Programplagiarism-detection with Marble

Jurriaan Hage (jur@cs.uu.nl)

Department of Information and Computing Sciences, Universiteit Utrecht
P.O.Box 80.089, 3508 TB Utrecht, The Netherlands

April 19, 2007

Overview

1. Introduction
2. Dealing with fraud in Utrecht
3. What can Marble do?
4. How does Marble work?
5. A small experiment
6. How does Marble do it?
7. Conclusions and related work

Plagiarism is a problem

- What is plagiarism?

  het in een scriptie of ander werkstuk gegevens of tekstgedeelten van anderen overnemen zonder bronvermelding. (Docenthandleiding Dept. Informatica)

  which translates to

  to copy information or textual passages written by others into a paper or other artifact without proper citation.

- Detecting plagiarism in computer programs is hard to do by hand:
  - discoveries tend to be accidental, based on remarkable similarities
  - only between assignments handed in in the same year
  - fewer discoveries if the group of students becomes very large
    - assignments are checked by various people
- Support is essential when students number in the hundreds, and the same assignment is given repeatedly

What is fraud?

- Everything a student does to make it impossible for the grader to make a correct estimation of his (cap)abilities
  - Obtaining an exam from the lecturer's computer account before it is given
  - Using a GSM to get answers during an exam
  - Handing in plagiarised work
  - But also includes a lot of behaviour that nobody would consider fraud!
- "Meeliftgedrag" is an example of fraud, that some might not consider to be plagiarism and some might.
- I simply regard plagiarism as a form of fraud.

Student files

- Since a few years, the exam committee keeps track of who was caught doing what
  - Avoids students performing the same trick over and over
- But we still depend on the lecturers to notify us.
- First offense: exclusion from the course for a year, and a notification in the student file
- Second offense: exclusion from all courses for at least one year, and advice to leave the program
- The first punishment is flexible, and as of this year, the second is not.
- Unflexibility in choice of punishment is not a good thing!

The protocol

- The lecturer discovers fraud
- He contacts the students and asks them to react
- A letter describing his findings and the reaction of the students are sent to the exam committee
- They consider the case and decide
  - whether fraud was committed
  - what the consequences are
  - might hear the student during a direct confrontation
  - or ask for additional information from the lecturer.
  - The student will be notified in writing, and a notice will be appended to his file.

Consider, however

- Depending on their background, students have only a vague idea of what is fraud or plagiarism
- Does translating a piece of text constitute plagiarism or fraud?
- Part of the task of the Exam Committee is to educate students as to what we regard to be fraud
    - Especially true for writing papers, less so for programming
    - Overdragen van Informaticaonderzoek

Marble

- Lends support in discovering plagiarism in (mainly Java) *programs*
  - listing pairs of file, sorted on amount of similarity
  - results in an executable script that shows these files with their similarities
  - also compares against a collection of assignments of previous years
  - is relatively fast (20,000 in 6 minutes and 20 seconds)
  - and was little work to program
  - Some of these properties are subject to change currently.
- Marble is tailored to Java, but variants made and applied to PHP, Perl and XSLT
- The same or similar ideas can be applied to written papers
  - But that is slowly ongoing work

The use of Marble on Java up to 2007

Name	*incarnations*	*assignments*	*source files*	*course*
mandelbrot	7	762	840	*IMP*
tournament	1	62	248	*INP*
animatedquicksort	5	187	1043	*GDP*
reversi	7	662	1141	*IMP*
treeroamer	2	46	335	*GDP*
monotoneframeworks	1	2	38	*APA*
petersonshortcut	5	104	578	*GDP*
sensornetwork	1	36	278	*GDP*
webshopservlets	1	47	112	*INP*
changroberts	1	40	210	*GDP*
spanningtree	4	87	411	*GDP*
prettyprint	2	95	217	*ALG*
threadedmergesort	4	78	482	*GDP*

Documented program plagiarism cases

- Eight cases since 2003.
  - Six for IMP
  - One for INP
  - One for GDP
- More have been detected, but not every lecturer involves the Exam Committee
- During IMP this year, five cases were discovered, still under consideration
- May not seem much, but beware: the use of a tool also prevents plagiarism!

Characteristics of Marble

- Compares all newly handed in assignments to
  - each other
  - to all formerly handed in assignments
  - by comparing them source file to source file
- Comparison is insensitive to
  - names of variables/identifiers
  - string, character or numerical constants
  - indentation
  - position or contents of comments
  - package structure (to some extent)
  - *order of definition of methods, inner classes and attributes*
  - how Java classes are distributed over Java source files
- Keywords are treated differently from identifiers, as are some special class and method names
- To avoid false positives, remove source code contributed by the lecturer, and remove small Java files.

How is Marble organized

- Two phases
  - the normalisation phase
    - Transforms source code into a special form suited for literal comparison
  - the detection phase
    - actually performs the comparisons and ranks the results
- Some assumptions are made about how assignments are organized:

  `halloworld/0405period1/jur/assignment2/`
- Inside the directory `assignment2` we make no assumptions.

Normalisation - an overview

- Consider each Java source file in turn
  - Anywhere inside the `assignment2` directory
- Split them up into a separate file for each class
- Normalise the names
  `assignment2/src/Hello World.java` becomes
  `assignment2/src!Hello@World.java`
- For each of these files, residing at top level, we perform actual normalisation.

Normalisation - an overview

- Consider each Java source file in turn
    - Anywhere inside the `assignment2` directory
- Split them up into a separate file for each class
- Normalise the names
  `assignment2/src/Hello World.java` becomes
  `assignment2/src!Hello@World.java`
- For each of these files, residing at top level, we perform actual normalisation.

Normalisation

Normalisation removes unessential detail from source files.
In particular, details that are easy to change without changing the behaviour of the program.

Normalisation in detail

- Remove comments and literal strings and characters
- Map identifiers to `X`, except
  - keywords (`while`), special constants (`true`), special methods (`wait`) and special types (`String`)
- We keep these special identifiers to avoid false positives
- Decimal and octal numbers $\Rightarrow N$
- Hexadecimal numbers $\Rightarrow H$
- Essentially, we map the *tokens* in the program to special uppercase letters.
- We retain all symbols like accolades, braces, arithmetic symbols.
- We try to put these tokens on separate lines

An example

The class

```
class Bliep extends Zwiep {
   String glob (int z) {
      int cnt = x;
      cnt = cnt*2;
   }
}
```

becomes

```
CLASS X EXTENDS X {
   STRING X ( INT X ){
      INT X = X;
      X = X * N;
   }
}
```

Actually...

```
CLASS
X
EXTENDS
X
{
STRING
X
(
INT
X
){
INT
X
=
X
;
X
```

Two variants

- In one variant (.nf) we are now done.
- In another variant (.nfs) we “sort” the methods, attributes and inner classes:
  - annotate each brace, { and }, with its nesting depth.
  - extract inner classes, methods and attributes based on position of {1 and }1 and semi-colons ;
  - group methods, attributes and inner classes together
  - sort within each group on the length of normalised code, then alphabetically
- If students actually moved methods around, using the .nfs version for comparison gives much better results

The detection phase

- Consider all files with extension (.nf or .nfs)
- Compare them using the standard `diff` utility

```
differentlines = diff file1 file2 | wc -l
length1 = wc -l file1
length2 = wc -l file2
measure = 100 * differentlines / (length1 + length2)
```

The generated output

- For each comparison of score below a given threshold generate one line of output

```
echo 00 59 59 && vimdiff \
   ../testsetzelf/jur/origineel/QSortObserver.java \
   ../historie/testset/versie9/QuickSortObserver.java
```

- File is sorted in ascending order (first on comparison score)
- File can be run as a script (under Linux/Unix)
  - shows each pair of files (originals) in an editor (here `vimdiff`)
  - also shows the score inside the terminal
  - typically, a lecturer goes through these until
    - discovers that the last five cases show similarities, but within limits
    - he gets fed up

The generated output

- For each comparison of score below a given threshold generate one line of output

```
echo 00 59 59 && vimdiff \
    ../testsetzelf/jur/origineel/QSortObserver.java \
    ../historie/testset/versie9/QuickSortObserver.java
```

- File is sorted in ascending order (first on comparison score)
- File can be run as a script (under Linux/Unix)
  - shows each pair of files (originals) in an editor (here `vimdiff`)
  - also shows the score inside the terminal
  - typically, a lecturer goes through these until
    - discovers that the last five cases show similarities, but within limits
    - he gets fed up

The bottom line

The lecturer must judge all suspect cases manually. Marble makes sure the most likely ones come first.

Experiment set-up

- Two student assistant, Arjen Swart and Arie Middelkoop,
- were handed somebody else's assignment for an exercise they also made themselves
- Their task: change the program as much as possible to avoid detection, but
- the program should behave in the same way and should be human readable.

Arjen Swart's nine versions

Version	*modification*	*nf score*	*nfs score*
1	*comment and layout changes*	00	00
2	*interchanged method declarations*	04	00
3	*attribute declarations exchanged*	04	00
4	*calls to GUI methods exchanged*	13	01
5	*imports changed*	13	01
6	*GUI text and colours changed*	14	01
7	*identifier names changed*	14	01
8	*rewrote some expressions*	14	02
9	*get / setmethods inlined*	14	02

- Scores are the lowest ones for a significantly large class file
- For comparison: lowest score obtained with assignments in the same year:
  - 49 for nf, 48 for nfs

Main characteristics

- Marble should be short, easy to implement and maintain
  - 440 lines of code, 220 line of comment
- Significantly flexible to change
  - *no parsing*, only work on lexical level
- Moving to Java 1.5:
  - add two new keywords to a specific array in the program (`enum` and `assert`)
  - make sure that generics do not interfere too much
- Programming language is Perl
  - ugly and obscure
  - regular expressions supported directly in the language
  - meant for report generation, text manipulation
  - familiar to me

Regular expressions

- The main tool for normalisation
- First map the whole program to lower case
- Capture tokens, replace them by an upper case character or remove them
- Uppercase parts are never matched against
- To replace all octal and decimals numerals by `N`:

```
$prog =~ s/\d+/N/gs;
```

- To remove all comments and literals in the program:

```
$prog =~ s/(\/\*(.|\n)*?\*\/)|(\/\/.*?\n)|
            (".*?")|('.*?')/ /g;
```

- Annotation of braces, retaining special identifiers: regular expressions again

Marble is not alone

- There are quite a few tools with the same goals as Marble
- Sim (Grune, Huntjens) a clone detector developed at UvA.
- Moss (Aiken et al.)
- JPLag (Malpohl et al.)
- In all these tools, complications arise from
  - finding whether two documents have significant overlap
    - And not whether two documents are the same
  - dealing with specific properties of the language
  - tools for refactoring code
- What helps us:
  - we need only one significant overlap to arouse suspicion
    - and using more tools can never hurt
  - assignments can't be obfuscated code

Summary

- Marble has been an ongoing side-track for over five years
- It has discovered a number of actual plagiarism cases
    - More than documented by the Exam Committee
- We consider plagiarism detection to be a serious affair
- An experiment was done to validate Marble
- Characteristics of the system are
    - little code, lots of documentation
    - uses token-abstraction to normalize code
    - by means of Perl's regular expression.
    - for the rest, the code is mainly administration.
    - command-line scripts with a script as output
    - running that script gives the most suspect cases first
    - using the right editor, quickly shows the lecturer whether it's plagiarism or not

Future work

- Combining Marble with the Submit system (currently in progress)
    - This is likely to change the Marble system somewhat
- A more flexible, interactive interface is needed for non-experts to use the system (currently in progress)
- Dealing with template versions of an assignment
- Dealing with written pieces of text: MicroSoft Word, Acrobat PDF, ...
- Generating plagiarism detectors from a high-level specification
- Comparison with existing systems: JPLag, Sim, Moss and others.

Questions anyone?