Background Report: CodeShovel::Python3

Braxton J Hall
The University of British Columbia
Vancouver, British Columbia, Canada
braxtonhall@alumni.ubc.ca

Brian Yeung
The University of British Columbia
Vancouver, British Columbia, Canada
brianyeung@alumni.ubc.ca

Danhui Jia
The University of British Columbia
Vancouver, British Columbia, Canada
danhui.jia@alumni.ubc.ca

Lilli Freischem
The University of British Columbia
Vancouver, British Columbia, Canada
lilli.freischem@alumni.ubc.ca

ABSTRACT

CodeShovel is a static analysis tool that navigates through a git commit history backward like a linked list, builds an Abstract Syntax Tree for each file, and interprets it looking for modifications to a specified method. This can be useful for isolating changes to methods, assisting in predicting defects and explaining the story of how a method came to be. This paper demonstrates the potential impact of and design plan for a Python 3 extension for CodeShovel.

ACM Reference Format:
Braxton J Hall, Brian Yeung, Danhui Jia, and Lilli Freischem. 2018. Background Report: CodeShovel::Python3. In *CPSC 311: Definition of Programming Languages.* ACM, New York, NY, USA, 7 pages. https://doi.org/00.0000/0000000.0000000

1 MOTIVATION

Histories are powerful, yet too often elusive. Consider the example[1]: a Python 3 developer prepares to merge a Pull Request from another developer on her team into the develop branch of their project. In the Pull Request she sees a method she is unfamiliar with. She performs a git log, walking through the file's history one commit at a time to better understand how and why the method came to be. However the method abruptly shows up as additions in its entirety only a few commits before the `HEAD` of the branch. Unbeknownst to the developer, the method had been moved up from a child class from that commit. The tools supposedly designed to archive code mutation have failed her; the method's true birth and early germination period in the child class are lost to the developer.

This developer would have been better served by a syntactically aware tool that is able to follow the history of a method through line ranges and between files. She would have been better served by CodeShovel. Unfortunately for her, CodeShovel, though open source, is only implemented for Java.

[1]Adapted from Felix Grund's original thesis paper on CodeShovel [5].

Project Deliverable 4, Oct 30, 2018, Vancouver, BC
© 2018 Association for Computing Machinery.
ACM ISBN 000-0-0000-0000-0/00/00…$00.00
https://doi.org/00.0000/0000000.0000000

2 BACKGROUND

This section introduces the value and workings of CodeShovel as demonstrated without a Python 3 version of the tool.

2.1 Histories

As demonstrated in previous work [5], without using tools like CodeShovel, if a developer was to find a history function level, she would have to look into the whole file that contains the function on version control.

For larger projects that have a longer life span, one method can undergo numerous changes, and these changes can be easily buried under the changes to other parts of the file that are irrelevant to that method. Some version control tools do have built-in functionality that enables users to select line-range based history. However, deleting or adding lines (comments or methods) above the selected range will cause the content within that range to shift away from the user, not to mention if the method was moved from files to files, one will have to manually traverse all the files for a complete history of changes to that method. Of course for large scale projects, traversing the history of every file that underwent change at every commit is impractical.

2.2 CodeShovel Primer

This subsection of the paper provides a brief introduction to CodeShovel, its effectiveness, and its usefulness as demonstrated by Grund in previous work [5].

2.2.1 *Inputs and Outputs.*

As described in the original paper, "CodeShovel enables developers to navigate the entire history of source code methods quickly and reliably, regardless of the transformations and refactorings the method has undergone over its lifetime, helping developers build a robust understanding of its evolution."

CodeShovel takes in "a number of arguments describing a method in a repository as input and [produces] the method's history as output". These inputs include *Repository*, *StartCommit*, *FilePath*, *MethodName* and *StartLine* that enable CodeShovel to detect changes of a method from the root of the repository tree to the *StartCommit*, and outputs the commits that had made changes to that method.

The commits in the output are typed, with the types defining what kinds of changes occurred to the method at that commit. An example output can be seen in table 1. A complete list of change types applicable to this research are defined in 4.1.1.

Table 1: Sample CodeShovel Output

SHA	Change
`b0dfa42`	Body, Parameter
`166d56c`	Moved From File
`8g607aa`	Introduced

2.2.2 Empirical Study. In an empirical study, CodeShovel's correctness was evaluated on open-source methods from Java repositories by comparing the produced results to the histories in the oracle. The results had shown a 91% accuracy validating 93 filtered methods. The median average runtime for CodeShovel to produce correct histories for this study was under 2 seconds, based on the 110,954 analyzed methods.

2.2.3 Industrial Study. An industrial study was conducted with 16 software developers, for each who had a average of 10 years programming experience and was familiar with a set of Java method histories, at a mid-sized software company in Germany. The participants were to self-select 45 methods in total, and CodeShovel was able to correctly produce 41 complete histories (91%).

According to the developers that participated in the study, scenarios that are useful to industrial developers include:

- Provenance: being able to identify the contributor(s) of a method.
- Traceability: being able to capture the changes of one method, which is not supported by IntelliJ nor `git log`.
- Onboarding: providing meaningful insight by discarding the changes that are irrelevant to that method. Unlike Git, which relies heavily on line-to-line statement comparison.
- Code understanding: providing better understanding of codes that are not familiar
- Automation: automating history-related tasks instead of manual traversal

All participants had rated the method histories; a total of 80% (13/16) rated them as at least somewhat helpful, and among these 7/16 rated as very helpful. None had rated as unhelpful, the remaining three participants held a neutral impression of the tool.

At least in its Java implementation, CodeShovel's usefulness has been measurably demonstrated by this study.

2.3 CodeShovel Methodology

To achieve method level histories, the implementation of CodeShovel follows a looping procedure with the following steps.

(1) *Parsing*. A specified file at the specified commit is parsed using a language specific parser. In its current Java only implementation, this parser is JavaParser[2] [3]. This generates an Abstract Syntax Tree (AST).

(2) *Function Finding*. Objects representing functions are generated to store metadata for every method or function in a file. These are attained by walking the AST with a Visitor[3].

(3) *Similarity Calculation*. A similarity score is assigned for every $\binom{n}{2}$ pair of methods. The algorithm returns a score based on the sum of weighted distances of a small set of function attributes. These attributes are as follows:
   (a) Body similarity: String similarity of the method body.
   (b) Name similarity: String similarity of the method name.
   (c) Parameter similarity: String similarity of parameter names and equality of parameter types.
   (d) Scope similarity: Name equality of the scope of the method, e.g. the class or parent function.
   (e) Line similarity: Distance between the start line of the two methods.

(4) *Loop?*
   (a) If the method is unchanged, specify the same file and the previous commit, and return to (1).
   (b) If there is a similar method, log a change, specify the similar method's file and the previous commit, and return to (1).
   (c) Else, return logged changes.

A graphical overview of this procedure can be seen in Figure 1.

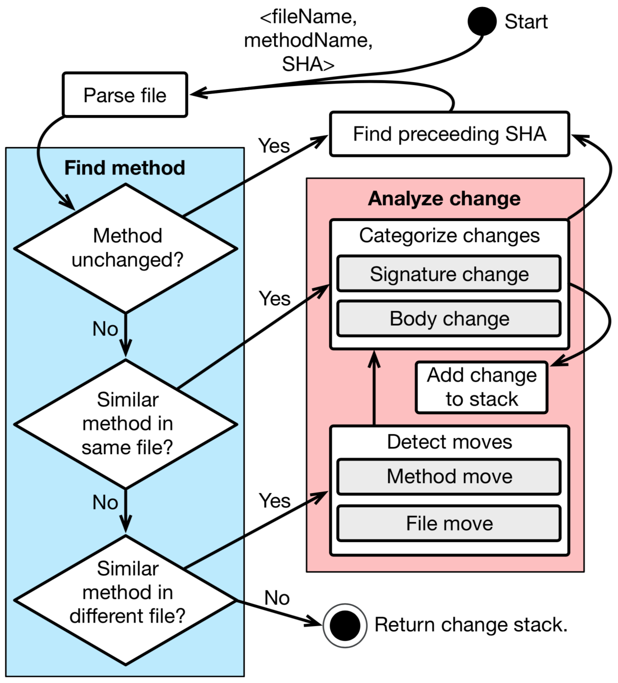


Figure 1: A high-level view of CodeShovel's methodology. Diagram by Grund [5].

3 RESEARCH AND FINDINGS

Entering Background Research phase of the project, the researchers had outlined two research questions:

- **RQ1**: Can users of CodeShovel benefit from having a Python version of the tool?
- **RQ2**: What are the use cases of CodeShovel::Python3 that are made measurably easier with the tool?

[2]JavaParser is Java based parser that builds Abstract Syntax Trees given Java source code [6]
[3]For more information on the Visitor design pattern, visit https://en.wikipedia.org/wiki/Visitor_pattern

However they found **RQ1** problematic and difficult to engage with, for the users of CodeShovel are too small of a group to generate any sample data for. So instead, **RQ1** has been replaced with:

- **RQ1.1**: Can users of Python benefit from having a Python 3 version of CodeShovel?
- **RQ1.2**: Can CodeShovel benefit from having a Python version of the tool?

Furthermore, **RQ2** cannot yet be answered as there is no Python 3 version of CodeShovel that yet exists for measurement. So until it has been built, **RQ2** has been substituted with:

- **RQ2.1**: What are the use cases of CodeShovel::Python3?

The researchers were able to satisfy both research questions through a survey of 82 of their peers, and researched into related work.

3.1 Study

To understand the use of Python and versioning tools among their peers, the researchers administered an online Qualtrics[4] survey interrogating the respondent's Python and git familiarity.

3.1.1 Design. The survey was structured around **RQ1.1**. The survey consisted of up to 11 multiple choice questions and a single all-that-applies question. The survey also asked respondents for their programming experience. Each survey took under three minutes to complete on median average.

Below are brief descriptions of the four blocks of questions presented to the respondents. The actual survey has been included in the appendix 7.1.

(1) *Block 1*: This block assesses the respondents' self perceived level of programming experience and whether or not they have used Python before.
(2) *Block 2*: This block only appears for respondents who have claimed to use Python before. It assesses which version of Python is used most commonly and how frequently the respondents use Python over other programming languages. Lastly it queries the respondents' proficiency in any other languages, to see if they are users of multiple languages.
(3) *Block 3*: This block is used to gauge familiarity with version control tools, perceived value of code history, as well as a respondent's current ability to perform the kinds of tasks that are automated by CodeShovel.
(4) *Block 4*: The final block asks only if the respondent believes that a tool like CodeShovel would be a direct improvement over their current methods of locating moments of mutation in the code.

3.1.2 Participants. The participants consisted of acquaintances of the researchers and fellow students in UBC's *CPSC 311: Definition of Programming Languages*. The survey was distributed through direct messaging on Social Media, email, and through Piazza (the main discussion forum for CPSC 311).

3.1.3 Response. The survey saw a total of 72 responses after filtering out incomplete responses. 72% of respondents self reported as computer science students, and 28% reported themselves as working software developers. Only 13% of individuals stated that they were not computer science students, software developers, or even hobbyist developers. Python proved to be extremely commonly used among respondents. 79% of all respondents had used Python at some point in their career.

3.1.4 Findings. We can immediately see that code histories are important to developers. 92% of survey respondents that identified as a computer science student, working developer or hobbyist agreed or strongly agreed to the sentiment that the history of the code that they work on is valuable.

It is worth noting, no required course in the UBC Computer Science Major (where most individuals were surveyed) teaches Python or uses it primarily for its assignments. With 79% of all respondents reporting having used Python, it is obviously a popular language in the private and industry domains. Additionally, 79% of the respondents who used Python specified that the version of Python that they used was Python 3.

38% of individuals surveyed who reported being familiar with git still did not agree that they were able to efficiently locate changes to a function given a git repository. This highlights a gap in the versioning tools that developers use to manage the history of their code.

In fact, 93% of all surveyed individuals that are familiar with Python did not disagree that a "tool that provides a list of historical changes to a function would be an improvement over [their] current approach to locating historical changes to a function."

At the very least in the perception of the researcher's peers, we have an answer to

RQ1.1: Can users of Python benefit from having a Python 3 version of CodeShovel?

As shown in the survey results, there are a large proportion of Python developers who could benefit from a Python 3 version of CodeShovel, and furthermore the sampled Python developers believe that the tool would be a benefit to their practice. All of this on top of the asserted importance of code histories demonstrates that Python users can benefit from having a Python 3 version of CodeShovel.

3.2 Other Research

In previous research [7] it has been shown that 84% of Python developers use Python 3, echoing the researchers' survey of their peers. This reaffirms the choice to specify CodeShovel's Python extension as a Python 3 extension over a Python 2 extension. Furthermore, it has been shown that 84% of surveyed Python developers use Python as their main programming language, and only 23% of the surveyed respondents cited Java usage. This brings about an answer to

RQ1.2: Can CodeShovel benefit from having a Python version of the tool?

By the metric of benefit being the number of potential users, we can see with the surprisingly small overlap between Java and Python users, a Python implementation opens up a wealth of potential users who could not engage with the tool in its purely Java based state.

To answer

RQ2.1: What are the use cases of CodeShovel::Python3?

[4]https://www.qualtrics.com

the researchers looked into Grund's work again. Five scenarios were found in Grund's survey documented in the CodeShovel thesis paper. These scenarios are listed in full in 2.2.3. As none of these scenarios are language specific, the researchers believe that they directly apply to CodeShovel::Python3.

4 DESIGN

4.1 Requirements

The bare minimum requirements that must be met such that CodeShovel ::Python3 can be considered implemented are as follows in this subsection.

4.1.1 *Functional Requirements.*

(1) **FR1**: It must parse Python 3 repositories.
(2) **FR2**: It must be able to locate specified methods in a repository when given a valid *Repository*, *StartCommit*, *FilePath*, *MethodName* and *StartLine*.
(3) **FR3**: It must locate the following change types[5] to Python 3 methods at the commit where the change occurred:
  (a) *Body Change*: The body of the method was changed.
  (b) *Method Rename*: The method has been renamed.
  (c) *Parameter Change*: The method's parameters have changed in name or arity.
  (d) *File Rename*: A refactoring operation in either the file name or the file path.
  (e) *Move From File*: A refactoring operation where a method was moved from one file to another.

4.1.2 *Non-Functional Requirements.*

(1) **NFR1**: It must correctly find the method histories of the 20 methods in the *Training Set* described in 4.3.
(2) **NFR2**: It must return results in under eight seconds on average across the methods in the *Training Set* described in 4.3.

4.2 Stretch Goals

Time permitting, the researchers aim to additionally provide the following stretch goals:

(1) **SG1**: It should be available in a web based online tool.
(2) **SG2**: It should support Python 2.

4.3 Test Plan

Following the methodology of Grund's (CodeShovel's creator), the researchers will manually construct a set of 40 unit tests that will also act as training data by which CodeShovel::Python3 will be tuned.

A total of four methods from each of the following ten open-source Python 3 repositories will be selected:

(1) `tensorflow/models`[6]
(2) `keras-team/keras`[7]
(3) `pallets/flask`[8]
(4) `scikit-learn/scikit-learn`[9]
(5) `zulip/zulip`[10]
(6) `pandas-dev/pandas`[11]
(7) `django/django`[12]
(8) `shobrook/rebound`[13]
(9) `asciinema/asciinema`[14]
(10) `jakubroztocil/httpie`[15]

For each repository, two of the four selected methods will be added to the *Training Set*, and the remaining two will be added to the *Validation Set*. Both the *Training Set* and the *Validation Set* shall contain 20 methods.

For each method, the researchers will manually annotate every change to the method back to its introduction in the repository.

Each test will be saved as JSON, with fields containing a link to the repository's origin, the SHA of the commit at which CodeShovel will begin its analysis, the filename, method name, and start line (to avoid naming collisions).

Development of CodeShovel::Python3 shall then commence using only the tests in the *Training Set* to tune its similarity function as described in 4.3.

By dividing tests into two sets, at the end of development there will already be a set of tests for measuring the generalizability of CodeShovel::Python3. The correctness of CodeShovel::Python3's evaluation of the *Validation Set* will be used as a starting point in further research during the Final Project phase of the project, where the researchers answer the research question:

- **RQ3**: Are the results returned by CodeShovel::Python3 correct?

4.4 Technical Design

As described in 2.3, CodeShovel finds method histories by Parsing Files, Finding Functions, and ranking Function Similarities in loops.

In order to successfully implement CodeShovel::Python3, there are three locations in CodeShovel that require additions.

(1) *The Python Parser*: To pull syntactic information out of the code, CodeShovel first parses the file. To do this, the researchers aim to leverage *ANTLR4*.
  (a) *ANTLR* (ANother Tool for Language Recognition) is a powerful parser generator that is used to create language-specific parsers for any programming language provided a grammar.
  Users define the grammar logic by specifying grammar rules in Extended Backus-Naur Format (EBNF) notation [10] and *ANTLR4* generates lexer and parser programs for the target language that can be used to build Abstract Syntax Trees from target language code [11].
  ANTLR4 is actively supported and available free under BSD licence [9], and is widely used in both academia and industry: at Twitter to parse billions of queries a day, in Hadoop for data warehousing and analysis, as well as to parse C++ in NetBeans IDEs [2].

[5]As defined by Grund [5].
[6]https://github.com/tensorflow/models
[7]https://github.com/keras-team/keras
[8]https://github.com/pallets/flask
[9]https://github.com/scikit-learn/scikit-learn
[10]https://github.com/zulip/zulip
[11]https://github.com/pandas-dev/pandas
[12]https://github.com/django/django
[13]https://github.com/shobrook/rebound
[14]https://github.com/asciinema/asciinema
[15]https://github.com/jakubroztocil/httpie

New in version 4 of *ANTLR* is the addition of Adaptive LL(*)[16] parsing for dynamic analysis [8], as well as changes that simplified specifying grammar, making it easier to use [10]. The grammar simplification incentivized the use of *ANTLR4* over other iterations.

(b) The researchers will begin by importing the *ANTLR4* using Maven to manage the dependency.

(c) The Python 3 grammar to be used with *ANTLR4* will be taken from the publicly available *ANTLR4* grammar repository[17].

(d) Using the Python 3 grammar, *ANTLR4* will be used to generate a Java based Python 3 parser, which generates an Abstract Syntax Tree (AST) that implements the Visitable Pattern.

(2) *The Python Function*: This is the unit at which histories are traced. Using a custom implementation of the Visitor pattern, a visitor will walk the AST, making Java objects out of Function nodes in the AST. These objects represent Python Functions and store metadata about every function in the parsed file, including the method name, body text, parameters, and location within the file. This metadata will be used when calculating the similarity of two functions.

(3) *The Function Similarity Algorithm*: Given a method that moves from one file to another, or even two methods in the same file on two different commits, they are considered the same method if they are similar and the most similar. Defining the most important attributes of similarity between two functions and their respective weight will constitute the process of tuning CodeShovel::Python3 to the *Training Set* described in 4.3.

4.5 Deliverable

The final project, having met both the Functional Requirements and Non-Functional Requirements listed in 4.1, will be delivered as a compiled Java .jar file. Using the finished project from the command line will be done with the following command:

```
java -jar codeshovel-py3.jar <OPTIONS>
```

Where `<OPTIONS>` is defined as is in the included appendix 7.2. The resulting output to both `stdout` and a specified output file will be a JSON formatted string containing the SHAs of commits that contained a change for the specified method, as well as the type of change and a diff for each.

Should **SG1** also be met, the project will be integrated with the live CodeShovel tool currently hosted on the UBC cs310 course account[18]. This would allow for online interaction with the tool, and an intuitive user interface outside of the command line.

[16]The Adaptive LL(*) parsing technology is based on LL(*) parsing which is a top-down parser working from left to right [1] with the grammar analysis carried out at runtime. This makes it stronger and faster than the previously used static grammar analysis algorithms.

[17]https://github.com/antlr/grammars-v4

[18]https://cs310.students.cs.ubc.ca/codeshovel

4.6 Foreseen Shortcomings

As CodeShovel requires a method name to build its history, Python's lambda functions will be omitted from CodeShovel::Python3's analysis. This mirrors how CodeShovel evaluates lambda functions in Java at present[19].

5 THREATS TO VALIDITY

The biggest threats to validity for these studies are external validity threats due to the limited number of participants and our open and anonymous model of survey distribution. This section includes an elaboration on those threats and the threats to construct and internal validity regarding the survey that was conducted.

5.1 External Validity

Selection of participants in the survey of peers was largely influenced by an acquaintance with the researchers. This has the potential to introduce bias and make the results irrepresentative of the researcher's actual peers. Additionally, as the survey was open and anonymous, the demographics of respondents were not controlled, allowing for potentially further skewed results.

5.2 Internal Validity

The anonymity of the survey opened the possibility for inaccurate results, as the majority of respondents are expected to be young students, who may not take the survey seriously.

5.3 Construct Validity

The survey questions interrogating the respondents knowledge of programming languages as a whole were only asked of individuals who reported using Python. Because of this, the survey may fail to capture statistics of programming language proficiency.

6 NEXT STEPS

With the delivery of the report, the researcher mark the achievement of **MS1** as defined in their project proposal. Forward, they will work toward

MS2: Derive a means to integrate a Python parser and AST with CodeShovel

This will include the tests as defined in 4.3, a working Python parser generated by *ANTLR4*, as well as a skeletal visitor for the generated AST (both described in 4.4).

[19]This behaviour was unspecified by Grund [5], however the researchers used the interactive instance of CodeShovel [4] to elicit this scenario and record its behaviour.

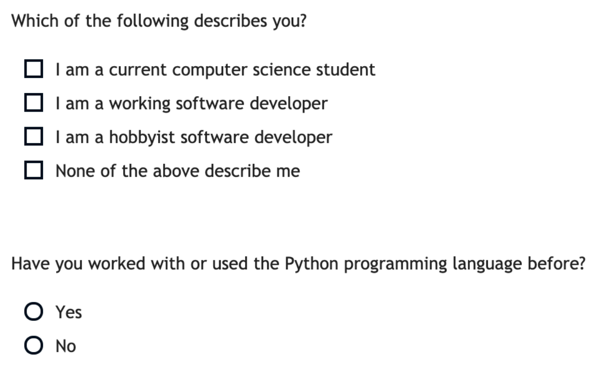
REFERENCES

[1] Karl R. Abrahamson. 2019. LL(1) Parsers. Retrieved October 29, 2019 from http://www.cs.ecu.edu/karl/5220/spr16/Notes/Top-down/LL1.html
[2] ANTLR. 2019. ANTLR. Retrieved October 17, 2019 from https://www.antlr.org
[3] ataraxie. 2019. CodeShovel. Retrieved October 17, 2019 from https://github.com/ataraxie/codeshovel
[4] codeshovel. 2019. codeshovel. Retrieved October 17, 2019 from http://cs310.students.cs.ubc.ca
[5] Felix Grund. 2019. CodeShovel : constructing robust source code history. Retrieved October 17, 2019 from https://open.library.ubc.ca/cIRcle/collections/ubctheses/24/items/1.0379647
[6] JavaParser. 2019. JavaParser - For processing Java code. Retrieved October 17, 2019 from https://javaparser.org
[7] JetBrains. 2018. Python Developers Survey 2018 Results. Retrieved October 17, 2019 from https://www.jetbrains.com/research/python-developers-survey-2018/
[8] Terrence Parr. 2014. Adaptive LL(*) parsing: the power of dynamic analysis. Retrieved October 29, 2019 from https://dl.acm.org/citation.cfm?doid=2714064.2660202
[9] Terrence Parr. 2017. antlr4/LICENSE.txt. Retrieved October 29, 2019 from https://github.com/antlr/antlr4/blob/master/LICENSE.txt
[10] Terrence Parr. 2017. The Definitive ANTLR4 Reference. Retrieved October 29, 2019 from http://lms.ui.ac.ir/public/group/90/59/01/15738_ce57.pdf
[11] Gabriele Tomassetti. 2017. The ANTLR Mega Tutorial. Retrieved October 29, 2019 from https://tomassetti.me/antlr-mega-tutorial/

7 APPENDIX

7.1 Survey

Block 1



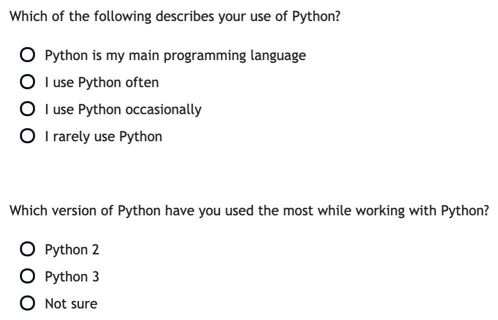
Which of the following describes you?

☐ I am a current computer science student
☐ I am a working software developer
☐ I am a hobbyist software developer
☐ None of the above describe me

Have you worked with or used the Python programming language before?

○ Yes
○ No

Block 2

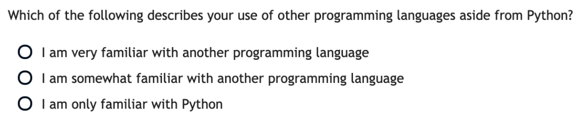


Which of the following describes your use of Python?

○ Python is my main programming language
○ I use Python often
○ I use Python occasionally
○ I rarely use Python

Which version of Python have you used the most while working with Python?

○ Python 2
○ Python 3
○ Not sure



Which of the following describes your use of other programming languages aside from Python?

○ I am very familiar with another programming language
○ I am somewhat familiar with another programming language
○ I am only familiar with Python

Block 3

How strongly do you agree with the following statement: "I am familiar with git"

○ Strongly Agree
○ Agree
○ Neither agree nor disagree
○ Disagree
○ Strongly disagree

How strongly do you agree with the following statement: "The tools that are currently available to me support isolating changes that have been made to my code base"

○ Strongly Agree
○ Agree
○ Neither agree nor disagree
○ Disagree
○ Strongly disagree

How strongly do you agree with the following statement: "I am able to efficiently find when a function was modified given a git repository"

○ Strongly Agree
○ Agree
○ Neither agree nor disagree
○ Disagree
○ Strongly disagree

How strongly do you agree with the following statement: "The history of the code I work on is valuable"

○ Strongly Agree
○ Agree
○ Neither agree nor disagree
○ Disagree
○ Strongly disagree

How strongly do you agree with the following statement: "I reference the revision history of my code during or after development"

○ Strongly Agree
○ Agree
○ Neither agree nor disagree
○ Disagree
○ Strongly disagree

Block 4

How strongly do you agree with the following statement: "A tool that provides a list of historical changes to a function would be an improvement over my current approach to locating historical changes to a function."

○ Strongly Agree
○ Agree
○ Neither agree nor disagree
○ Disagree
○ Strongly disagree

7.2 Command Line Options

Adapted from the `README.md` in the `ataraxie/codeshovel` repository [3].

```
<OPTIONS> ::= <PATH> <METHOD> <REPO> <LINE> <OPs>

<OPs> ::=
  | <OP> <OPs>

<OP> ::= <OUTFILE>
  | <REPONAME>
  | <COMMIT>

# path within the repository to
# the file containing the method
<PATH> ::= -filepath <string>

# name of the method at the starting commit
<METHOD> ::= -methodname <string>

# path to the repository on the local file system
<REPO> ::= -repopath <string>

# start line of the method within the specified file
<LINE> ::= -startline <number>

# path to the file output will be stored.
# Defaults to the current working directory.
<OUTFILE> ::= -outfile <string>

# Name of the repository. Defaults to the
# last part of the repopath (before "/.git")
<REPONAME> ::= -reponame <string>

# SHA of the commit to begin with backwards
# history traversal. Defaults to HEAD
<COMMIT> ::= -startcommit <string>
```