Interface Driven Code Clone Detection
1. Introduction

Code clones occur when developers replicate codes within or between the software repositories through copying and pasting, automatic code generation or plagiarism. A common replication that appears in large software repositories is method interface. It refers to the return type, method name and parameter types of a method that repeats exactly or similarly across the code repositories. If two methods contain the similar interface, it is very likely that those perform analogous functionalities. It indicates that these methods should be semantic or syntactic code clone to each other. So, interface similarity should have significance for detecting clones.

2. Overview of IDCCD

A light weight Interface Driven Code Clone Detection (IDCCD) technique is proposed here. This technique can detect clones by using method interface. First, source files are tokenized into method blocks and interface (i.e., keywords from method name, return type and parameter types) is extracted from method signature. An inverted index is built using that interfaces of methods and mapped into the method block tokens. For each method block, similar interfaces are queried from that index and compare the tokens of those methods with a similarity function. Finally, pairs of method blocks are reported as clones, if those satisfy a minimum similarity threshold. The experimental results are promising that yields to use interface similarities for detecting clones.

3. Experimental Dataset

BigCloneEval [1] evaluates the performance of the clone detection tool with recall and precision with manually validated clone pairs. Distribution of the clones in BigCloneEval is shown in the following figure. It contains

- 25K open source Java projects with 3 million source files and 250 MLOC.
- All types of clones such as Type-1 (T1), Type-2 (T2), Type-3 (T3) present in BigCloneEval.
- In BigCloneEval, VST3 and ST3 clones contain 90% and 70-90% syntactical similarity.
- Standard configuration parameters are set to minimum 6 lines and greater than 50 tokens with 70% similarity threshold.
- IDCCD is run in BigCloneEval’s VM with an average workstation (e.g., 3.26GHz quad-core i7, 8GB ram and 500GB drive).

4. Evaluation and Results

Recall and precision measured by BigCloneEval is represented in the following figures. It is summarized using various types of clones with state of the art tools such as NiCad [2], SourcererCC [3] and CloneWorks [4].

- IDCCD had perfect detection of T1 clones with 100% recall. For detecting T2, it achieved near perfect recall of 98%. In VST3 and ST3, it gained 96% and 81% recall similar to other tools.
- For intra-project and inter-project clones, IDCCD performed well and was able to gain 98% and 88% recall.
- Comparing to the other tools, IDCCD had the third best recall overall, with NiCad taking the lead. In case of precision, IDCCD also performed better comparing to others and got the second best precision 84%.

Since the number of similar interfaces in a code repository is smaller than the total number of interfaces, the use of query in the index reduces the number of candidate method block comparison with lower time complexity comparing to NiCad [2], SourcererCC [3] and CloneWorks [4].

5. Conclusion

Code clone detection using interface information has never been performed before. In this poster, an Interface Driven Code Clone Detection (IDCCD) approach is developed that can detect clones by using interface information (e.g., method name return and parameter types). IDCCD gained an acceptable recall and precision. Comparing the execution time of IDCCD with other tools and scalability testing are potential future directions of this work.

6. References