Singapore Management University

Institutional Knowledge at Singapore Management University

SMU Research Data | Schools

2-2011

Deckard - a tree-based, scalable, and accurate code clone detection tool (version 1.2.1)

Lingxiao JIANG
Singapore Management University, lxjiang@smu.edu.sg

Ghassan MISHERGHI
ghassanm@ucdavis.edu

Zhendong SU
su@ucdavis.edu

Glondu STEPHANE
steph@glondu.net

Follow this and additional works at: https://ink.library.smu.edu.sg/researchdata

 Part of the Computer Sciences Commons

Citation

Lingxiao, J., Misherghi, G., Zhendong, S., & Glondu, S. (2007). DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones. Paper presented at the 29th International Conference on Software Engineering. ICSE 2007. DOI: 10.1109/ICSE.2007.30

This Software is brought to you for free and open access by the Schools at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in SMU Research Data by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email cherylds@smu.edu.sg.

This is a release package of Deckard -- a tree-based, scalable, and accurate
code clone detection tool. It is also capable of reporting clone-related bugs.

```
************************************************************************
*
* LICENSE
*
************************************************************************
```

Copyright (c) 2007-2010, University of California
  Lingxiao Jiang   <lxjiang@ucdavis.edu>
  Ghassan Misherghi   <ghassanm@ucdavis.edu>
  Zhendong Su   <su@ucdavis.edu>
  Stephane Glondu   <steph@glondu.net>
All rights reserved.
Three-clause BSD licence

```
************************************************************************
*
* Version 1.2.1
* Feb 11th, 2011
*
************************************************************************
```

```
************************************************************************
*
* Installation
*
************************************************************************
```

In bash shell or cygwin, run the build script:

/path/to/src/main/build.sh

For convenience, add "src/main" into $PATH.

NOTE: Deckard's built-in parser for Java cannot handle Java 1.5 or later features, which means when Deckard processes a Java 1.5 file, it is very likely
there will be no vector generated.

NOTE: The compiled executables may not be "executable" (showing "Permission
Denied") on Windows Vista/7 due to false alarms of UAC rules (based on file
path/hash of a .exe). A simple (but may not be desirable) workaround is to run
cygwin shell with elevated privileges before invoking the above scripts. Also,
Deckard's performance may be tens of times slower when executed in cygwin than
on Linux due to slow I/O operations.

To uninstall, simply

/path/to/src/main/clean.sh

```
**********************************************************************
*
* Usage
*
**********************************************************************
```

1. For clone detection (suppose the source code of your application is in /path/to/app/src):

 - Specify the location of your source code, say /path/to/app/src.

 - Create a "config" file in /path/to/app/, following the sample "config" in
   samples/ or the template "config-sample" in scripts/clonedetect/. Make sure
   all paths are valid and the programming language is specified correctly.

 - (Optional) create other three directories in /path/to/app/ for storing
   outputs (see what's in samples/). These directories may be automatically
   created if specified in 'config'.

 - Batch mode run of clone detection (no bug detection by default):

   "/path/to/scripts/clonedetect/deckard.sh"

   An optional parameter to the script is 'clean', 'clean_all', or 'overwrite'

 - Instead of running 'deckard.sh', you may also run the scripts called in
   'deckard.sh' step-by-step by yourself:

   -- Vector generation: from where "config" is, run

      "/path/to/scripts/clonedetect/vdbgen"

      An optional parameter to the script is 'clean', 'clean_all', or 'overwrite'

   -- Vector clustering (i.e., clone detection): from where "config" is, run

      "/path/to/scripts/clonedetect/vertical-param-batch"

      An optional parameter to the script is 'clean', 'clean_all', or 'overwrite'

2. Vector generation for parts of a file:

- Identify the source file name, say /path/to/src/filename.java and the range
   [s, e] of line numbers you'd like to have a vector generated

- Run "src/main/jvecgen [options] /path/to/src/filename.java --start-line-number s --end-line-number e"
   Run "jvecgen -h" for more options. Note that different vecgen (cvecgen,
   jvecgen, phpvecgen) should be used for files in different languages.

This vecgen command will generate a vector representing the code between Line 's' and
'e' in the source file, and store the vector in "filename.java.vec" by default.

3. Detection of clone-related bugs:

- Invoke 'bugfiltering' on a clone report file with a specified language, e.g.,

   /path/to/scripts/bugdetect/bugfiltering cluster_result c > bug_result

- Optionally transform 'bug_result' to a html file for easier inspection of
   the reported potentially buggy clones in a web browser:

   /path/to/src/main/out2html bug_result > bug_result.html

- See 'deckard.sh' for how to run it in a batch mode (not enabled by default).

****
*
* What are in the package
*
****

1. Organization

The whole package is organized according to the several components in Deckard:

- Parse tree generation
   -- src/include/ : a generic interface for trees
   -- src/ptgen/ : ANTLR parser generator
   -- src/ptgen/gcc : a grammar for C (GNU C extensions) and its parse tree generator
   -- src/ptgen/java : a grammar for Java (<=1.4) and its parse tree generator
   -- src/ptgen/php5 : a grammar for php5 and its parse tree generator

- Vector generation
   -- src/vgen/treeTra/ : a generic tree traversal framework based on the
      generic tree interface in src/include, and vector generation based on tree
      traversal, mostly C++.

```
   -- src/vgen/vgrouping/ : code for vector grouping (mix of
C,C++,python,bash)

 - Vector clustering
   -- src/lsh/ : the LSH package and an interface for Deckard to use
      (src/lsh/source/enumBuckets.cpp).

 - Main entrances
   -- src/main/ptree.cc : an implementation of the tree interface
   -- src/main/main.cc : entrance for vector generation
   -- src/main/parseTreeMain.cc : entrance for parse tree dumping, can be
      useful for inspecting detected clones, bugs, and their related
parse trees
   -- src/main/bugmain.cc : entrance for bug filtering
   -- src/main/out2html.C : entrance for adding html tags into clone/bug
reports

 - Scripts gluing things together
   -- scripts/clonedetect/ : bash and python scripts
      --- deckard.sh : batch-mode clone detection
      --- vdbgen : batch-mode vector generation
      --- vertical-param-batch : batch-mode vector clustering
   -- scripts/bugdetect/ : bash and python scripts
   -- various auxiliary scripts for simple statistics

 - Others
   -- README
   -- LICENSE

2. Details about the clone/bug detection algorithms can be found in these
two
papers:

 - DECKARD: Scalable and Accurate Tree-based Detection of Code Clones, by
   Lingxiao JIANG, Ghassan MISHERGHI, Zhendong SU, and Stephane GLONDU.
In the
   proceedings of 29th International Conference on Software Engineering
(ICSE
   '07), Minneapolis, Minnesota, USA, 2007.

 - Context-Based Detection of Clone-Related Bugs, by Lingxiao JIANG,
Zhendong
   SU, and Edwin CHIU. In the proceedings of the 6th joint meeting of the
   European Software Engineering Conference and the ACM SIGSOFT Symposium
on the
   Foundations of Software Engineering (ESEC/FSE'07), Dubrovnik, Croatia,
2007.


**********************************************************************
*
* How to programmably use the vectors and the clone reports?
*
**********************************************************************

1. How to get the subtree representing each clone?
```

Each clone in the reports has a TBID and a TEID, in addition to the file name,
and line numbers. The TBID and TEID uniquely identify the IDs of the first
token and the last token in the clone from the original file (possibly
containing parsing errors). To maintain consistent counting of the IDs, you
should leave the work to "yyparse()" and Deckard's TokenCounter for how the IDs
are calculated (see TraGenMain::run() for implementation details).

The following are the main steps for getting the subtree for a clone (please
refer to "src/vgen/treeTra/token-tree-map.h" for more implementation details):

 - Given a line from the clone report file, parse it to get file name, line
   numbers, TBID, and TEID, etc. C.f. the function:

   bool parse(char * line, regex_t patterns[], int dim=ENUM_CLONE_THE_END)

 - Call the following function (which calls "yyparse()" and a token counter) to
   get a whole parse tree for the source file and the token IDs for every node:

   ParseTree* TokenTreeMap::parseFile(const char * filename)

 - Call the following function to get the smallest tree that contains all
   tokens between TBID and TEID:

   Tree* tokenRange2Tree2(std::pair<long, long> tokenrange, ParseTree* pt)

 - Then do whatever you'd like with the returned tree. Note that vectors are
   NOT generated for this tree yet. If vectors are needed, do the following:

   -- Create a new object of type TraGenMain and call "TraGenMain::run(0, 0)"
      (c.f., src/main/main.cc)

   -- Retrieve the vector for the tree:

      TreeVector* tv = TreeAccessor::get_node_vector(Tree* tree_node_pointer)

   -- If you also want some merged vectors from the child nodes of this tree,
       that would require calls to TraGenMain::run() with different parameters
       or adjust the internals of TraGenMain::run(), depending on how you want
       the vectors to be presented to you. Feel free to improve the vector

generation, both the core and its interface/APIs.

2. How to get the vector for a line or a sequence of lines from a file?

- Option 1: See above: Use "vector generation for parts of a file" with your
  scripts.

- Option 2: Given the parse tree for a file (produced by
  TokenTreeMap::parseFile() and yyparse()) and the starting and ending line
  numbers, do the following:

  -- (If not done before,) Call Deckard's vector generator on the parse tree
     through TraGenMain::run, same as above. Please refer to src/main/main.cc,
     TraGenMain::run(int startln, int endln), and
     VecGenerator::traverse(Tree* root, Tree* init).

  -- Call the following function (c.f. src/include/ptree.h, src/main/ptree.cc) to
     return the smallest tree enclosing all elements from these lines:

     Tree* ParseTree::line2Tree(int startln, int endln)

  -- Then retrieve the vector (the actual vector generation is done beforehand):

     TreeVector* tv = TreeAccessor::get_node_vector(tree_node_pointer)

Enjoy and Feedback :=)
@Deckard : Am I a clone?