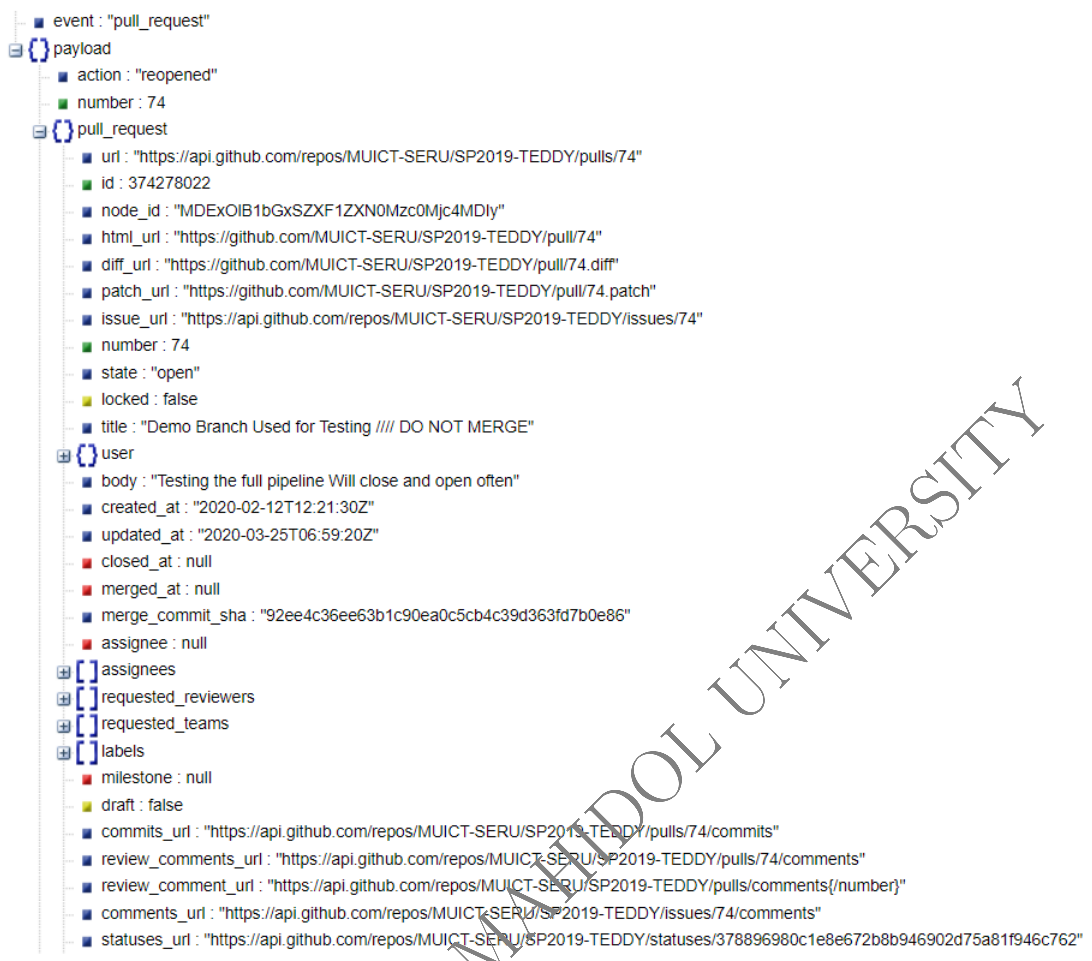
3. Siamese - Handles the search and analysis of IPs and NIPs. It sends suggestions back to Node.js to be passed onto GitHub.

4.3.2 Implementation Detail

Prevention mode can be split into four major parts which have specific functions needed to make the whole pipeline operational. First, whenever activities happen on GitHub, a system has to be designed to intercept those activities and send it somewhere for analysis. The second part is designing a channel for the activities to be accepted into our system and parse only certain parts of the activities which include the commit ID and the edits content. Next, the edits content would be sent to Siamese in which Siamese would use it to scan for IP and NIP code fragments, and retrieve recommendation data for each NIP fragment that is found. Lastly, the system would send back a response containing the components which are the file name and edit suggestions for non-idiomatic Python back to GitHub pull request.

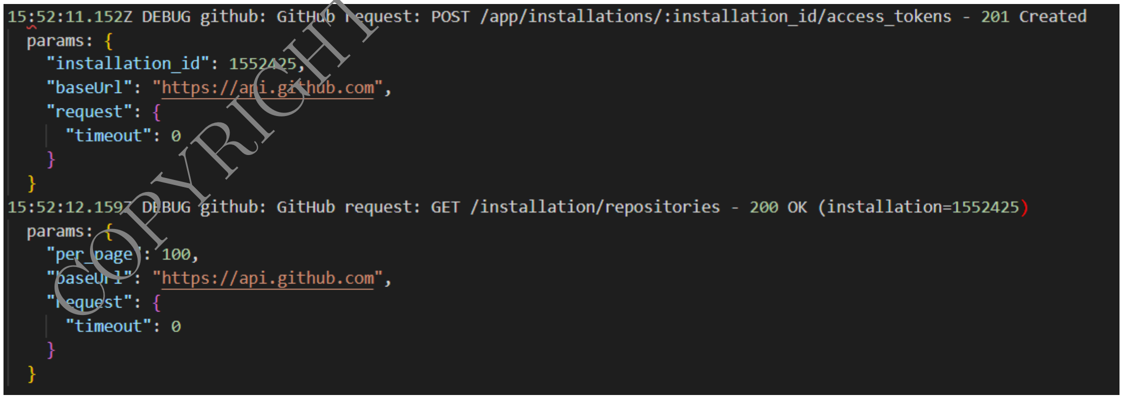
Communication Between GitHub and Smee.io

In the first phase, our goal is to obtain information from GitHub, in which whenever an activity happens on GitHub, there would be a new auto-generated log to keep track on what has occurred. GitHub allows the integration of Bots and Webhooks to interact with this information. Smee.io has been chosen as the candidate to be the Webhook between our Probot and GitHub. Thus, Probot and Smee.io were used to receive important data from GitHub, such as the repository, user ID, and pull request ID. Next, the data is sent to Teddy as a JSON file with GitHub's generic structure.



```
event : "pull_request"
{} payload
    action : "reopened"
    number : 74
    {} pull_request
        url : "https://api.github.com/repos/MUICT-SERU/SP2019-TEDDY/pulls/74"
        id : 374278022
        node_id : "MDExOlB1bGxSZXF1ZXN0Mzc0Mjc4MDIy"
        html_url : "https://github.com/MUICT-SERU/SP2019-TEDDY/pull/74"
        diff_url : "https://github.com/MUICT-SERU/SP2019-TEDDY/pull/74.diff"
        patch_url : "https://github.com/MUICT-SERU/SP2019-TEDDY/pull/74.patch"
        issue_url : "https://api.github.com/repos/MUICT-SERU/SP2019-TEDDY/issues/74"
        number : 74
        state : "open"
        locked : false
        title : "Demo Branch Used for Testing //// DO NOT MERGE"
        {} user
        body : "Testing the full pipeline Will close and open often"
        created_at : "2020-02-12T12:21:30Z"
        updated_at : "2020-03-25T06:59:20Z"
        closed_at : null
        merged_at : null
        merge_commit_sha : "92ee4c36ee63b1c90ea0c5cb4c39d363fd7b0e86"
        assignee : null
        [ ] assignees
        [ ] requested_reviewers
        [ ] requested_teams
        [ ] labels
        milestone : null
        draft : false
        commits_url : "https://api.github.com/repos/MUICT-SERU/SP2019-TEDDY/pulls/74/commits"
        review_comments_url : "https://api.github.com/repos/MUICT-SERU/SP2019-TEDDY/pulls/74/comments"
        review_comment_url : "https://api.github.com/repos/MUICT-SERU/SP2019-TEDDY/pulls/comments{/number}"
        comments_url : "https://api.github.com/repos/MUICT-SERU/SP2019-TEDDY/issues/74/comments"
        statuses_url : "https://api.github.com/repos/MUICT-SERU/SP2019-TEDDY/statuses/378896980c1e8e672b8b946902d75a81f946c762"
```

Figure 4.1: JSON structure used by GitHub

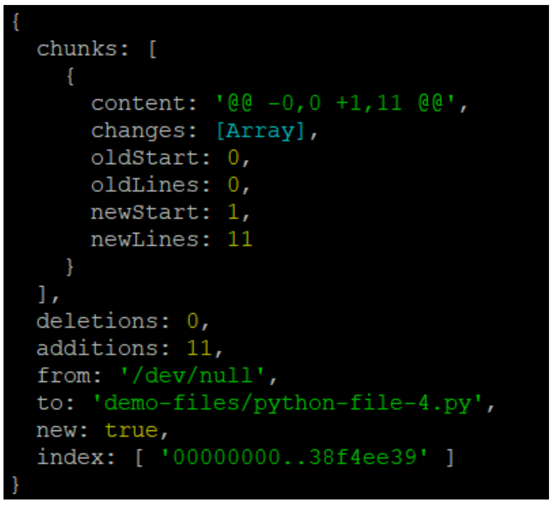


```
15:52:11.152Z DEBUG github: GitHub request: POST /app/installations/:installation_id/access_tokens - 201 Created
  params: {
    "installation_id": 1552425,
    "baseUrl": "https://api.github.com",
    "request": {
      "timeout": 0
    }
  }
15:52:12.159Z DEBUG github: GitHub request: GET /installation/repositories - 200 OK (installation=1552425)
  params: {
    "per_page": 100,
    "baseUrl": "https://api.github.com",
    "request": {
      "timeout": 0
    }
  }
```

Figure 4.3: API response used by GitHub

Extracting Data from Smee.io

Once the JSON package has been received from Smee.io, Probot would comb through the JSON file and obtain necessary information, such as the commit ID, names



```
{
  chunks: [
    {
      content: '@@ -0,0 +1,11 @@',
      changes: [Array],
      oldStart: 0,
      oldLines: 0,
      newStart: 1,
      newLines: 11
    }
  ],
  deletions: 0,
  additions: 11,
  from: '/dev/null',
  to: 'demo-files/python-file-4.py',
  new: true,
  index: [ '00000000..38f4ee39' ]
}
```

Figure 4.2: JSON structure used by Teddy

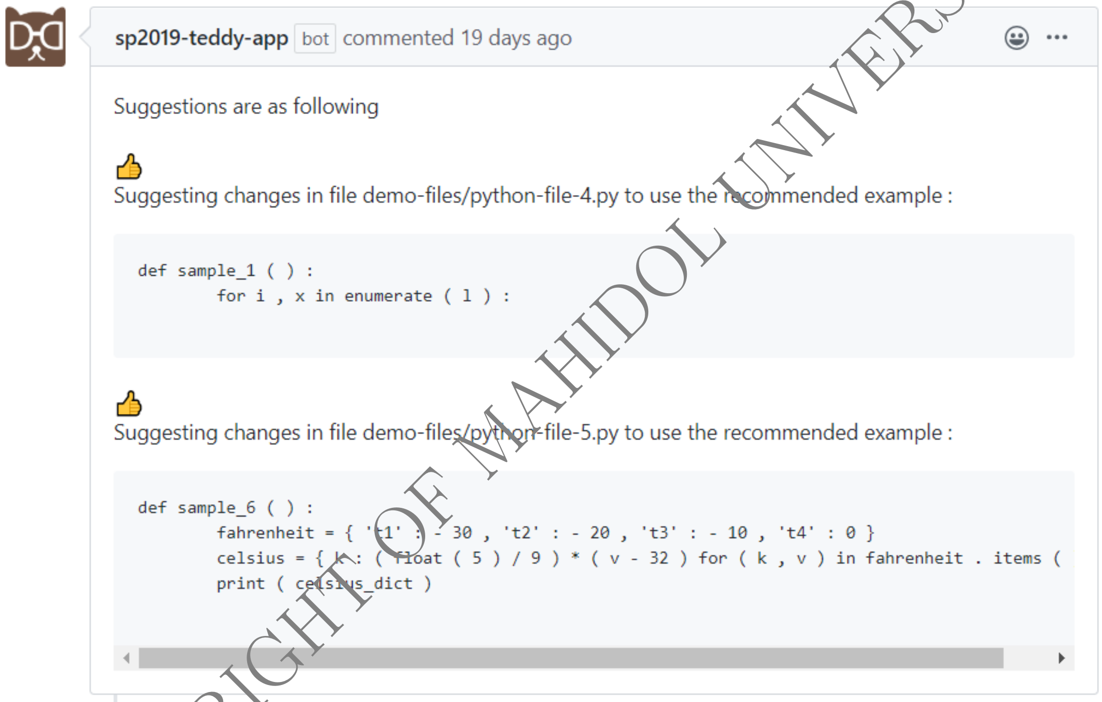
of the files that were edited and editing contents. The information would then be parsed to match with our JSON structure as shown in Figure 4.2. The JSON would be sent off to Siamese for further processing.

Siamese and Idiom Matching

A Spring Boot framework was integrated into Siamese which was originally a command line tool without any module to support HTTP requests. By having data model Java classes that replicate the structure of GitHub JSON object (Figure 4.1), SiameseX, as a Spring Boot Application, can handle an incoming HTTP POST request containing the commit content in JSON format. The structure of the received JSON and sample values of each field can be seen in Figure 4.2. Then, the values from `edit` attributes, where the actual code changes of the pull request's commits are located, are extracted and put together as a query that is used to match with IPs and NIPs within the search index, i.e., the idiom database. Inside the index contains 113 code snippets of different IP and NIP types, including the recommendation correcting pattern for each NIP type. If there is a match between an NIP from the query and an NIP sample in the index, the correcting pattern for that NIP type is retrieved from the index and included as part of response's payload.

Response and Commenting on a Pull Request

SiameseX sends back a response containing the original pull request information that it received. The structure is similar to that of JSON query package but with the added NIP correction recommendation patterns - an additional attribute appended to the "chunk" structure inside the JSON object. Finally, SiameseX sends the serialized form of the JSON response object back to Probot. During this, Probot creates a direct connection to the GitHub API, then makes a comment and a pass-check directly to the pull request as in Figure 4.4.


sp2019-teddy-app bot commented 19 days ago

Suggestions are as following

Suggesting changes in file demo-files/python-file-4.py to use the recommended example :

```
def sample_1 ( ) :
        for i , x in enumerate ( l ) :
```

Suggesting changes in file demo-files/python-file-5.py to use the recommended example :

```
def sample_6 ( ) :
        fahrenheit = { 't1' : - 30 , 't2' : - 20 , 't3' : - 10 , 't4' : 0 }
        celsius = { k : ( float ( 5 ) / 9 ) * ( v - 32 ) for ( k , v ) in fahrenheit . items (
        print ( celsius_dict )
```

Figure 4.4: Teddy's automated comment in a GitHub pull request

4.3.3 Requirements

As many tools that have been built, there are requirements that need to be met for a tool to operate and function properly. Here are the requirements needed for this Teddy tool to function.

1. A private/public GitHub repository
2. The GitHub user called "TeddyMuict" must be added as a collaborator in the repository

3. The GitHub Bot "Teddy" must be installed into the project.

4. The Bot "Teddy" must have the permission to comment on pull request.

4.4 Detection Mode

The detection mode works by providing a visualization of usage of IPs and NIPs across historical versions of source code from a GitHub repository. This is for the code developers to gain quick perception of how the software project has developed in terms of usage of IPs and NIPs, and make changes to their codes accordingly to improve code readability and performance.

4.4.1 Techniques and Tools Involved in the Implementation

Implementation of detection modes is done by incorporating a set of tools and techniques which are available, and adjust them to accommodate the application.

1. GNU Bash [23] - Detection mode consists of several self-sustained modules and components with little to no built-in interactive functions to relay the data with other modules. Therefore, bash scripts are used in order to manage the sequence of modules' execution and direct the flow of data that occurs within the detection mode.

2. `git` vcs [24] - `git` version control system is needed as detection mode must handles and manage historical versions of the code. Therefore, `git` command lines are used for cloning and organizing the versions of the cloned repository.

3. Siamese [1] - Siamese is implemented as a search engine that looks for IPs and NIPs within the cloned software repository, and report the search results in the form of `csv` files. The query is a set of pre-selected IP and NIP sample code snippets.

4. Bokeh [25] - Bokeh, an interactive visualization library for Python, is used to produce the final visualization of IPs and NIPs in detection mode. The format is in the form of a `html` file.

4.4.2 Implementation Detail

As mentioned earlier, GNU bash commands are necessary to connect the individual modules that makes up detection mode together, and direct the flow of data in proper manner. Thus all of the commands to execute each module are written in bash scripts.

First, a `git` command is executed to clone a selected repository (the URL must be provided as an argument when executing the script). The specified repository is then cloned and located at where the script is. Then, the script utilizes `git checkout` to revert the version of the repository to its first commit version as the iteration begins.

For each commit version of the repository, the script triggers the execution of `siamese.jar` which handles the indexing of the source code (of that commit version) into the running Elasticsearch, and query that index with a prepared set of IPs and NIPs code snippets. Once Siamese completes the querying process, the search results is written onto a `.csv` file.

After Siamese completes it process for one iteration of a commit version, the script instructs the version of the repository to be shifted to the next version. This step is repeated until Siamese finishes searching for IPs/NIPs in the latest commit version of the cloned repository.

With one search result file generated for one commit version, the visualization module then takes all of the `.csv` files as the input to process the content sides for the visualization. A bash script is also used to run a Python script that is responsible for generating the visualization graph. Finally, an interactive visualization plot of IPs and NIPs is produced and put in a local `.html` file.

4.4.3 IP and NIP Visualization

The visualization is done by contributing the tool called Python Bokeh (ver.2.0.1) in the form of a scatter plot to represent the occurrences of both IPs and NIPs in each file in every commit in a Python project from GitHub.

The red legends are markers for NIPs while the green ones are for IPs. The different marker symbols are used for identifying each type of idiom that have been covered in this project. Details of the mapping between the marker symbols and the

Table 4.3: Table showing mapping between the marker and type of IPs/NIPs

IPs/NIPs	Marker symbol
Dictionary Comprehension	○ (Circle)
Enumerate	△ (Triangle)
File Reading Statement	□ (Square)
List Comprehension	◇ (Diamond)
If Statement	⬡ (Hex)
String Formatting	∗ (Asterisk)
Code Formatting	⊗ (Circle Cross)
Set	× (Cross)
Tuple	▽ (Inverted Triangle)
Variable Swapping	⊠ (Square Cross)

idiom types are in Table 4.3.

Figure 4.5 illustrates an overview of the visualization of Teddy's detection mode. The x-axis represents the commit ID number of the software project while the y-axis represents the file names of each file contained in the project. The green legends of the scatter plot illustrate the IP and the red legends are for NIP.

The zoomed version of the visualization is shown in Figure 4.6. From the figure, it can be seen that the scatter plot has different markers which represents each type of idioms. In our case study, we have the idiom in total for 10 types including: dictionary comprehension, enumerate, file reading statement, list comprehension, if statement, string formatting, code formatting, set, tuple, and variable swapping.

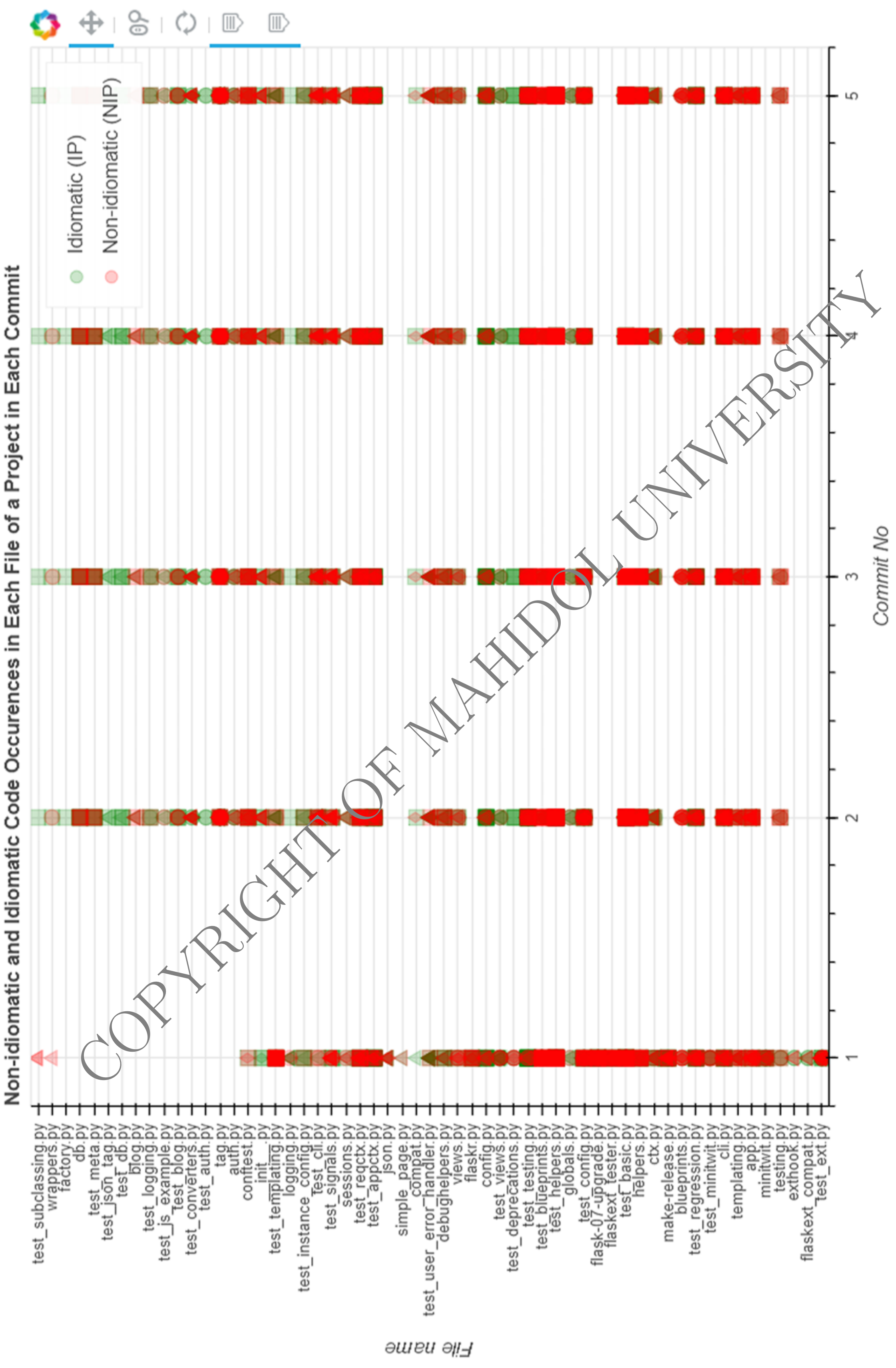
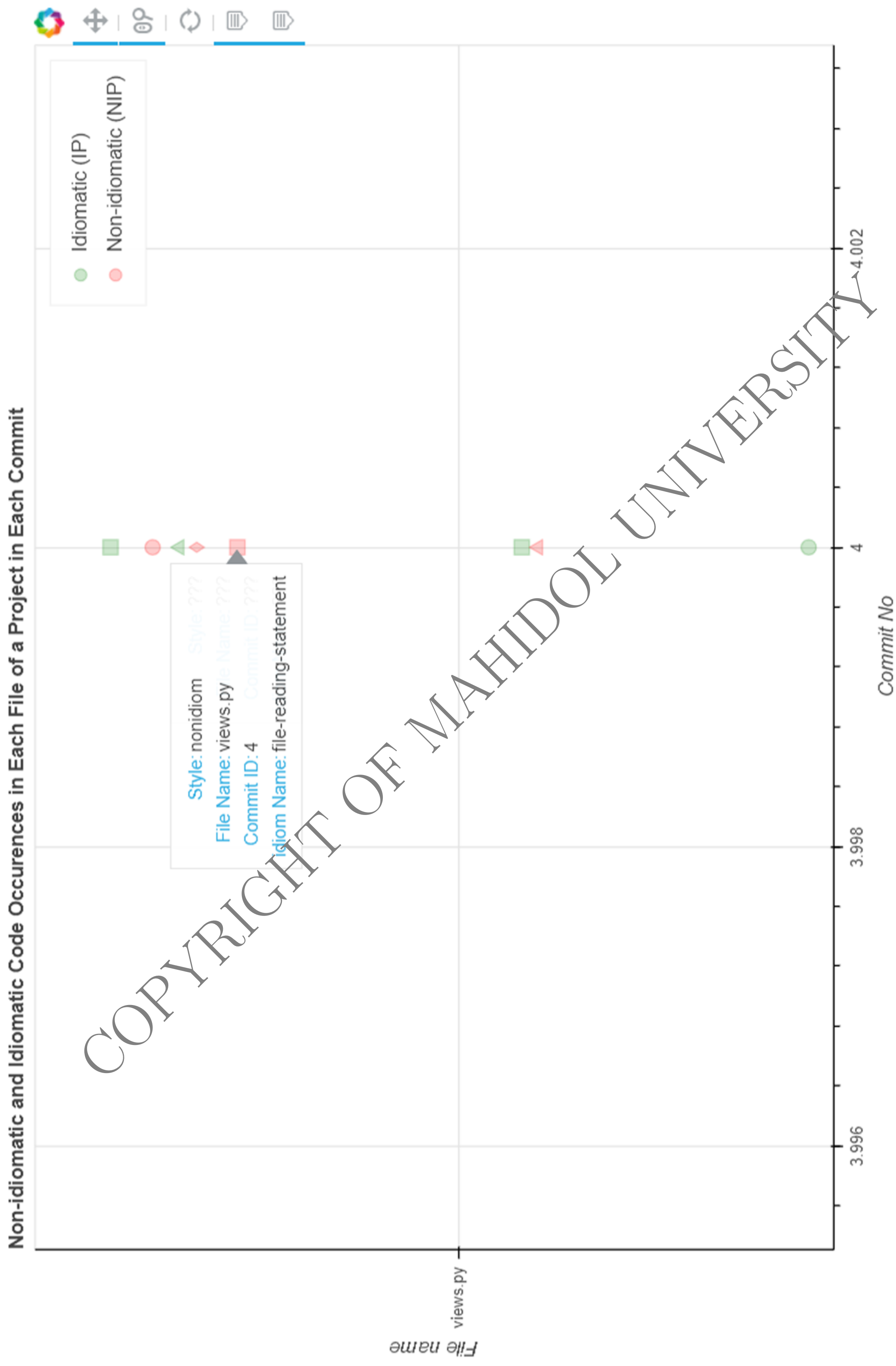


Figure 4.5: Overview of the sample visualization

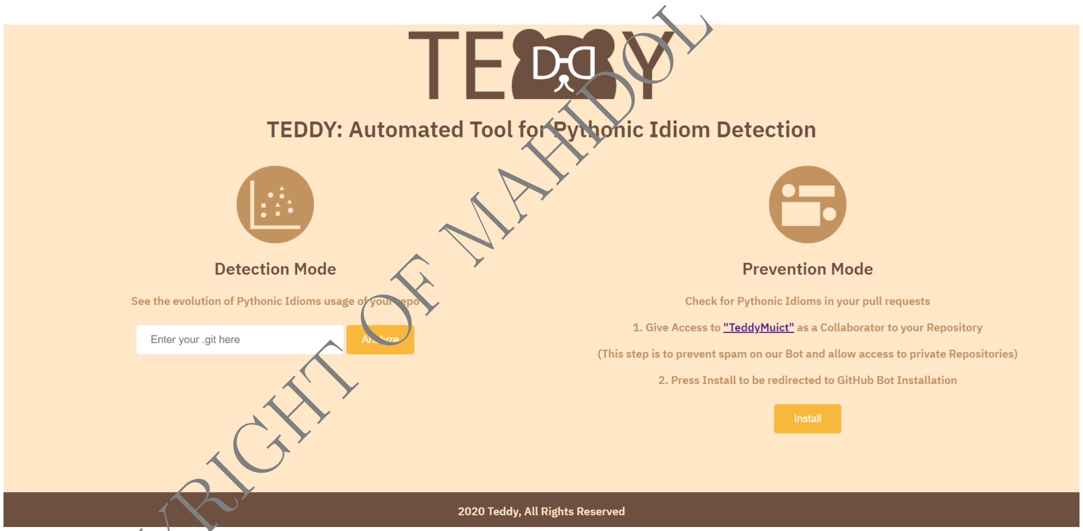

Non-idiomatic and Idiomatic Code Occurences in Each File of a Project in Each Commit
Idiomatic (IP)
Non-idiomatic (NIP)
File name
views.py
Commit No
3.996
3.998
4
4.002

Style: nonidiom
File Name: views.py
Commit ID: 4
Idiom Name: file-reading-statement

Figure 4.6: The zoomed view of the sample visualization

4.5 User Interface

Figure 4.7 shows the user interface which has been designed for minimal interaction but high usability. Users do not have to navigate through settings as most of the process in both modes are automated. The user interface is divided into two sections, detection mode (left) and prevention mode (right). In the detection mode, users have to put in a GitHub repository URL link in the provided box, and press "Analyze" button. This will triggers the entire detection mode pipeline automatically as explained in Section 4.4. Once the process is done, a new page would pop up showing the visualization. As for the prevention mode, users have to follow the instructions, which require them to allow "TeddyMuict" to access into their repository as a collaborator and install "TeddyBot". The prevention mode mechanism will be automatically run whenever a new pull request is created as explained in Section 4.3.


TEDDY: Automated Tool for Pythonic Idiom Detection

Detection Mode

See the evolution of Pythonic Idioms usage of your repo

Prevention Mode

Check for Pythonic Idioms in your pull requests

1. Give Access to "TeddyMuict" as a Collaborator to your Repository

(This step is to prevent spam on our Bot and allow access to private Repositories)

2. Press Install to be redirected to GitHub Bot Installation

2020 Teddy, All Rights Reserved

Figure 4.7: Web interface for Teddy tool

CHAPTER 5
EVALUATION AND DISCUSSION

This chapter focuses on the testing of Teddy tool, including the detection mode on synthetic data set to evaluate its detection precision and recall, and the detection mode on 3 software projects.

5.1 Detection Mode: Idiom Detection Accuracy

5.1.1 Methodology

Before Teddy could be run on a real software project, an evaluation experiment was conducted to verify its performance and make sure it can correctly return results as desired, or else some minor adjustments have to be made. A set of syntactic unbiased data set was created, with error measures used to carry out the assessment.

Evaluation Data Set

To make a ground-truth data set for evaluation of Teddy, a collection of Python code has been prepared accordingly. In order to assure that there is no bias in the testing set, the prepared code has been from third-party sources - mainly the GitHub repository of Flask, Tensorflow and Manim.

Within the data set consists of three groups of Python code files, as shown in Table 5.1. The first group - the normal code - is the group of Python codes that do not contain parts relevant to being classified as neither NIP nor IP. There are a total of 30 files for this group. The second group - the IP group - represents 20 snippets of IP code (one snippet per one file), each of which is sampled from the three open-source Python projects, as well as other online sources. And the third group - the NIP group - consists of 20 NIP code snippets (one snippet per one file) that are also picked from the similar sources as the IP group.

To be able to systematically extract IP and NIP snippets from hundreds of source code files from the three mentioned software projects, representative regular expressions are created for each of the IP and NIP type. Using a tool called CCGrep [26], those expressions are used to scan for IP/NIP patterns in the actual files. The given results, however, has to undergo manual filtering once before being added to the data set.

Among 20 IP and NIP code files, 2 samples for each of IP and NIP types are included. In total, with the normal codes, there are 70 Python files in this ground-truth testing data set.

Table 5.1: Table summarizing the contents inside evaluation data set

File group	Description	Number of files
Normal code	Python codes without any IP or NIP statements	30
IP code	Python codes with IP statements	20
NIP code	Python codes with NIP statements	20
Total		**70**

Experimental Framework

As detection mode aims to find and label different IP and NIP code snippets within historical versions of a software repository, the experiment has been designed to simulate one iteration of such scenario. A set of 113 IP and NIP code snippets, 55 IP snippets and 58 NIP snippets, has been prepared and used to query for IP and NIP codes inside the ground-truth synthetic data set.

The focus is to optimize and adjust the search parameters of Siamese search engine so that it can accurately find and match the IP and NIP codes in the query with those in the data set. The following error measures are used as assessing benchmarks for one particular configuration variant.

Error Measures

- Mean Average Precision (MAP) - The mean average precision is the mean value of average precision values over all the queries. The average precision is computed from different recall level, i.e., each time a relevant document is found. It is defined as

$$MAP(Q) = \frac{1}{|Q|}\sum_{j=1}^{|Q|}\frac{1}{|m_j|}\sum_{k=1}^{|m_j|} \mathit{Precision}(R_{jk}) \tag{5.1}$$

where Q is the set of the queries $\{q_1, q_2, ..., q_{|Q|}\}$, m_j is the set of relevant results for a query q_j, R_{jk} is the set of ranked retrieved items from the first-ranked item until a relevant document d_k, and *Precision*(X) is the function to compute normal precision for X.

- Query Recall (QR) - Query recall is the measure to evaluate how complete is the number of relevant items retrieved in respect to the subset of queries whose results are not empty (called "returned queries"). QR for a set of returned queries r is defined as:

$$QR = \frac{1}{|r|}\sum_{i=1}^{|r|}\frac{|RRI_i|}{|TRI_i|} \tag{5.2}$$

where RRI_i is the set of retrieved relevant items and TRI_i the set of all relevant items for the i-th returned query, respectively.

For this particular evaluation, the total number of relevant items (TRI_i) inside the data set is two for every IP/NIP query. Therefore, the possible values of recall for a single query can either be 0 ($|RRI_i| = 0$), 0.5 ($|RRI_i| = 1$), or 1 ($|RRI_i| = 2$). After summing all the recalls of every returned query, the final QR is then computed by averaging the summed amount by the number of returned queries $|r|$.

- Overall Recall (OR) - Overall recall is the measure to evaluate how complete is the number of relevant items retrieved in respect to the entire set of query R:

$$OR = \frac{1}{|R|}\sum_{i=1}^{|R|}\frac{|RRI_i|}{|TRI_i|} \tag{5.3}$$

where RRI_i is the set of retrieved relevant items and TRI_i the set of all relevant items for the i-th query, respectively.

In contrast to the error measure QR, OR also takes into account the queries which are not returned any result, in other words having empty list of results. For example, provided that there are a total of 113 individual IP and NIP snippet queries, if 88 of them are returned with non-empty list of results, the error measure QR is computed using the set of 88 returned queries while OR also considers the remaining 25 queries which have empty results.

- Mean Reciprocal Rank (MRR) - The mean reciprocal rank is the average of the reciprocal ranks of results for a sample of queries Q:

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i} \tag{5.4}$$

where $rank_i$ refers to the rank position of the first relevant document for the i-th query. In our implementation, the reciprocal rank of a query (equivalent to a row in the `csv` output file) is the multiplicative inverse of the rank of the first true-positive match between the IP/NIP of the query and IP/NIP of the data set. The reciprocal rank of each individual row is added together and averaged by the number of rows in that file.

- Overall Recall (OR) - Overall recall is the measure to evaluate how complete is the number of relevant items retrieved in respect to the entire set of query R:

$$OR = \frac{1}{|R|}\sum_{i=1}^{|R|}\frac{|RRI_i|}{|TRI_i|} \tag{5.5}$$

where RRI_i is the set of retrieved relevant items and TRI_i the set of all relevant items for the i-th query, respectively.

In contrast to the error measure QR, OR also takes into account the queries which are not returned any result, in other words having empty list of results. For example, provided that there are a total of 113 individual IP and NIP snippet queries, if 88 of them are returned with non-empty list of results, the error measure QR is computed using the set of 88 returned queries while OR also considers the remaining 25 queries which have empty results.

- Mean Reciprocal Rank (MRR) - The mean reciprocal rank is the average of the

reciprocal ranks of results for a sample of queries Q:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i} \tag{5.6}$$

where $rank_i$ refers to the rank position of the first relevant document for the i-th query. In our implementation, the reciprocal rank of a query (equivalent to a row in the `csv` output file) is the multiplicative inverse of the rank of the first true-positive match between the IP/NIP of the query and IP/NIP of the data set. The reciprocal rank of each individual row is added together and averaged by the number of rows in that file.

5.1.2 Results and Discussion

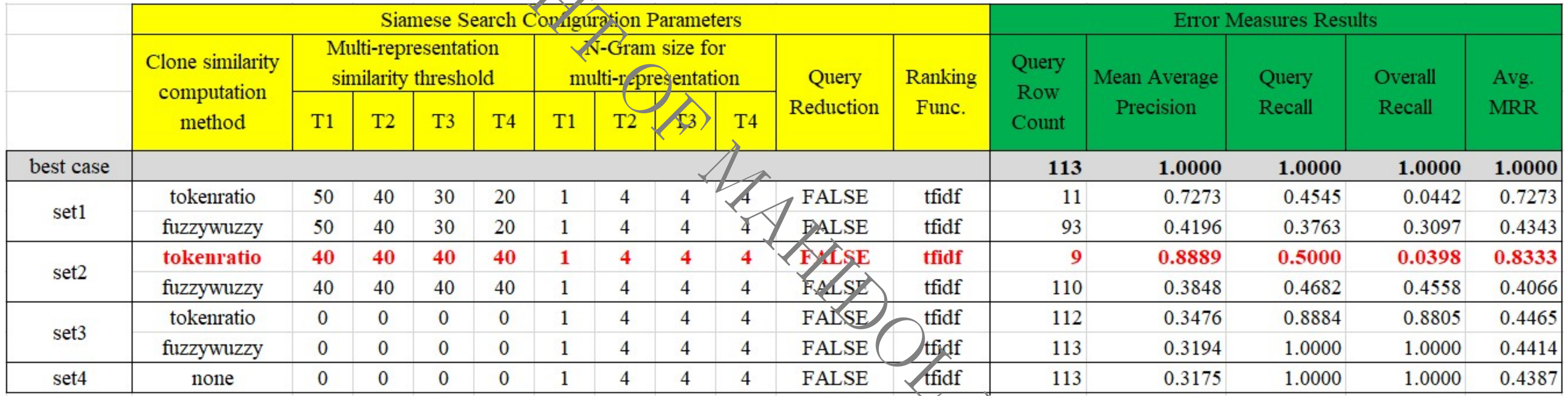
The results from running the experiment underwent thorough manual analysis and evaluation of the authors, using the error measures previously mentioned. With a vast array of adjustable parameters in Siamese's configuration, the margin of variance for error measures is from as low as 0.04 to the best case's of 1.

After several trial-and-errors with different settings over 40 variations, it has been observed and concluded that the following group of tests, in Figure 5.1, has the best overall results across the four error measures used.

From the table, only the similarity computation method and multi-representation similarity thresholds are the independent variables of interest while the other remaining settings are fixed as controlled variables. For each of 3 different multi-representation similarity threshold permutations - set1 (50-40-30-20), set2 (40-40-40-40), and set3 (0-0-0-0) - two sub-variations between clone similarity computation method of tokenratio and fuzzywuzzy were tested.

It can be observed that the two different clone similarity computation methods contribute to different aspects of the search quality. By taking a close look at QR and OR measures, the tests which applied fuzzywuzzy for clone similarity computation threshold have higher values comparing to their tokenratio counterpart in the same experiment set. They also have larger set of returned queries as well. In exchange for lower recall, the tests with tokenratio setting gain much higher MAP and MRR values comparing to those of fuzzywuzzy.

An unexpected finding was also made from set3 experiments. It is clear that despite all of multi-representation similarity threshold being set to zero (0-0-0-0), there are still some "unreturned" queries (query with empty result) with tokenratio as similarity threshold computing method, thus making the recall less than 1.



	Siamese Search Configuration Parameters											Error Measures Results				
	Clone similarity computation method	Multi-representation similarity threshold				N-Gram size for multi-representation				Query Reduction	Ranking Func.	Query Row Count	Mean Average Precision	Query Recall	Overall Recall	Avg. MRR
		T1	T2	T3	T4	T1	T2	T3	T4							
best case												**113**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
set1	tokenratio	50	40	30	20	1	4	4	4	FALSE	tfidf	11	0.7273	0.4545	0.0442	0.7273
	fuzzywuzzy	50	40	30	20	1	4	4	4	FALSE	tfidf	93	0.4196	0.3763	0.3097	0.4343
set2	**tokenratio**	**40**	**40**	**40**	**40**	**1**	**4**	**4**	**4**	**FALSE**	**tfidf**	**9**	**0.8889**	**0.5000**	**0.0398**	**0.8333**
	fuzzywuzzy	40	40	40	40	1	4	4	4	FALSE	tfidf	110	0.3848	0.4682	0.4558	0.4066
set3	tokenratio	0	0	0	0	1	4	4	4	FALSE	tfidf	112	0.3476	0.8884	0.8805	0.4465
	fuzzywuzzy	0	0	0	0	1	4	4	4	FALSE	tfidf	113	0.3194	1.0000	1.0000	0.4414
set4	none	0	0	0	0	1	4	4	4	FALSE	tfidf	113	0.3175	1.0000	1.0000	0.4387

Figure 5.1: Table of Siamese's parameter tuning experiment and the resulting error measures

5.2 Detection Mode: Test on Real Software Projects

After an optimal setting for NIP and IP searching was discovered from the experiment, three actual software GitHub repositories were run with Teddy's detection mode to inspect and verify the functionality.

The first selected repository was Flask, a lightweight Python WSGI web application framework. The repository's master branch was cloned and the tool iterated through its 3,887 commits (as of April 16, 2020) from first to last. Figure 5.2 shows the final visualization output of NIPs and IPs usage found in different version of the project. From the figure, we can see that most of the files in the project use only IPs or NIPs in their code without changing to it's counterpart all along the commits while few files use both styles. However, there is some file, for example `app.py`, that improve their style of writing at later commits which can be seen that the scatter plot starts to turn green in the later commits while on the other hand, `flask-07-upgrade.py` has trend of changing the code from IPs into NIPs.

Following Flask is the ipyparallel project. This is an ipython's GitHub repository for interactive parallel computing in Python with 1,792 commits in total. From Figure 5.3, the visualization illustrates that both IPs and NIPs spreading all over the project which can be seen as thin green liner at the edge of each plot. The plot can tell that the project does not seem to present any changes of coding style.

The third repository chosen was Tensorflow's tfx, an end-to-end platform for deploying production machine learning pipelines, which has 1,260 commits since its first initialization. From the visualization shown in figure 5.4, it can be seen that most of the files in the project contain IPs more than NIPs since the there are much more green plots. In additional, the plot also shows the trend of changing from NIPs to IPs from the changes of green legend into red one, for example, in file `dependency_util.py`.

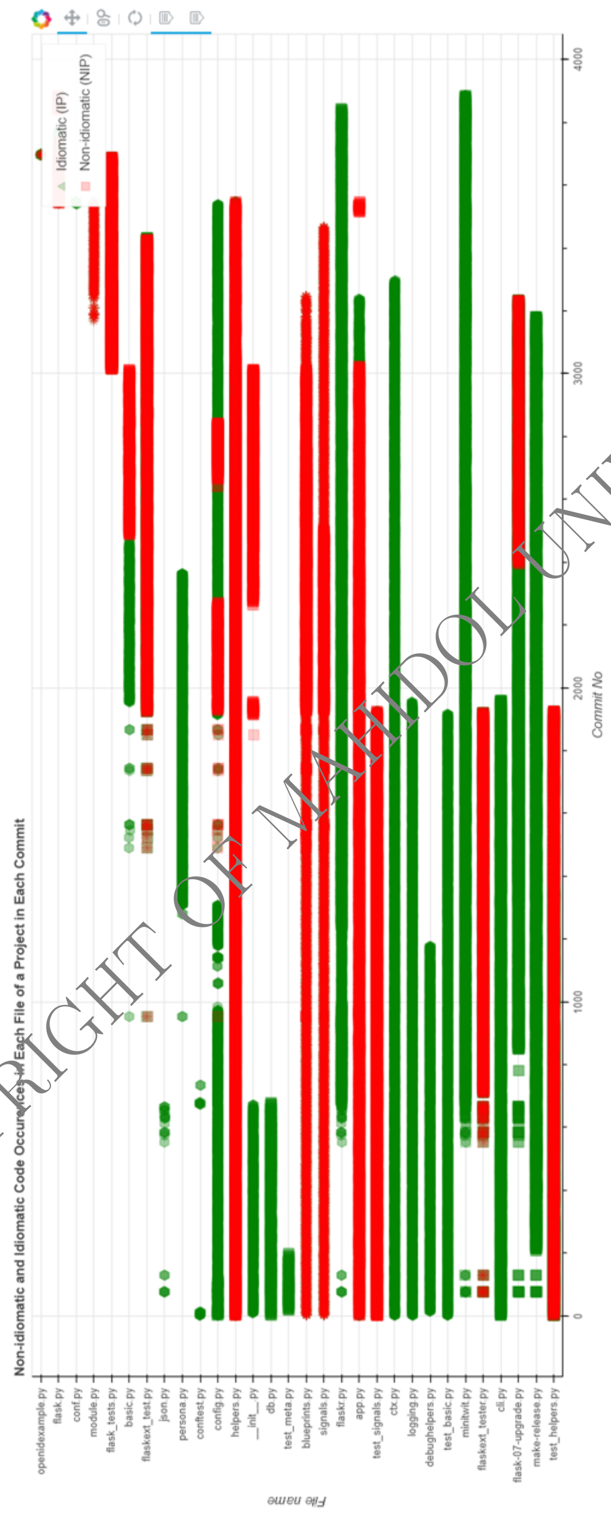


Figure 5.2: Visualization of IP and NIP usage in GitHub project Flask

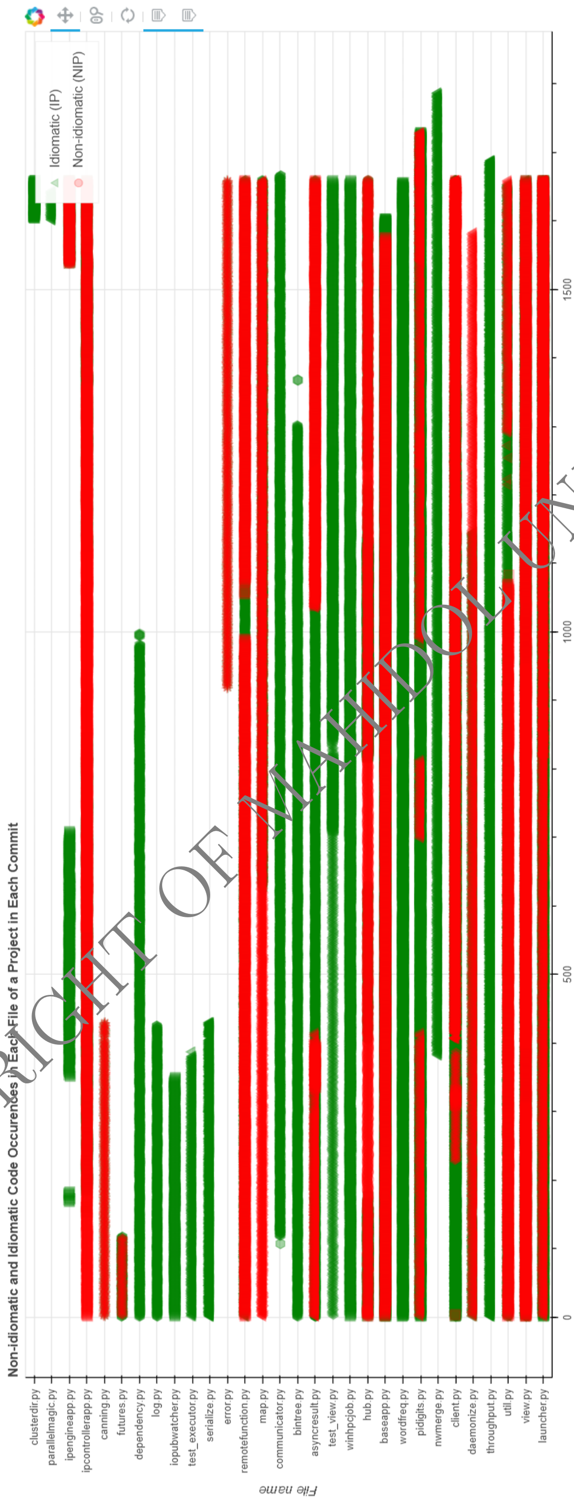


Figure 5.3: Visualization of IP and NIP usage in GitHub project ipyparallel

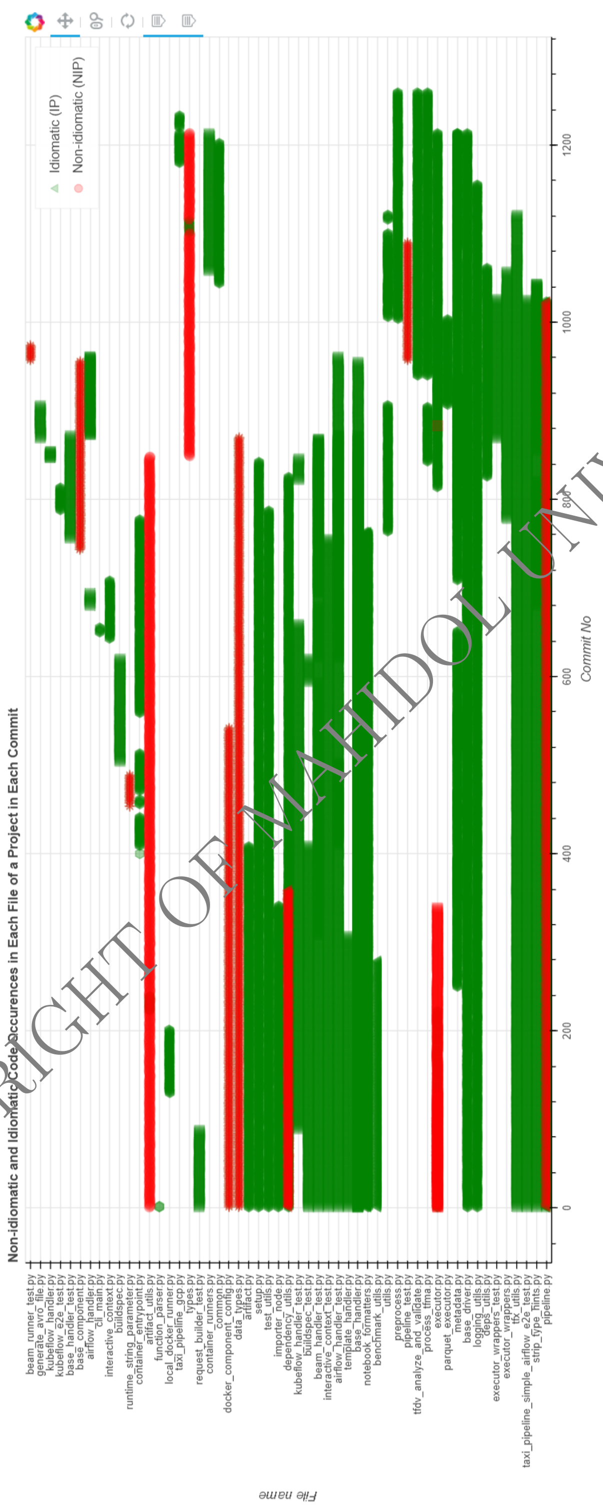


Figure 5.4: Visualization of IP and NIP usage in GitHub project tfx

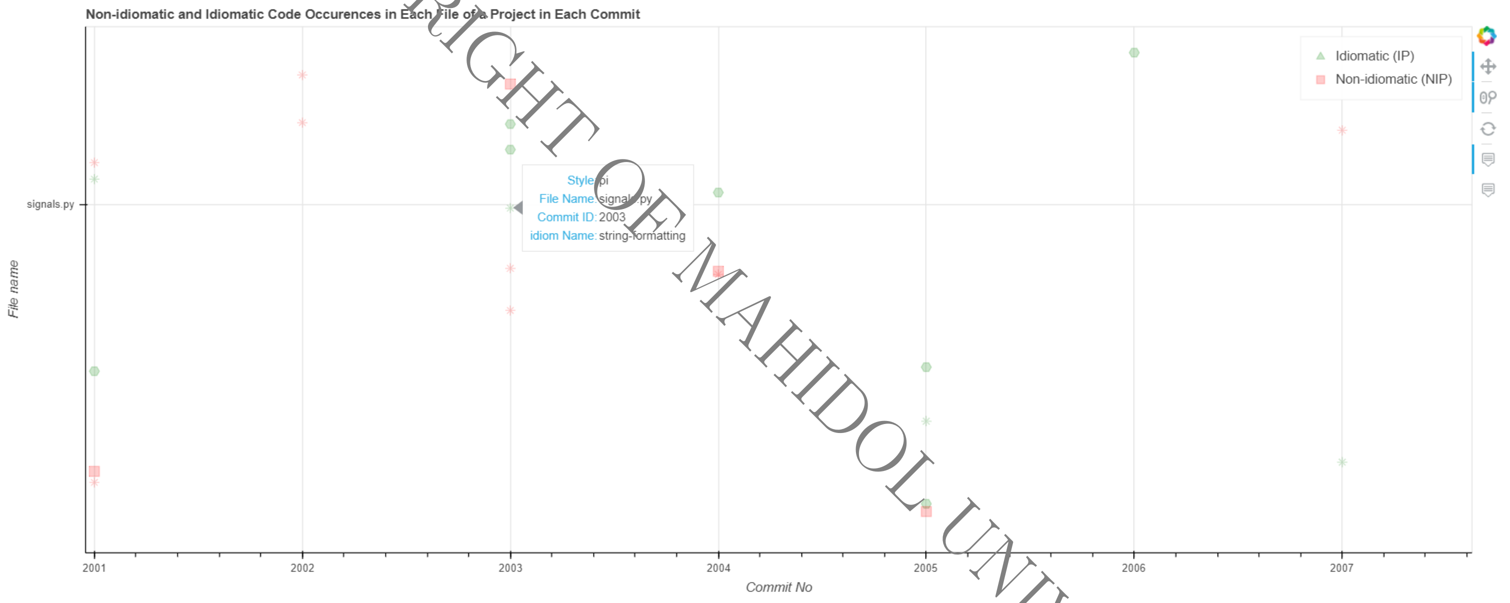


Figure 5.5: Zoom-in view of visualization of IP and NIP usage in GitHub project Flask

CHAPTER 6
CONCLUSION

This chapter summarizes the research and discuss the limitations, including the future directions of this project.

6.1 Conclusion

The main goal of this project is to give programmers a tool that can help them to analyze their code in a GitHub repository. We create an automated tool called Teddy that can detect the usage of idiomatic Python code during the development (code review time) and over historical commits. Teddy integrates several tools and techniques including GitHub integration (GitHub API, Probot, Smee.io), idiomatic code detection (SiameseX) and idiomatic code visualization (Bokeh library).

The Teddy tool contains two modes of usage. The first mode is prevention mode, in which the tool works actively to give response to the user at real-time during pull requests. The other mode is detection mode, in which the tool gives feedback from going back to the start of the project and analyze the IPs and NIPs usage over all the project's commits.

We have evaluated the Teddy tool using a synthetic idiomatic code data set and thee real software projects. Using Mean Average Precision (MAP), Query Recall, Overall Recall, Mean Reciprocal Rank (MRR) to evaluate the accuracy of the tool. We have found that Teddy gives relatively high precision for idiomatic and non-idiomatic Python code detection. Moreover, the evaluation based on the experiment on real Python projects shows that Teddy can visualize the usage of IPs and NIPs with over 3,000 commits.

This Teddy tool is a valuable addition to nowadays modern software development and can be plugged-in to GitHub seamlessly. We hope that the tool will be useful for Thai programmers and other programmers around the world.

6.2 Problems and Limitations

For the prevention mode with the tool being unable to handle a large number of incoming queries, the response time for handling multiple requests can be observed, as the tool can only handle one request at a time.

From using only one synthetic data set (due to time constraint in creating a ground-truth base from an actual soft repository) to test the tool, the level of performance, both precision and recall, is subject to vary for different application on different software projects. Also, regarding CCGrep, the tool offers little varieties of regular expression pattern that were needed in the process of creating the synthetic testing data set, causing many of the sampled IPs and NIPs to be inaccurate.

With the current performance of Teddy, despite its high precision, the tool is not able to identify the remaining IP and NIP code snippets as many as expected, therefore resulting in very low recall. One possible explanation is that with the unique semantic nature of idiomatic Python codes, it is impractical to capture the pattern with the principle of code clone detection and its multi-representation formats. The tool also has a problem when being left idle for a while, causing it to stop functioning until a request has been made several times.

For the visualization, the limitation is that it is not flexible enough to manipulate the plot and the data in the plot. The plot can only handled by JavaScript while the data is nearly impossible to adjust throughout the plot later which means that the data must be ready and completed to be plotted.

6.3 Future Work

Since the unfortunate event of COVID-19 (Coronavirus 2019) prohibiting us from doing the user studies, our future work is to have Teddy successfully tested with the expected target user groups in order to obtain comments from real user experiences and get feedback to develop our tool to reach a higher level of efficiency and usability. The initial plan was to conduct a user study on Teddy with the target user as software developers who play the role of both developers and code reviewers. The research procedure starts with making the participants do the pre-test which involves their background knowledge of IPs and NIPs. Then, we, the researchers, would set up the Teddy software for the participants to use. The participants would be given time to use the tool between 2 to 4 weeks to make sure they are familiar and keen enough to use the tool efficiently. At the end of the usage period, the participants will be instructed to do the same test once again (post-test). Moreover, we would interview them in additional to receiving their comments about the tool.

In the future we aim to have the tool be integrated into GitHub repositories without any problem regarding response time and handling multiple queries. We also intend to expand the number of NIPs and IPs types to be included in Teddy's functionality to support wider scope of coding style and idioms. There are also many other unexplored minor settings of Siamese (and SiameseX) that remains potential key factors to the accuracy and recall of the search retrieval mechanism. In addition, we would like to try more flexible and adjustable visualization tool that would liberate the form of our visualization into more interactive way.

Moreover, if our tool can present the relationship between the appearing and disappearing IP/NIP Python codes in the project, it would be more beneficial for the user to track the evolution and usage of idiomatic Python code.

APPENDIX A
IP CODE SNIPPETS

Listing A.1: List of IP dictionary comprehension code snippets

```
def i1():
        emails = {user.name: user.email for user in users if user.email}

def i2():
        dict_compr = {k: k**2 for k in range(10000)}

def i3():
        new_dict_comp = {n:n**2 for n in numbers if n%2 == 0}

def i4():
        dict1 = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f':6}
        dict1_tripleCond = {k:v for (k,v) in dict1.items() if v>2 if v%2 == 0
            ↪  if v%3 == 0}
        print(dict1_tripleCond)

def i5():
        nested_dict = {'first':{'a':1}, 'second':{'b':2}}
        float_dict = {outer_k: {float(inner_v) for (inner_k, inner_v) in
            ↪ outer_v.items()} for (outer_k, outer_v) in nested_dict.items()}
        print(float_dict)

def i6():
        # Initialize the `fahrenheit` dictionary
        fahrenheit = {'t1': -30,'t2': -20,'t3': -10,'t4': 0}
        # Get the corresponding `celsius` values and create the new
            ↪ dictionary
        celsius = {k:(float(5)/9)*(v-32) for (k,v) in fahrenheit.items()}
        print(celsius_dict)

def i7():
        mcase = {'a':10, 'b': 34, 'A': 7, 'Z':3}
        mcase_frequency = { k.lower() : mcase.get(k.lower(), 0) + mcase.get(k
            ↪ .upper(), 0) for k in mcase.keys() }
```

Listing A.2: List of IP `enumerate` code snippets

```
def i8():
    for i, x in enumerate(l):
        # ...

def i9():
    try:
        x = next(i for i, n in enumerate(l) if n > 0)
    except StopIteration:
        print('No positive numbers')
    else:
        print('The index of the first positive number is', x)

def i10():
    ls = list(range(10))
    for index, value in enumerate(ls):
      print(value, index)

def i11():
    a = [3, 4, 5]
    for i, item in enumerate(a):
        print i, item

def i12():
    for i, val in enumerate(array):
        #do stuff with i
        #do stuff with val

def i13():
    for index, element in enumerate(my_container):
        print (index, element)

def i14():
    my_list = ['apple', 'banana', 'grapes', 'pear']
    for c, value in enumerate(my_list, 1):
        print(c, value)

def i15():
    my_list = ['apple', 'banana', 'grapes', 'pear']
    counter_list = list(enumerate(my_list, 1))
    print(counter_list)
```

Listing A.3: List of IP file reading statement code snippets

```
def i16():
    with open('file.txt') as f:
        for line in f:
            print line

def i17():
    with open(path, "rb") as f:
        result = do_something_with(f)
        print("Got result: {}".format(result))

def i18():
    with open('file.ext') as f:
        contents = f.read()

def i19():
    with open("welcome.txt") as file:
        data = file.read()
        do something with data

def i20():
    with open(path_to_file, 'r') as file_handle:
        for line in file_handle:
            if raise_exception(line):
                print('No! An Exception!')
```

Listing A.4: List of IP list comprehension code snippets

```
def i21():
    result_list = [el for el in range(10000000)]

def i22():
        [print(i) for i in wordList]

def i23():
        new_list = [n**2 for n in numbers if n%2==0]

def i24():
        ls = [element for element in range(10) if not(element % 2)]
```

```
def i25():
        valedictorian = max([(student.gpa, student.name) for student in
            ↪ graduates])

def i26():
        a = [3, 4, 5]
        b = a
        a = [i + 3 for i in a]

def i27():
        return [[float(a_ij) for a_ij in a_i]
            for a_i in matrix_of_anything]
```

Listing A.5: List of IP `if` statement code snippets

```
def i28(countNotMax):
if countNotMax:
    # Some code here

def i29():
    if itemListEmpty():
        return "List is empty"

def i30(es):
    if not recreateIndex:
        es.connect()

def i31():
    if not gitHubRepo():
        for doc in index.getDoc:
            doc.setLicense(null)

def i32():
    num = input("Enter weight: ")
    if not num:
        print("No input found")
        else
            print("Processing your input")

def i33():
    name = 'Tom'
    is_generic_name = name in ('Tom','Dick','Harry')
```

```
def i34():
    num = 2
    prime_less_than_10 = num in (5,3,2,7)
    return prime_less_than_10

def i35():
    char = input("Enter a character A-Z")
    if char in ('A','E','I','O','U')
        print("Input is an vowel")

def i36():
    if itemListEmpty():
        return "List is empty"

def i37():
    if name:
        print(name)
        print(address)
        count++

def i38(sentence):
    if(sentence.endswith('?'))
        return 'Interrogative sentence'
        else
            return 'Informative sentence'
```

Listing A.6: List of IP string formatting code snippets

```
def i39(Store):
output = 'ID: {s.branch_ID}, City: {s.city}, Manager: {s.manager}'.format(s=
    ↪ Store)
return output

def i40(self, name, age, gender):
        self.name = name
        self.age = age
        self.gender = gender
    person = Person("John", 36, 'M')
    return 'Name: {p.name}\nAge: {p.age}\nGender: {p.gender}'.format(person=
        ↪ p1)

def i41():
    book_info = ' The Three Musketeers: Alexandre Dumas'
    formatted_book_info = book_info.strip().upper().replace(':', ' by').
        ↪ append(', ISBN:')
```

Listing A.7: List of IP set code snippets

```
def i42():
student_nationality = ['Thai','Malaysian','Thai','Vietnamese','Vietnamese','
    ↪ Vietnamese','Singaporean','Laos','Cambodian','Cambodian','Chinese']
unique_nationality = set(student_nationality)

def i43():
    staff_name = ['Catherine','Bryan', 'Kevin', 'Frank', 'Emily', 'Steven', '
        ↪ George', 'Hallen', 'Sasha', 'Nathan', 'Edward', 'Phillip', '
        ↪ Scarlet', 'Robert']
    staff_year_of_birth = [1997, 1960, 1971, 1982, 1990, 1995, 1994, 1960,
        ↪ 1983, 1997, 1996, 1960, 1981, 1982]
    unique_year_of_birth = set(staff_year_of_birth)

def i44():
    max_temp = [35.6, 34.7, 34.7, 36.1, 36.4, 36.8, 36.2, 36.2, 35.1, 35.0]
    min_temp = [27.1, 27.0, 26.8, 26.8, 27.0, 27.5, 27.2, 27.2, 26.9, 26.7]
    unique_max_temp = set(max_temp)
    unique_min_temp = set(min_temp)

def i45():
    grade = ['A','B','B','B','C','D','F','C','C','D','A']
    unique_grade = set(grade)
```

Listing A.8: List of IP tuple code snippets

```
def i46():
    list_from_comma_separated_value_file = ['dog', 'Fido', 10]
    (animal, name, age) = list_from_comma_separated_value_file

def i47():
    catherine_info = ['Catherine', 1960, 'Australian', 'F', 165, 50, 'Trainee
        ↪ ']
    class Staff:
        name = ''
        year-of-birth = 0
        nationality = ''
        gender = ''
        height = 0
        weight = 0
        position = ''
    cat = Staff()
    (cat.name, cat.year-of-birth, cat.nationality, cat.gender, cat.height,
       ↪ cat.position) = catherine_info

def i48{():
    DOB_numbers = [11,27,1,9,1996]
    (h,m,D,M,Y) = DOB_numbers
    return '%i:%i %i-%i-%i' % (h,m,D,M,Y)

def i49():
    blood_groups = ['A','B','O','B']
    class Person:
        blood = 'X'
    p1,p2,p3,p4 = Person()
    (p1.blood,p2.blood,p3.blood,p4.blood) = blood_groups
```

Listing A.9: List of IP variable swapping code snippets

```
def i50():
seat_A1 = 'Mike Wazowski'
seat_A2 = 'James Sullivan'
(seat_A1, seat_A2) = (seat_A2, seat_A1)

def i51(var_A, var_B):
    (var_A, var_B) = (var_B, var_A)

def i52():
    english = 4.0
    math = 3.5
    (english, math) = (math, english)

def i53(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                (arr[j], arr[j+1]) = (arr[j+1], arr[j])
```

Listing A.10: List of IP code formatting code snippets

```
def i54():
if file_name == "-":
    module = types.ModuleType("input_scenes")
    code = "from manimlib.imports import *\n\n" + sys.stdin.read()
    try:
        exec(code, module.__dict__)
        return module
    except Exception as e:
        print(f"Failed to render scene: {str(e)}")
        sys.exit(2)
else:
    module_name = file_name.replace(os.sep, ".").replace(".py", "")
    spec = importlib.util.spec_from_file_location(module_name, file_name)
    module = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(module)
    return module

def i55():
    self.set_cairo_context_path(ctx, vmobject)
    self.apply_stroke(ctx, vmobject, background=True).apply_fill(ctx,
        ↪ vmobject).apply_stroke(ctx, vmobject)
    return self
```

APPENDIX B
NIP CODE SNIPPETS

Listing B.1: List of NIP dictionary comprehension code snippets

```
def n1():
    emails = {}
    for user in users:
      if user.email:
        emails[user.name] = user.email

def n2():
    d = {}
    for k in range(10000):
        d[k] = k**2

def n3():
    for n in numbers:
        if n%2==0:
            new_dict_for[n] = n**2

def n4():
    dict1_tripleCond = {}
    for (k,v) in dict1.items():
        if (v>=2 and v%2 == 0 and v%3 == 0):
                dict1_tripleCond[k] = v
    print(dict1_tripleCond)

def n5():
    nested_dict = {'first':{'a':1}, 'second':{'b':2}}
    for (outer_k, outer_v) in nested_dict.items():
        for (inner_k, inner_v) in outer_v.items():
            outer_v.update({inner_k: float(inner_v)})
    nested_dict.update({outer_k:outer_v})
    print(nested_dict)

def n6():
    fahrenheit = {'t1':-30, 't2':-20, 't3':-10, 't4':0}
    #Get the corresponding `celsius` values
    celsius = list(map(lambda x: (float(5)/9)*(x-32), fahrenheit.values()))
    #Create the `celsius` dictionary
    celsius_dict = dict(zip(fahrenheit.keys(), celsius))
    print(celsius_dict)
```

Listing B.2: List of NIP `enumerate` code snippets

```
def n7():
    for i in range(len(l)):
        x = l[i]
    try:
        x = next(i for i, n in enumerate(l) if n > 0)
    except StopIteration:
        print('No positive numbers')
    else:
        print('The index of the first positive number is', x)

def n8():
    x = next(n for n in l if n > 0)
    except StopIteration:
        print('No positive numbers')
    else:
        print('The first positive number is', x)

def n9():
    ls = list(range(10))
    index = 0
    while index < len(ls):
      print(ls[index], index)
      index += 1

def n10():
    # Add three to all list members.
    a = [3, 4, 5]
    b = a #a and b refer to the same list object
    for i in range(len(a)):
        a[i] += 3 #b[i] also changes

def n11():
    for i in range(len(array)):
        #do stuff with i
        #do stuff with array[i]
```

```
def n12():
    index = 0
    for element in my_container:
        print (index, element)
        index+=1
```

Listing B.3: List of NIP file reading statement code snippets

```
def n13():
    f = open('file.txt')
    a = f.read()
    print a
    f.close()

def n14():
    f = open(path, "rb")
    result = do_something_with(f)
    f.close()
    print("Got result: {}".format(result))

def n15():
    f = open('file.ext')
    try:
      contents = f.read()
    finally:
      f.close()

def n16():
    file = open("welcome.txt")
    data = file.read()
    print data
    file.close()

def n17():
    file_handle = open(path_to_file, 'r')
    for line in file_handle.readlines():
        if raise_exception(line):
            print('No! An Exception!')
```

Listing B.4: List of NIP list comprehension code snippets

```
def n18():
    result_list = []
    for el in range(10000000) :
        result_list.append(el)

def n19():
    for i in range(len(wordList)) :
        print(wordList[i])
        i += 1

def n20():
    ls = []
    for element in range(10):
        if not (element%2):
            ls.append(element)

def n21():
    new_list = []
    for n in numbers:
        if n%2==0:
            new_list.append(n**2)

def n22():
    list = [1, 3, 5, 7, 9]
    while i < length:
        print(list[i])
        i += 1

def n23():
    ls = list(filter(lambda element: not(element % 2), range(10)))

def n24():
    a = [3, 4, 5]
    b = a
    for i in range(len(a)):
        a[i] += 3

def n25(matrix_of_anything):
    n = len(matrix_of_anything)
    n_i = len(matrix_of_anything[0])
    new_matrix_of_floats = []
    for i in xrange(0, n):
        row = []
        for j in xrange(0, n_i):
            row.append(float(matrix_of_anything[i][j]))
        new_matrix_of_floats.append(row)
    return new_matrix_of_floats
```

Listing B.5: List of NIP `if` statement code snippets

```
def n26(countNotMax):
    if countNotMax == True:
        # Some code here

def n27():
    if itemListEmpty() == True:
        return "List is empty"

def n28(es):
    if recreateIndex == False:
        es.connect()

def n29():
    if gitHubRepo() == False:
        for doc in index.getDoc:
            doc.setLicense(null)

def n30():
    num = input("Enter weight: ")
    if num == None:
        print("No input found")
        else
        print("Processing your input")

def n31():
    name = 'Tom'
    if name == 'Tom' or name == 'Dick' or name == 'Harry':
         is_generic_name = True

def n32():
    num = 2
    if num = 5 or num = 3 or num = 2 or num = 7:
        prime_less_than_10 = True

def n33():
    char = input("Enter a character A-Z")
    if char = 'A' or char = 'E' or char = 'I' or char = 'O' or char = 'U':
        print("Input is an vowel")

def n34():
    if itemListEmpty() == True: return "List is empty"

def n35():
    if name: print(name); print(address); count++;

def n36():
    if !gitHubRepo(): for doc in index.getDoc: doc.setLicense(null);

def n37(sentence):
        if(sentence.endswith('?')) return 'Interrogative sentence'; else
            ↪ return 'Informative sentence';
```

Listing B.6: List of NIP string formatting code snippets

```
def n38(Store):
return 'ID: ' + Store.branch_ID + ' City: ' + Store.city + ' Manager: ' +
    ↪ Store.manager

def n39(Store):
    return 'ID: %i City: %s Manager: %s' % (Store.branch_ID, Store.city,
        ↪ Store.Manager)

def n40(self, name, age, gender):
        self.name = name
        self.age = age
        self.gender = gender
    person = Person("John", 36, 'M')
    return 'Name: ' + person.name + '\nAge: ' + person.age + '\nGender: ' +
        ↪ person.gender

def n41(self, name, age, gender):
        self.name = name
        self.age = age
        self.gender = gender
    person = Person("John", 36, 'M')
    return 'Name: %s\nAge: %i\nGender: %c' % (person.name, person.age, person
        ↪ .gender)

def n42():
    book_info = ' The Three Musketeers: Alexandre Dumas'
    formatted_book_info = book_info.strip()
    formatted_book_info = formatted_book_info.upper()
    formatted_book_info = formatted_book_info.replace(':', ' by')

def n43(sentence):
    output = sentence.capitalize()
    output = output.swapcase()
    output = output.replace('I\'m', 'I am')
    output = output.replace('You\'re','You are')
    output = output.replace('can\'t','cannot')
    return output

def n44(sentence):
    formatted = sentence.capitalize()
    formatted = output.swapcase()
    return formatted.endswith('.')
```

Listing B.7: List of NIP set code snippets

```
def n45():
student_nationality = ['Thai','Malaysian','Thai','Vietnamese','Vietnamese','
    ↪ Vietnamese','Singaporean','Laos','Cambodian','Cambodian','Chinese']
unique_nationality = []
for nationality in student_nationality:
    if nationality not in unique_nationality:
        unique_nationality.append(nationality)

def n46():
    staff_name = ['Catherine','Bryan', 'Kevin', 'Frank', 'Emily', 'Steven', '
        ↪ George', 'Hallen', 'Sasha', 'Nathan', 'Edward', 'Phillip', '
        ↪ Scarlet', 'Robert']
    staff_year_of_birth = [1997, 1960, 1971, 1982, 1990, 1995, 1994, 1960,
        ↪ 1983, 1997, 1996, 1960, 1981, 1982]
    unique_year_of_birth = []
    for year in staff_year_of_birth:
        if year not in unique_year_of_birth:
            unique_year_of_birth.append(year)

def n47():
    max_temp = [35.6, 34.7, 34.7, 36.1, 36.4, 36.8, 36.2, 36.2, 35.1, 35.0]
    min_temp = [27.1, 27.0, 26.8, 26.8, 27.0, 27.5, 27.2, 27.2, 26.9, 26.7]
    unique_max_temp = []
    unique_min_temp = []
    for temp in max_temp:
        if temp not in unique_max_temp:
            unique_max_temp.append(temp)
    for temp in min_temp:
        if temp not in unique_min_temp:
            unique_min_temp.append(temp)

def n48():
    grade = ['A','B','B','B','C','D','F','C','C','D','A']\
    student_placeholder = 'John Doe'
    unique_grade = []
    for g in grade:
        if g not in unique_grade:
            unique_grade.append(g)
```

Listing B.8: List of NIP tuple code snippets

```
def n49():
list_from_comma_separated_value_file = ['dog', 'Fido', 10]
animal = list_from_comma_separated_value_file[0]
name = list_from_comma_separated_value_file[1]
age = list_from_comma_separated_value_file[2]

def n50():
    catherine_info = ['Catherine', 1960, 'Australian', 'F', 165, 50, 'Trainee
        ↪ ']
    class Staff:
        name = ''
        year-of-birth = 0
        nationality = ''
        gender = ''
        height = 0
        weight = 0
        position = ''
    cat = Staff()
    cat.name = catherine_info[0]
    cat.year-of-birth = catherine_info[1]
    cat.nationality = catherine_info[2]
    cat.gender = catherine_info[3]
    cat.height = catherine_info[4]
    cat.weight = catherine_info[5]
    cat.position = catherine_info[6]
```

```
def n51():
    DOB_numbers = [11,27,1,9,1996]
    h = DOB_numbers[0]
    m = DOB_numbers[1]
    D = DOB_numbers[2]
    M = DOB_numbers[3]
    Y = DOB_numbers[4]
    return '%i:%i %i-%i-%i' % (h,m,D,M,Y)

def n52():
    blood_group = ['A','B','O','B']
    class Person:
        blood = 'X'
    p1 = Person()
    p1.blood = blood_group[0]
    p2 = Person()
    p2.blood = blood_group[1]
    p3 = Person()
    p3.blood = blood_group[2]
    p4 = Person()
    p4.blood = blood_group[3]
```

Listing B.9: List of NIP variable swapping code snippets

```
def n53():
seat_A1 = 'Mike Wazowski'
seat_A2 = 'James P. Sullivan'
temp = seat_A1
seat_A1 = seat_A2
seat_A2 = temp

def n54(numA, numB):
    temp = numA
    numA = numB
    numB = temp

def n55():
    english = 4.0
    math = 3.5
    temp = english
    english = math
    math = temp
```

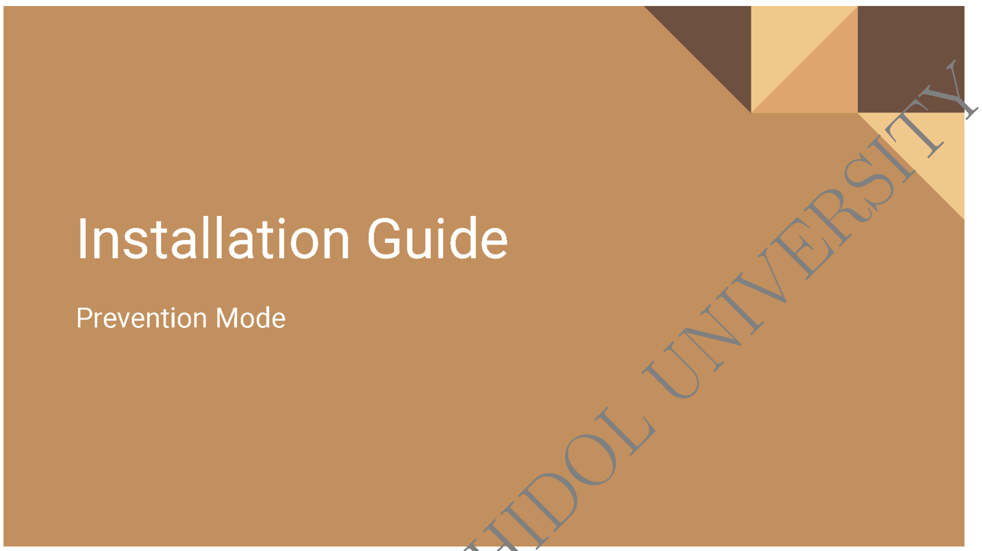
```
def n56(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                temp = arr[j]
                arr[j] = arr[j+1]
                arr[j+1] = temp
```

Listing B.10: List of NIP code formatting code snippets

```
def n57(file_name):
    if file_name == "-":
        module = types.ModuleType("input_scenes"); code = "from manimlib.
            ↪ imports import *\n\n" + sys.stdin.read()
        try:
            exec(code, module.__dict__); return module;
        except Exception as e:
            print(f"Failed to render scene: {str(e)}"); sys.exit(2);
    else:
        module_name = file_name.replace(os.sep, ".").replace(".py", ""); spec
            ↪  = importlib.util.spec_from_file_location(module_name,
            ↪ file_name); module = importlib.util.module_from_spec(spec);
        spec.loader.exec_module(module)
        return module

def n58(self, vmobject, ctx):
    self.set_cairo_context_path(ctx, vmobject); self.apply_stroke(ctx,
        ↪ vmobject, background=True); self.apply_fill(ctx, vmobject); self.
        ↪ apply_stroke(ctx, vmobject);
    return self
```

APPENDIX C
PREVENTION MODE INSTALLATION

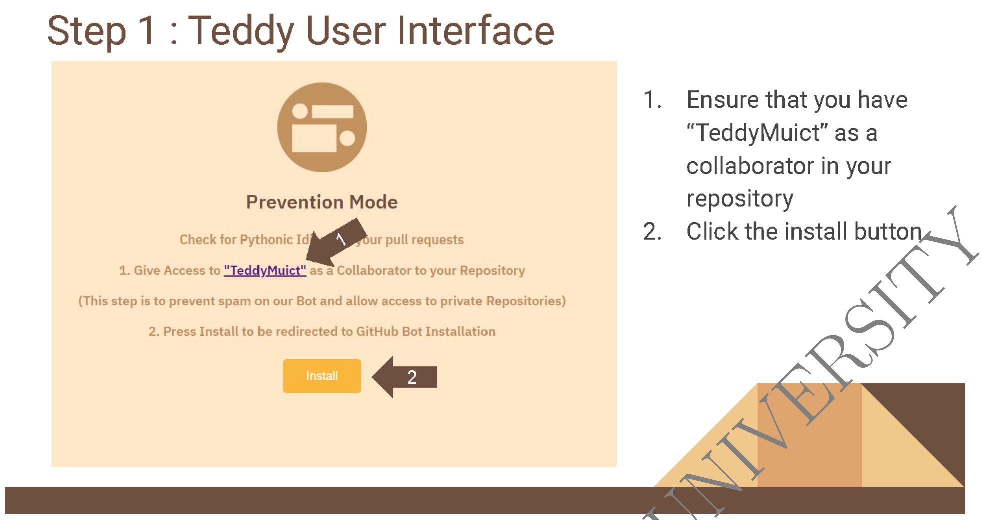

Installation Guide

Prevention Mode

Figure C.1: Installation guide for Teddy prevention mode

C.1 Prevention Mode Installation guide

The prevention mode for Teddy can be installed by following the guidelines provided in the Teddy user interface, these will include a total of 4 different steps as shown in C.2, C.3, C.4 and C.5.



Step 1 : Teddy User Interface

Prevention Mode

Check for Pythonic Idioms in your pull requests

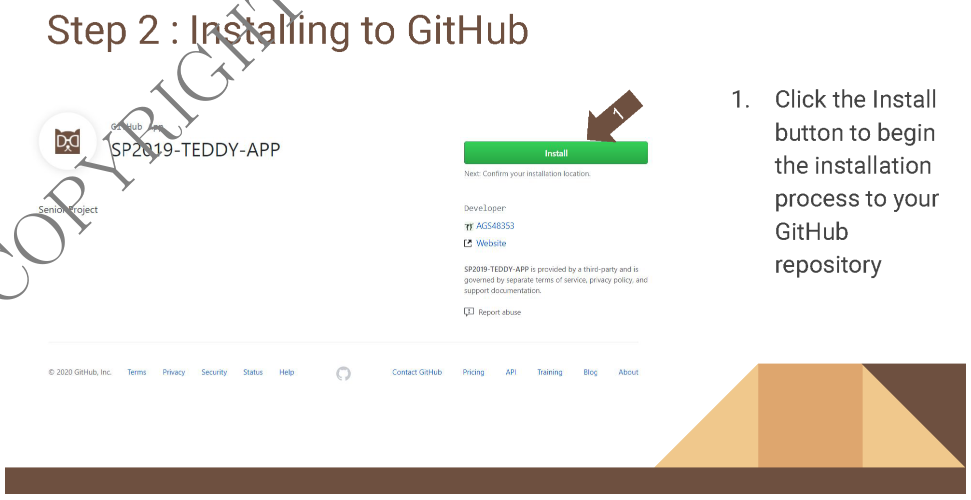
1. Give Access to "TeddyMuict" as a Collaborator to your Repository

(This step is to prevent spam on our Bot and allow access to private Repositories)

2. Press Install to be redirected to GitHub Bot Installation

1. Ensure that you have "TeddyMuict" as a collaborator in your repository
2. Click the install button

Figure C.2: Prevention mode installation guide step 1



Step 2 : Installing to GitHub

GitHub App

SP2019-TEDDY-APP

Senior Project

Next: Confirm your installation location.

Developer

AGS48353

Website

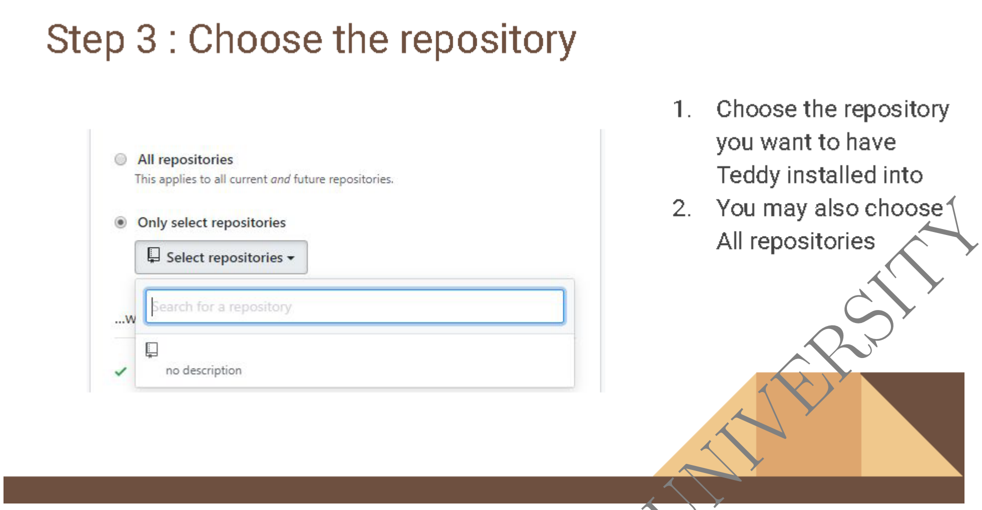
SP2019-TEDDY-APP is provided by a third-party and is governed by separate terms of service, privacy policy, and support documentation.

Report abuse

© 2020 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub Pricing API Training Blog About

1. Click the Install button to begin the installation process to your GitHub repository

Figure C.3: Prevention mode installation guide step 2



Step 3 : Choose the repository

All repositories
This applies to all current *and* future repositories.

Only select repositories

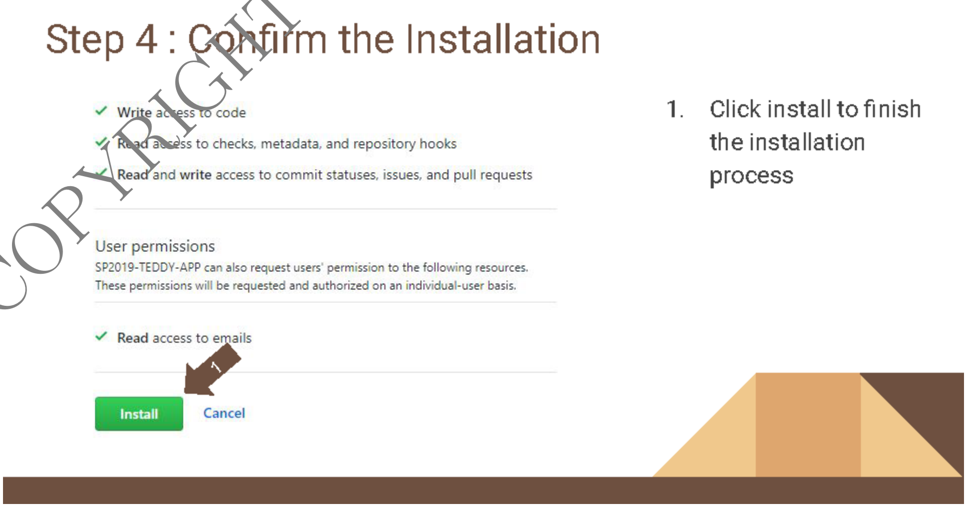
Select repositories ▾

Search for a repository

no description

1. Choose the repository you want to have Teddy installed into
2. You may also choose All repositories

Figure C.4: Prevention mode installation guide step 4



Step 4 : Confirm the Installation

✓ **Write** access to code
✓ **Read** access to checks, metadata, and repository hooks
✓ **Read** and **write** access to commit statuses, issues, and pull requests

User permissions
SP2019-TEDDY-APP can also request users' permission to the following resources.
These permissions will be requested and authorized on an individual-user basis.

✓ **Read** access to emails

1. Click install to finish the installation process

Figure C.5: Prevention mode installation guide step 5

REFERENCES

[1] Ragkhitwetsagul C., Krinke J., "Siamese: scalable and incremental code clone search via multiple code representations", Empirical Software Engineering. 2019;p. 1–49.

[2] BEN PUTANO s., "A Look At 5 of the Most Popular Programming Languages of 2019"; November 2019 [cited 14 November 2019], [Online]. Available: https:// stackify.com/popular-programming-languages-2018.

[3] Knupp J., Writing Idiomatic Python 3.3, Amazon; 2013, [Online]. Available: https://www.amazon.com/Writing-Idiomatic-Python-Jeff-Knupp/dp/1482374811.

[4] "What is idiomatic code?"; 2008 [cited 12 May 2020], [Online]. Available: https:// stackoverflow.com/questions/84102/what-is-idiomatic-code.

[5] Foundation PS., editor, . "Glossary - Python 3.8.0 documentation"; [updated 13 November 2019; cited 10 November 2019], [Online]. Available: https:// docs.python.org/3/glossary.html.

[6] Guttag JV., 2nd ed. The MIT Press; 2016, [Online]. Available: https:// www.amazon.com/ Introduction-Computation-Programming-Using-Python-ebook/dp/B01K6F2236.

[7] "The 10 most popular programming languages, according to the Microsoft-owned GitHub"; November 2019 [cited 12 November 2019], [Online]. Available: https:// www.businessinsider.com/ most-popular-programming-languages-github-2019-11.

[8] Alexandru CV., Merchante JJ., Panichella S., Proksch S., Gall HC., Robles G., "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. ACM; 2018. p. 1–11.

[9] Roy CK., Cordy JR., Koschke R., "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach", Science of Computer Programming. 2009;74(7):470 – 495, [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167642309000367.

[10] Peters T.. "PEP 20 – The Zen of Python", Python Software Foundation; 2004, [Online]. Available: https://www.python.org/dev/peps/pep-0020/.

[11] "Pythonic"; [cited 12 November 2019], [Online]. Available: https://pythonic-examples.github.io/.

[12] Chatley R., Jones L., "Diggit: Automated code review via software repository mining", In: 25th International Conference on Software Analysis, Evolution and Reengineering, SANER 2018, Campobasso, Italy, March 20-23, 2018; 2018. p. 567–571, [Online]. Available: https://doi.org/10.1109/SANER.2018.8330261.

[13] "Stack Overflow | Where developers learn share & build careers"; [cited 14 November 2019], [Online]. Available: https://stackoverflow.com/.

[14] Ragkhitwetsagul C., Krinke J., Paixão M., Bianco G., Oliveto R., "Toxic Code Snippets on Stack Overflow", ArXiv. 2018;abs/1806.07659.

[15] GitHub I., editor, . "The world's leading software development platform · GitHub"; [cited 14 November 2019], [Online]. Available: https://github.com/.

[16] GitHub I., editor, . "About · GitHub"; [cited 14 November 2019], [Online]. Available: https://github.com/about.

[17] GitHub I., editor, . "The State of the Octoverse | The State of the Octoverse celebrates a year of building across teams, time zones, and millions of merged pull requests."; [cited 14 November 2019], [Online]. Available: https://octoverse.github.com/footnote–community-overview–developer-count.

[18] "Probot | GitHub Apps to automate and improve your workflow"; [cited 14 November 2019], [Online]. Available: https://probot.github.io/.

[19] "smee.io | Webhook payload delivery service"; [cited 14 November 2019], [Online]. Available: https://smee.io/.

[20] V. EB., editor, . "Elasticsearch: The Official Distributed Search & Analytics Engine | Elastic"; [cited 14 November 2019], [Online]. Available: https://www.elastic.co/products/elasticsearch.

[21] V. EB., editor, . "What is Elasticsearch | Elastic"; [cited 14 November 2019], [Online]. Available: https://www.elastic.co/what-is/elasticsearch.

[22] Zolkifli NN., Ngah A., Deraman A., "Version Control System: A Review", Procedia Computer Science. 2018;135:408 – 415, The 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018) : Empowering Smart Technology in Digital Era for a Better Life, [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050918314819.

[23] "Bash - GNU Project - Free Software Foundation"; 2017 [updated 15 December 2017; cited 17 April 2020], [Online]. Available: https://www.gnu.org/software/bash/.

[24] "Git"; 2020 [cited 17 April 2020], [Online]. Available: https://git-scm.com/.

[25] "bokeh/bokeh: Interactive Data Visualization in the browser, from Python"; 2020 [updated 17 April 2020; cited 17 April 2020], [Online]. Available: https://github.com/bokeh/bokeh.

[26] Inoue K., Miyamoto Y., German D., Ishio T., "Code Clone Matching: A Practical and Effective Approach to Find Code Snippets". 3 2020;.

BIOGRAPHIES

NAME	Mr. Purit Phan-udom
DATE OF BIRTH	1 September 1996
PLACE OF BIRTH	Bangkok, Thailand
INSTITUTIONS ATTENDED	Matthayomwatnairong, 2016: High School Diploma Mahidol University, 2020: Bachelor of Science (ICT)

NAME	Mr. Naruedon Wattanakul
DATE OF BIRTH	14 December 1996
PLACE OF BIRTH	Aberdeen, Scotland
INSTITUTIONS ATTENDED	Benchamamaharat School, 2016: High School Diploma Mahidol University, 2020: Bachelor of Science (ICT)

NAME	Ms. Tattiya Sakulniwat
DATE OF BIRTH	19 July 1997
PLACE OF BIRTH	Bangkok, Thailand
INSTITUTIONS ATTENDED	Satriwithaya School, 2016: High School Diploma Mahidol University, 2020: Bachelor of Science (ICT)