Clone Detection Beyond Copy&Paste

Elmar Juergens, Florian Deissenboeck, Benjamin Hummel
Institut für Informatik, Technische Universität München
{juergens,deissenb,hummelb}@in.tum.de

Abstract

We argument three positions: 1) independently developed semantically similar code is unlikely to be representationally similar, 2) existing clone detection approaches are ill-suited for detecting such similarities and 3) dynamic clone detection is a promising approach to detect semantically similar yet representationally different code.

Numerous clone detection approaches have been proposed [4]. They differ in many aspects. However, all of them use the same fundamental detection approach, namely to *statically search a suitable representation* of a program for similar parts. Independent of whether they work on text, tokens, ASTs, PDGs or models [1], they are thus limited to detection of code clones with *similar representation*.

This works well for clones created by copy&paste, since copying maintains representational similarity. Recent approaches [3, 4] can also cope acceptably well with minor differences resulting from copy&paste&modify. However, similarity in programs is not limited to copy&paste. Programmers that face the same problems can independently produce *semantically similar* solutions. But how representationally similar is semantically similar code that has been developed independently?

To better understand this question, we asked students to implement a simple specification of an email address validator. Of the 155 implementations we received, 89 were correct (compiled and passed our unit test suite). We performed a very tolerant clone detection[1] using CloneDetective [2]. Only 154 of the overall 3916 pairs of implementations were found to have at least one common clone. Hence, only 3,9% of the independently developed semantically similar code exhibited sufficient representational similarity. Due to very substantial variation between different solutions, we do not expect PDG based approaches to perform significantly better.

Although of limited transferability, these results coincide well with our experiences from industry: in several case studies of cross-project-cloning, detected clones were limited to projects with developer overlap and can thus likely be attributed to copy&paste. Hence, we strongly suspect program-representation-based clone detection approaches to be poorly suited to detect semantically similar code of independent origin.

```
public static String fillString(int length, char c) {
  char[] characters = new char[length];
  Arrays.fill(characters, c);
  return new String(characters);
}

private static String padding(int repeat, char padChar) throws ... {
  if (repeat < 0) {
    throw new IndexOutOfBoundsException("..."+repeat); }
  final char[] buf = new char[repeat];
  for (int i = 0; i < buf.length; i++) {
    buf[i] = padChar;}
  return new String(buf);
}
```

Figure 1. Examples of behavioral clones

Instead of searching for similar representation, we propose to detect such similarities by searching for *similar behavior*. While undecidable in general, we are optimistic that it can be approximated acceptably well in practice using *dynamic clone detection*. As proof of concept, we have implemented a prototypical dynamic clone detector for Java using techniques similar to random testing, with encouraging results. An example of detected semantically similar functions from CCSM commons[2] and Apache commons is depicted in Figure 1.

References

[1] F. Deissenboeck, B. Hummel, E. Juergens, B. Schätz, S. Wagner, J.-F. Girard, and S. Teuchert. Clone detection in automotive model-based development. In *ICSE '08*. ACM, 2008.

[2] E. Juergens, F. Deissenboeck, and B. Hummel. Clonedetective - A workbench for clone detection research (Tool Demo). In *ICSE '09*. IEEE, 2009. To appear.

[3] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner. Do code clones matter? In *ICSE '09*. IEEE, 2009. To appear.

[4] C. K. Roy and J. R. Cordy. A survey on software clone detection research. TR 2007-541, Queen's University, 2007.

[1] Min. clone length 5 stmts, max. edit dist. 1, aggressive normalization

[2] `conqat.cs.tum.edu/index.php/CCSM_Commons`