Statistical-based Clone Detection (STCD)

He Feng
Department of Physics
fenghe@vt.edu

Liuqing Li
Department of Computer Science
liuqing@vt.edu

Outline

- Problem and Proposed Solution
- Structure of Designed Program
- Implementation Status and Test Result
- Challenges and Future Work

Problem and Proposed Solution

- Code Clone
  - Definition
    - Code fragment that is identical or similar to another
  - Common
    - X Windows System : about 19%
    - Core parts of Linux : between 15% and 25%
  - Pros
    - Improve the efficiency of a program
    - Increase readability
  - Cons
    - Low maintainability, difficult to make change
    - Increase code size

Problem and Proposed Solution

- Clone types
  - Type 1
    - Variations only in whitespace, layouts and comments
  - Type 2
    - Allows more variations in identifiers, literals and types, in addition to Type 1
  - Type 3
    - Allows further changed, added or removed statements, in addition to Type 2
  - Type 4
    - Perform the same computation but are implemented in different ways

Problem and Proposed Solution

- Techniques and Tools
  - Text-based
    - Compare whole lines to each other textually
    - Exact string match, ambiguous match
  - Token-based
    - Abstract and convert program to token sequence
  - Graph-based
    - AST, CFG, PDG matching
    - Compare leaves and sub-trees (sub-graphs)

Our Proposed Solution

- Method
  - Design and implement a Statistical-based Clone Detection(STCD) method to calculate the similarity between code fragments
  - Detect Type 1, 2, 3 code clone based on tokens
- Reliability
  - Developers won't make dramatic changes in code clone
  - Token-based detection tool have good performance
  - Include ambiguous match, introduce different weights to tokens to improve accuracy

Our Proposed Solution

- Fragment A and B
  - Use a parser to catch all the tokens
  - Categorize key tokens into types, variables, identifiers, operators, etc.
  - Accumulate the occurrences of each key token
  - Transform the fragment into a list of key token and frequency
  - Calculate the similarity between two lists
  - Set a threshold to the final output

Structure of Designed Program

Implementation Status and Test Result

- Output format:

  Clone Group # → Similarity: #

  Method Name 1 Start Line # End Line #

  Method Name 2 Start Line # End Line #

- An example:

```
Clone Group 1 --> Similarity :0.5166
       populate                67              75
testFindElementByClassName             141             153
Clone Group 2 --> Similarity :0.5119
       populate                67              75
  testAttribute               171             182
Clone Group 3 --> Similarity :0.5102
testUIComputation               85              94
testFindElementByClassName             141             153
```

An Example of Test Result (Similarity = 0.62)

```
142     public void testFindElementByClassName() throws Exception {
143         Random random = new Random();
144
145         WebElement text = driver.findElementByClassName("UIATextField");
146         int number = random.nextInt(MAXIMUM - MINIMUM + 1) + MINIMUM;
147         text.sendKeys(String.valueOf(number));
148
149         driver.findElementByClassName("UIAButton").click();
150
151         // is sum equal ?
152         WebElement sumLabel = driver.findElementByClassName("UIAStaticText");
153         assertEquals(String.valueOf(number), sumLabel.getText());
154     }
```

```
172     public void testAttribute() throws Exception {
173         Random random = new Random();
174
175         WebElement text = driver.findElement(By.xpath("//UIATextField[1]"));
176
177         int number = random.nextInt(MAXIMUM - MINIMUM + 1) + MINIMUM;
178         text.sendKeys(String.valueOf(number));
179
180         assertEquals("TextField1", text.getAttribute("name"));
181         assertEquals("TextField1", text.getAttribute("label"));
182         assertEquals(String.valueOf(number), text.getAttribute("value"));
183     }
```

Source: https://github.com/appium/sample-code

Implementation Status and Test Result

- Tested several Java programs from GitHub with LOC 200~3000
- Able to find out clone fragments and print out line numbers
- Matches well with manual examination
- Source code: https://github.com/CSCC5704

Challenges

- As a statistical-based clone detection(STCD) tool, to maintain precision, fragments need to be big enough, e.g. line ≥ 8
- Use ASTParser tool to catch the tokens, excessive time cost
- Manually set weight and threshold, need to be improved

Future Direction

- Use ambiguous match(bi-gram) to compare variable names
- Improve the tokenize process, reduce time cost
- Use machine learning to set weight and threshold
- Need test(training) data that's labeled out the clones
- Compare with other tools, CCFinder, etc.
- Develop a UI, demonstrate clone fragments by clicking

Thank you !