DebCheck: Efficient Checking for Open Source Code Clones in Software Systems

James R Cordy
Queen's University, Kingston, Canada
Email: cordy@cs.queensu.ca

Chanchal K. Roy
University of Saskatchewan, Saskatoon, Canada
Email: croy@cs.usask.ca

***Abstract*—The problem of finding code cloned from open source code in software systems is of interest both to the open source community (e.g., for GPL and other open source license enforcement) and the industrial community (e.g., to prevent GPL "contamination" of proprietary commercial software systems). The largest collection of open source software in general distribution is the collection of eight DVDs in the Debian source distribution, and checking for cross-cloning with the Debian source distribution goes a long way towards finding any possible copying from the set of all open source code in the world. The NiCad clone detector is an open source language-sensitive robust clone detector that has been shown to yield both high precision and high recall in detecting syntactically meaningful near-miss clones such as functions and blocks. Given a directory of new source code to check, DebCheck uses NiCad in its incremental mode to efficiently check the system for near-miss clones of C functions in the entire Debian source base in a few minutes on a 2 Gb home computer. The same technique can be used to check systems for cross-clones with any large source collection.**

***Keywords*-clone detection, open source, licensing, GPL**

I. Introduction

The question of whether code has been borrowed or copied from other systems is a key legal and ethical question in the development of new software systems [4]. Reuse is an efficient and practical way to rapidly get results, but can cause both technical and legal difficulties later. In particular, the question of whether GPL or other open source licensed code has been copied into a system is of strong interest to both the open source community (e.g, in order to enforce "copy-left" requirements of the GPL license and thus add to the open source catalogue), and to commercial software developers and users (e.g., in order to avoid those requirements). In a world where so much software development is done by third party suppliers, checking for open source code in a source system has become an important problem.

In this demonstration we will show our DebCheck command-line tool, which is designed to address exactly this problem. Given the root of a system's source code directories, DebCheck will extract all of the C functions embedded in C source files of the system and check every one of them for near-miss clones in the Debian open source distribution, the world's largest packaged collection of open source code. Using the incremental mode of the NiCad clone detector [1], DebCheck can check a system of a few hundred files in less than five minutes on a standard 2 Gb single processor home computer.

II. The NiCad Clone Detector

The NiCad clone detection method [2] is a hybrid method that combines language-sensitive parsing with language-independent similarity analysis to yield structurally meaningful near-miss clones. It has been shown to yield both high precision and high recall [3] in detecting near-miss intentional clones at both the function and block level.

NiCad has three main stages, *parsing, normalization,* and *comparison*. In the first stage the input sources are parsed to extract and pretty-print all syntactic fragments of a given granularity ("potential clones"), such as functions or blocks. This extraction step need be done only once for any given software system at any given granuarity. We take advantage of this property in DebCheck to avoid reprocessing the Debian open source collection when checking for clones, by extracting and pretty-printing all C functions in the collection only once.

In the second stage, extracted fragments can be normalized, filtered or abstracted before comparison. Potential clones can be consistently or uniformly renamed to remove differences between identifiers in the same roles. In this first version of DebCheck, we pretty-print to standardize all formatting and remove comments, but do no renaming or filtering of the code.

In the final stage, the extracted and normalized fragments are linewise compared using an LCS (longest common subsequence) algorithm to detect similar fragments (clones). The algorithm detects near-miss clones by allowing for linewise differences up to a given threshold. In DebCheck we use a threshold of 0.3, corresponding to up to 30% different lines between C functions in the system to be checked with those in the Debian source collection.

The NiCad method is efficiently implemented by the NiCad clone detector [1], which has two modes - whole system and incremental. In the whole system mode, it detects clones within a single version of a system or systems. In the incremental version, it takes two systems, normally a previous version and a new version of the same system, and reports only clones that cross between the two versions. In DebCheck we use incremental NiCad to check for near-miss C function clones that cross between the system to check for open source code (as the "new version"), and the entire eight DVD Debian source distribution (as the "old version").

III. DebCheck Implementation

DebCheck works by taking advantage of the separation of the extraction step identifying "potential clones" and

```
linux%  debcheck monit-4.2
Checking Debian source base for function clones in monit-4.2
Wed Feb  9 12:18:35 EST 2011
Checking for clones in Debian/disk1main
Extracted 437 functions
Found 106 clones (125 fragments, 516 pairs) in 19 clusters
Checking for clones in Debian/disk2main
Found 160 clones (175 fragments, 1338 pairs) in 15 clusters
Checking for clones in Debian/disk3main
Found 299 clones (439 fragments, 1403 pairs) in 140 clusters
Checking for clones in Debian/disk4main
Found 135 clones (151 fragments, 939 pairs) in 16 clusters
Checking for clones in Debian/disk1contrib
Found 0 clones (0 fragments, 0 pairs) in 0 clusters
Checking for clones in Debian/disk2contrib
Found 0 clones (0 fragments, 0 pairs) in 0 clusters
Checking for clones in Debian/disk3contrib
Found 0 clones (0 fragments, 0 pairs) in 0 clusters
Checking for clones in Debian/disk4contrib
Found 0 clones (0 fragments, 0 pairs) in 0 clusters
Done. Detailed log in monit-4.2/Debiantest.log
Wed Feb  9 12:27:24 EST 2011
```

Figure 1. Script of DebCheck run on monit-4.2 source

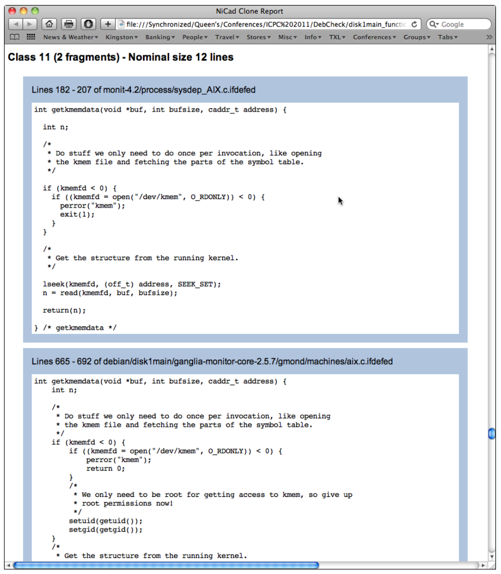
Each "cluster" is a distinct function of monit-4.2, and each "clone" is a near-miss copy of it in the Debian open source collection. The disk3main result is no surprise, since another version of monit appears in it.

the comparison step in NiCad. It begins by using the C function extractor of NiCad to extract and pretty-print all of the functions in each of the eight DVDs of the Debian source distribution. This process is itself relatively slow, since involves parsing all of the C source files, the slowest part of the NiCad process. It takes approximately 10 hours to complete the parsing and extraction of 3,596,111 functions on a desktop PC running Ubuntu Linux.

Fortunately, this mass extraction need be carried out only once per Debian source distribution. Once it is done, we can use NiCad's incremental mode to extract and compare the functions of any new C system to the C functions extracted from the Debian source distribution very efficiently. In order to maintain scalability without requiring large hardware, DebCheck runs incremental NiCad once for the extracted functions from each DVD (about 500,000 functions per DVD) rather than on the 3 million functions extracted from all of the Debian DVDs all at once.

DebCheck is a script run from the command line, and is given the root directory of the source of the system to be checked as an argument. It uses NiCad to extract and check the functions of the system for clones of the functions extracted from the first Debian DVD ("disk1main"), and then re-uses the same extracted functions to run NiCad against the functions extracted from each of the remaining seven DVDs of the Debian source distribution. Depending on the number of functions extracted, the entire process typically takes a few minutes to check a new system of a few hundred source files on a 2 Gb Linux PC.

By default DebCheck configures NiCad to its defaults, normalizing to remove commenting and spacing differences and pretty-print to standard form for comparison at function granularity allowing for a 30% near-miss threshold. Like all NiCad runs, DebCheck uses a NiCad configuration file to specify options, and other options such as block granularity,



Class 11 (2 fragments) - Nominal size 12 lines

Lines 182 - 207 of monit-4.2/process/sysdep_AIX.c.ifdefed

```
int getkmemdata(void *buf, int bufsize, caddr_t address) {

  int n;

  /*
   * Do stuff we only need to do once per invocation, like opening
   * the kmem file and fetching the parts of the symbol table.
   */

  if (kmemfd < 0) {
    if ((kmemfd = open("/dev/kmem", O_RDONLY)) < 0) {
      perror("kmem");
      exit(1);
    }
  }

  /*
   * Get the structure from the running kernel.
   */

  lseek(kmemfd, (off_t) address, SEEK_SET);
  n = read(kmemfd, buf, bufsize);

  return(n);

} /* getkmemdata */
```

Lines 665 - 692 of debian/disk1main/ganglia-monitor-core-2.5.7/gmond/machines/aix.c.ifdefed

```
int getkmemdata(void *buf, int bufsize, caddr_t address) {
    int n;

    /*
     * Do stuff we only need to do once per invocation, like opening
     * the kmem file and fetching the parts of the symbol table.
     */
    if (kmemfd < 0) {
        if ((kmemfd = open("/dev/kmem", O_RDONLY)) < 0) {
            perror("kmem");
            return 0;
        }
        /*
         * We only need to be root for getting access to kmem, so give up
         * root permissions now!
         */
        setuid(getuid());
        setgid(getgid());
    }
    /*
     * Get the structure from the running kernel.
```

Figure 2. NiCad report for clones of monit-4.2 functions in Debian source

consistent or blind renaming, filtering or abstraction of declarations or other forms, and a tighter or looser near-miss threshold can be specified to use more (or less) aggressive clone detection.

IV. Example Use

Figure 1 shows a typical run, checking the source code of *monit-4.2*, an open source application for monitoring resources on UNIX-like systems, against the eight DVDs of Debian open source code distribution for near-miss function clones. As we can see, clones are found in all four of the main distribution disks (the disk3main result is no surprise, since another version of *monit* appears in it). The NiCad web page reports (e.g., Figure 2) show us the original sources and files of the copied functions, which we find are GPL source files, indicating that *monit* has no choice but to be GPL open source (which it is).

V. Demonstration

In this demonstration we will interactively run DebCheck on real systems to demonstrate its features and performance.

References

[1] J.R. Cordy and C.K. Roy, "The NiCad clone detector", Tool demo submitted to *ICPC 2011*, Kingston, Canada, June 2011.

[2] C.K. Roy and J.R. Cordy, "NICAD: Accurate Detection of Near-miss Intentional Clones using flexible pretty-printing and code normalization", In *ICPC 2008*, pp. 172-181, Amsterdam, Netherlands, June 2008.

[3] C.K. Roy and J.R. Cordy, "A mutation / injection-based automatic framework for evaluating code clone detection tools", In *Mutation 2009*, pp. 157-166, Denver, USA, Apr. 2009.

[4] D.M. German, M. Di Penta, Y.-G. Gueheneuc and G. Antoniol, "Code siblings: Technical and legal implications of copying code between applications", In *Proc. 6th Intl. Working Conf. on Mining Software Repositories*, pp. 81-90, Vancouver, Canada, May 2009.