Universiteit Utrecht

Plagiarism detection for Java: a tool comparison

Jurriaan Hage

e-mail: jur@cs.uu.nl

homepage: http://www.cs.uu.nl/people/jur/

Joint work with Peter Rademaker and Nikè van Vugt.

Department of Information and Computing Sciences, Universiteit Utrecht

June 7, 2012

Overview

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

1. Context and motivation

Plagiarism detection

- ▶ plagiarism and fraud are taken seriously at Utrecht University
- ▶ for papers we use Ephorus, but what about programs?
- ▶ plenty of cases of program plagiarism found
- ▶ includes students working together too closely
- ▶ reasons for plagiarism: lack of programming experience and lack of time

- ▶ uneconomical
- ▶ infeasible:
  - ▶ large numbers of students every year
    - ▶ since this year 225, before that about 125
  - ▶ multiple graders
  - ▶ no new assigment every year: compare against older incarnations
- ▶ manual detection typically depends on the same grader seeing something idiosyncratic

Universiteit Utrecht

Automatic inspection

- ▶ tools only list similar pairs (ranked)
- ▶ similarity may be defined differently for tools
- ▶ in most cases: structural similarity
- ▶ comparison is approximative:
  - ▶ false positives: detected, but not real
  - ▶ false negatives: real, but escaped detection
- ▶ the teacher still needs to go through them, to decide what is real and what is not.
  - ▶ the idiosyncracies come into play again
- ▶ computer and human are nicely complementary

Universiteit Utrecht

- various tools exist, including my own
- do they work "well"?
- what are their weak spots?
- are they complementary?

Universiteit Utrecht

2. Introducing the tools

- available
- free
- suitable for Java

- Guido Malpohl and others, 1996, University of Karlsruhe
- web-service since 2005
- tokenises programs and compares with Greedy String Tiling
- getting an account may take some time

- Jurriaan Hage, University of Utrecht, 2002
- instrumental in finding quite many cases of plagiarism in Java programming courses
- two Perl scripts (444 lines of code in all)
- tokenises and uses Unix diff to perform comparison of token streams.
- special facility to deal with reorderability of methods: "sort" methods before comparison (and not)

- ▶ MOSS = Measure Of Software Similarity
- ▶ Alexander Aiken and others, Stanford, 1994
- ▶ fingerprints computed through winnowing technique
- ▶ works for all kinds of documents
    - ▶ choose different settings for different kinds of documents

- Ahtiainen and others, 2002, Helsinki University of Technology
- workings similar to JPLag
- command-line Java application, not a web-app

- ▶ Dick Grune and Matty Huntjens, 1989, VU.
- ▶ software clone detector, that can also be used for plagiarism detection.
- ▶ written in C

3. The qualitative comparison

- ▶ supported languages - besides Java
- ▶ extendability - to other languages
- ▶ how are results presented?
- ▶ usability - ease of use
- ▶ templating - discounting shared code bases
- ▶ exclusion of small files - tend to be too similar accidentally
- ▶ historical comparisons - scalable
- ▶ submission based, file based or both
- ▶ local or web-based - may programs be sent to third-parties?
- ▶ open or closed source - open $=$ adaptable, inspectable

Universiteit Utrecht

- ▶ JPlag: C#, C, C++, Scheme, natural language text
- ▶ Marble: C#, and a bit of Perl, PHP and XSLT
- ▶ **MOSS**: just about any major language
  - ▶ shows genericity of approach
- ▶ Plaggie: only Java 1.5
- ▶ Sim: C, Pascal, Modula-2, Lisp, Miranda, natural language

- JPlag: no
- Marble: adding support for C# took about 4 hours
- MOSS: yes (only by authors)
- Plaggie: no
- **Sim**: by providing specs of lexical structure

How are results presented

- **JPlag**: navigable HTML pages, clustered pairs, visual diffs
- Marble: terse line-by-line output, executable script
  - integration with submission system exists, but not in production
- MOSS: HTML with built-in diff
- Plaggie: navigable HTML
- Sim: flat text

Universiteit Utrecht

Usability

- ▶ **JPlag**: easy to use Java Web Start client
- ▶ Marble: Perl script with command line interface
- ▶ MOSS: after registration, you obtain a submission script
- ▶ Plaggie: command line interface
- ▶ Sim: command line interface, fairly usable

- ▶ JPlag: yes
- ▶ Marble: no
- ▶ MOSS: yes
- ▶ Plaggie: yes
- ▶ Sim: no

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

- ▶ JPlag: yes
- ▶ Marble: yes
- ▶ MOSS: yes
- ▶ Plaggie: no
- ▶ Sim: no

- ▶ JPlag: no
- ▶ Marble: yes
- ▶ MOSS: yes
- ▶ Plaggie: no
- ▶ Sim: yes

- JPlag: per-submission
- Marble: per-file
- **MOSS**: per-submission and per-file
- Plaggie: presentation per-submission, comparison per-file
- Sim: per-file

- ▶ JPlag: web-based
- ▶ Marble: local
- ▶ MOSS: web-based
- ▶ Plaggie: local
- ▶ Sim: local

- JPlag: closed
- Marble: open
- MOSS: closed
- Plaggie: open
- Sim: open

Universiteit Utrecht

4. Quantitively: sensitivity analysis

[Faculty of **Science**
Information and Computing Sciences]

- take a single submission
- pretend you want to plagiarise and escape detection
- To which changes are the tools most sensitive?
- Given that original program scores 100 against itself, does the transformed program score lower?
- Absolute or even relative differences mean nothing here.

- ▶ we came up with 17 different refactorings
- ▶ applied these to a single submission (five Java classes)
- ▶ we consider only the two largest files (for which the tools generally scored the best)
  - ▶ Is that fair?
- ▶ we also combined a number of refactorings and considered how this affected the scores
- ▶ baseline: how many lines have changed according to plain diff (as a percentage of the total)?
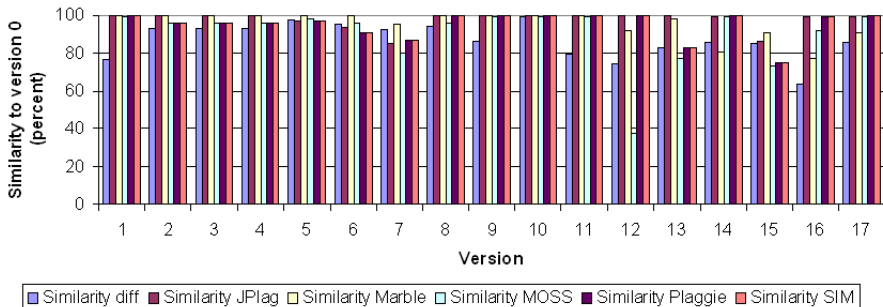
The first refactorings §4

1. comments translated
2. moved 25% of the methods
3. moved 50% of the methods
4. moved 100% of the methods
5. moved 50% of class attributes
6. moved 100% of class attributes
7. refactored GUI code
8. changed imports
9. changed GUI text and colors
10. renamed all classes
11. renamed all variables

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

12. clean up function: use this qualifier for field and method access, use declaring class for static access

13. clean up function: use modifier final where possible, use blocks for if/while/for/do, use parentheses around conditions

14. generate `hashcode` and `equals` function

15. externalize strings

16. extract inner classes

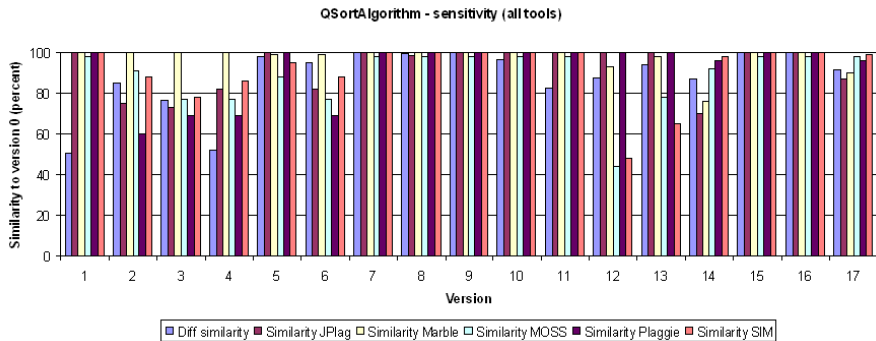17. generate getters and setters (for each attribute)

Results for a single refactoring

QSortApplet - sensitivity (all tools)

- ▶ PoAs: MOSS (12), many (15), most (7), many (16)
- ▶ reordering has little effect

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]
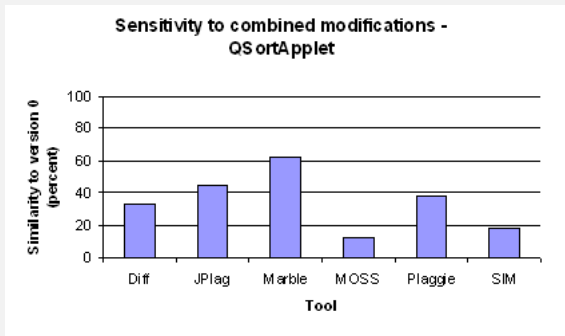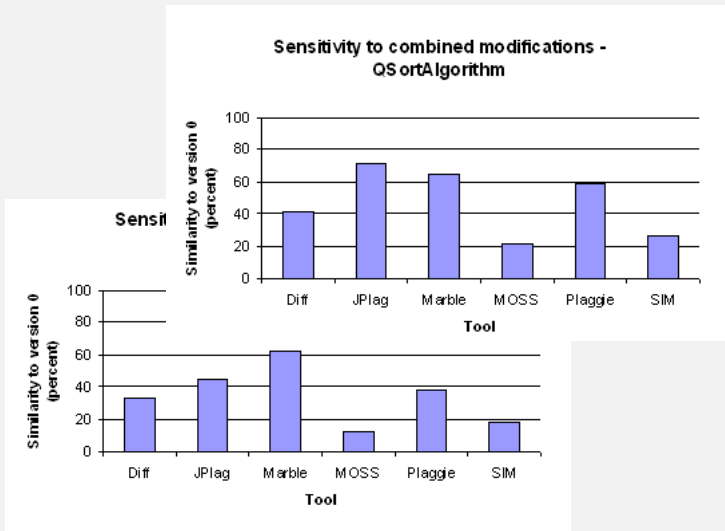
QSortAlgorithm - sensitivity (all tools)

▶ reordering has strong effect

▶ 12, 13 and 14 generally problematic (except for Plaggie)

- ▶ reorder all attributes and methods (4 and 6)
- ▶ apply all Eclipse refactorings (12 – 17)

Sensitivity to combined modifications - QSortApplet

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Sensitivity to combined modifications - QSortAlgorithm

- all tools do well for most, and badly for a few refactorings.
- differences depend on the program: sometimes certain refactorings have no effect
- except Marble all tools have a hard time with reordering of methods
- Eclipse clean-up refactorings can influence scores strongly (which is bad!)
- MOSS bad on variable renaming
- combined refactorings are much harder to deal with
  - and we could have made it worse.

5. Quantitively: top 10 comparison

- an extremely insensitive tool can be very bad: every comparison scores 100.
- normally, tools are rated by precision and recall:
    - when we kill 75 percent of the bad guys, how much collateral damage is there?
- depends on knowing who is bad and who is good
- too much manual labour for us, so we approximate

- consider top 10 file comparisons of each tool
- consider each of them manually to decide on similarity
- for bad guys in the top 10 in tool X, we hope to find these in the top 10 of all tools
- for good guys in the top 10 of X, we hope not to find it in any other top 10

- ► Mandelbrot assignment: small, typically one class, from course year 2002 up to course year 2007
- ► 913 submissions in all, with a number of known plagiarism cases in there
- ► the top-10 of the five tools generate a total of 28 different pairs (min. 10, max. 50)

- ▶ 3 self comparisons
- ▶ 5 resubmissions
- ▶ 11 false alarms
- ▶ 5 plagiarism
- ▶ 3 similar (but no plagiarism)
- ▶ 1 due to smallness

- ▶ Plaggie has many false alarms, and many real cases do not attain the top 10
- ▶ Plaggie and JPlag "failed" on uncompilable sources
- ▶ JPlag misses a plagiasram case that the others did find
- ▶ easy misses by MOSS (similar) and Sim (resubmission)
- ▶ Marble does generally well, assigning substantial scores to all plagiarism and similar cases

6. Wrapping up

- ▶ comparison of five plagiarism detection tools (for Java)
- ▶ qualitatively on an extensive list of criteria
- ▶ quantitively by means of
    - ▶ sensitivity to plagiarism masking
    - ▶ top-10 comparison between tools
- ▶ in terms of maturity of tool experience, JPlag ranks highest
- ▶ genericity leads to unspecificity (MOSS)
- ▶ except for Marbe, tools can't deal with reordering of methods
- ▶ tool need to improve to deal well with combined refactorings

- other tools: Sherlock, CodeMatch (commercial), Sid (?)
- other languages?
- making the experiment repeatable
- larger collections of programs
- other quantitative comparison criteria