**1. Hypothesis and the mapping to data**

**1.1 Hypothesis**

1.1.1 The more complex the code is, the harder it is to discover its bug(s).

1.1.2 The more complex the code is, the harder it is to fix its bug(s).

1.1.3 The bugs in the code from experienced developers are hard to discover.

1.1.4 The bugs in the code from experienced developers are easy to fix.

**1.2 Data**

**1.2.1 Code level**

1.2.1.1 The indicator of code complexity

1. the nloc (lines of code without comments),
2. CCN (cyclomatic complexity number),
3. token count of functions.
4. parameter count of functions.

**1.2.2 Commit level**

1.2.2.1 The indicators reflecting whether the bug is easy to be found

1) The time from the developer's commit to the "First Find Bug" commit pointed to it

1.2.2.2 The indicators reflecting whether the bug is easy to be fixed

1) The fix distances

The fix distance is the number of commits between bug commit and fix commit, which can reflect how difficult it is to fix a discovered bug in some degree.

2) The time difference between Bug commit and Fix commit

**1.2.3 Developers level**

1.2.3.1 The indicators of developers' experience

1) The numbers of developers' commit

**1.3 Mapping**

**1.3.1 Hypothesis: “**The more complex the code is, the harder it is to discover its bug(s).”

Data 1.2.2.1: The indicators reflecting whether the bug is easy to be found

Data 1.2.1.1: The indicator of code complexity

**1.3.2 Hypothesis: “**The more complex the code is, the harder it is to fix its bug(s).”

Data 1.2.2.2: The indicators reflecting whether the bug is easy to be fixed

Data 1.2.1.1: The indicator of code complexity

**1.3.3 Hypothesis: “**The bugs in the code from experienced developers are hard to discover.”

Data 1.2.2.1: The indicators reflecting whether the bug is easy to be found

Data 1.2.1.1: The indicator of code complexity

Data 1.2.3.1: The indicators of developers' experience

**1.3.4 Hypothesis:** “The bugs in the code from experienced developers are easy to fix.”

Data 1.2.2.2: The indicators reflecting whether the bug is easy to be fixed

Data 1.2.1.1: The indicator of code complexity

Data 1.2.3.1: The indicators of developers' experience

**2. Requirements**

**2.1 Functional requirements**

Rq1: The data stated in 1.2 shall be extracted from linux-stable kernel git.

Rq1.1: The indicators that measure code complexity can be calculated using third-party package.

Rq1.2: The raw data in commit level and developer level shall be collected from git log to ensure its reliability.

Rq2: The data collected shall be cleaned.

Rq2.1: The outlier can be considered.

Rq2.2: The names of developers can be cleaned by normalizing the character sets.

Rq3: The data shall be analyzed to justify the hypothesis.

Rq3.1: The appropriate technology shall be chosen to use when analysing.

**2.2 Non-functional requirements**

Rq4: The results shall be interpretated.

Rq5: The technology selection and evaluation shall be reported.

Rq6: The concept shall be documented.

**2.3 Technical requirements**

Rq7: Python3 shall be used.

Rq8: The program shall work well in Linux system.

**3.Design**

**3.1 Data Extract**

---- extractor

---- prepare\_data ----------------------- preparation for data extracting

|--> get\_all\_author.py ----------------------- gets all authors and their commits’ hash ID

|--> get\_all\_bug\_commits.py ----------------------- gets all fix-bug commits’ hash ID

|--> author\_data.py ----------------------- extracts author level raw data

|--> commit\_data.py ----------------------- extracts commit level and code level raw data

|--> repo.py ----------------------- uses to run git command

|--> tools.py ----------------------- contains tools used in data extracting

**3.2.1 class**

3.2.1.1 Repo

3.2.1.2 CommitsFeatureExtractor

**3.2.2 Function**

3.2.2.1 get\_all\_fix\_bug\_commits: Get all the hash ID of fix-bug commits

3.2.2.2 get\_all\_author: Get all of the authors in linx-stable and the hash ID of their commits

3.2.2.3 error\_log(message:str): Record error message when data extracting

**3.2 Data Storage**

---- data

---- rdata ----------------------- raw data

---- prepare\_data

|--> all\_author@<date>.json

|--> all\_fix\_bug\_commit@<date>. json

1. date:

2. sample version:

3. random seed:

---- author

|--> [sample\_name@<sample version>](mailto:sample_name@x.x).csv

|--> [author@<sample version>](mailto:author@x.x).csv

---- code&commit

---- code\_content\_<random seed>

|--> [code\_<random seed>](mailto:code_0@x.x).csv

|--> commit\_<random seed>.csv

---- pdata ----------------------- data after preprocess

|--> author\_stat@<sample version>.csv

|--> commit\_stat\_<random seed>.csv

**3.3 Data Preprocess**

|--> preprocess.ipynb

**3.4 Data Analyze**

|--> analyze.ipynb

**4.Implemention and concept**

**4.1 Data collecting**

**4.1.1 Data Preparation**

4.1.1.1 Get all the authors’ name and hash IDs of their commits

|  |  |
| --- | --- |
| **Program** | **Store** |
| get\_all\_author.py | all\_author@<date>.json |

* **Reason**

Remove some error and unsuitable data and make a more scientific sample selecting

4.1.1.2 Get all the hash ID of Fix commits and the commits pointed by them

|  |  |
| --- | --- |
| **Program** | **Store** |
| get\_all\_bug\_commits.py | all\_fix\_bug\_commit@<date>. json |

* **Reason**

Fix commit --> Bug commit

The commit we get is bug commit, we need to find the fix commit

The time complexity is high when we use travel

we decide to sacrifice space to store all of the "Fix commit --> Bug commit"

into json(dict) format which we can use the key(bug commit) to find the value(fix commit) directly

**4.1.2 Main Process**

4.1.2.1 Get the commit level raw data

|  |  |
| --- | --- |
| **Program** | **Store** |
| commit\_data.py | commit\_<random seed>.csv |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Column Name** | **bug** | **fix** | **fix\_distance** | **find\_bug\_time**(Second) | **fix\_bug\_time**(Second) |
| **Type** | string | string | float64 | float65 | float66 |

4.1.2.2 Get the author level raw data

|  |  |
| --- | --- |
| **Program** | **Store** |
| author\_data.py | athour@<sample version>.csv |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Column Name** | **author** | **commits number** | **bug commits number** | **total fix distance** | **total find bug time** | **total fix bug time** |
| **Type** | str | float64 | float64 | float64 | float64 | float64 |

4.1.2.3Get the raw code of bug commits

|  |  |
| --- | --- |
| **Program** | **Store** |
| commit\_data.py | code\_content\_<random seed>/ (dir) |

**4.2 Data Preprocess**

**4.2.1 Data Complete**

4.2.1.1 Code Level Data

|  |  |
| --- | --- |
| **Program** | **Store** |
| lizard package | code\_<random seed>.csv |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Column Name** | **bug** | **nloc** | **CCN** | **function numbers** | **token count** | **parameter count** |
| **Type** | string | float64 | float64 | float64 | float64 | float64 |

4.2.1.2 Author Level Data

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Column Name** | **author** | **commits number** | **bug commits number** | **total fix distance** | **total find bug time** | **total fix bug time** |
| **Type** | str | float64 | float64 | float64 | float64 | float64 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Column Name** | **bug\_ratio** | **average fix distance** | **average find bug time** | **average fix bug time** |
| **Type** | float64 | float64 | float64 | float64 |

**4.2.2 Data Cleaning**

**4.2.3 Data Normalization**

**4.3 Data Analysis**

**4.3.1 Linear Regression**

4.3.1.1

4.3.1.2

4.3.1.3

4.3.1.4

**5. Conclusion**

From the analysis above, we cannot conclude any of the hypothesis reliably. There are not obvious relations between the factors.