Department of Computer Science

Plagiarism Policy

February 2015

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Contents

1 Introduction

This plagiarism policy is an extension of the general plagiarism policy of the University of Pretoria(UP) that is consistent the general policy. The policy of UP describes plagiarism in a broader context while this policy deals with the concept of plagiarism specifically with respect to source code written in a programming language taking software engineering principles such as collaborative development and code reuse into account. It does not override anything of the plagiarism policy of UP, it merely extends it.

Educating students about cheating, plagiarism, intellectual property rights and the consequences of academic dishonesty is an important objective of higher education institutions. These goals should be achieved without hampering software engineering techniques to enhance code quality and speed up development such as collaborative development and code reuse. When students have a deeper understanding of academic integrity, it is more likely that they will be able to develop their skills in constructive collaborative work and proper code reuse without fear.

In software engineering it is important that code be written in ways to maximise code reuse. To apply this, software engineers should know where to find reusable code and how to incorporate it into their software. The quality of software can be improved through the efficient use of the expertise of experienced programmers. Software reuse has the potential to increase productivity by reducing development time [18, 8].

Section 2 defines code plagiarism while Section 3 describes ways to work with existing code that are closely related to plagiarism.

Students are required to illustrate their honesty when submitting code for assessment. This is achieved through acknowledgement of resources along with the description of their own contribution to the code base that is submitted for assessment. The required documentation is discussed in more detail in Sections 4, 5 and 6 while Section 7 addresses collaboration. Section 8 stipulates the consequences if it is found that plagiarised work was submitted.

Adherence to these specifications is intended to allow students to be open and honest about their work and is likely to result in a deeper understanding of academic integrity. It also creates an opportunity for students to apply their knowledge to improve the quality of the code they submit in terms of compliance with our coding standards. These coding standards are fairly generic and quite comprehensive. It includes guidelines to improve various quality attributes such as readability, clarity, reliability, user friendliness, flexibility, portability, effectiveness and efficiency [12].

2 Plagiarism

The use of code that originates from sources such as textbooks, lecture notes, online tutorials, on-line code repositories or even peers does not necessarily constitute plagiarism.

Code plagiarism is defined as using someone else's code **and passing it off as your own**. This is irrespective if the code is copied from someone else's work without consent, copied from a friend's work (even with consent) or copied from other sources such as textbooks or the Internet. In some cases reusing your own code can be seen as a from of plagiarism called self-plagiarism [2].

Note that the use of someone else's code *per sé* is **not** condemned. Using other peoples work and ideas is fundamental to learning and should be encouraged [5]. The difference between honest code reuse and code plagiarism lies in the integrity of the person reusing the code. In the case of plagiarism the person submitting the code pretends that the work is his/her own.

It is dishonest to reuse code with the criminal intention to get marks (or money or fame) for work that was done by someone else. It is in violation of the value based goals of the University of Pretoria.

Programming assignments are primarily given to create an opportunity for students to apply a specific concept in order to gain experience in the application of the concept. Marks that are awarded for an assignment should never be seen as a goal in itself. Marks should be applied merely to gauge the level of your understanding. Students that understand and subscribe to this reason for doing assignments will also understand the futility of submitting work that is not the outcome of their own learning experience.

3 Working with existing code

3.1 Hacking

Many students seem to be under the impression that when they alter existing code to suit a specific need that is very similar to the intention of the original code that they are reusing the code. As is explained in Section 3.3, this is not the case. Doing this is sometimes called *hacking*[1]. In the sense the word *hack* is used here, it describes the actions of an inexperienced programmer who changes sections of code without proper understanding in a trial-and-error fashion. This is probably the most ineffective way to learn how to program.

The following advice was given by Wayne Packer on the COS132 course discussion board in 2013 to students to explain how one can honestly learn by using existing code. He also points out that dishonesty is futile.

> *Study the code and understand what it does. Then code your own solution; this makes you better at coding and eliminates the need to plagiarise. If you just copy code all the time what do you learn?*

If you would hack an assignment that was created by someone else, and submit the altered code as your own, it is very likely to be deemed plagiarised.

[1] Hacking is more often used to refer to an action to secretly getting access to a computer system

3.2 Refactoring

Fowler [3] defines refactoring as follows:

> *Refactoring is a controlled technique for improving the design of an existing code base. Its essence is applying a series of small behavior-preserving transformations, each of which "too small to be worth doing". However the cumulative effect of each of these transformations is quite significant. By doing them in small steps you reduce the risk of introducing errors. You also avoid having the system broken while you are carrying out the restructuring - which allows you to gradually refactor a system over an extended period of time*

In large legacy systems refactoring may be applied to gradually improve the design of the system instead of replacing it. Refactoring can also be applied on smaller scale to improve the readability, clarity, reliability, user friendliness, flexibility, portability, effectiveness and efficiency of small portions of code. Sadly simple refactorings are often used by dishonest students in an attempt to hide the fact that their code is plagiarised.

Martins *et al.* [10] categorised refactorings that are commonly applied by students to avoid the detection of their copied code by plagiarism detection software. Table 1 was compiled using Martin's categorisation as basis. It defines seven refactorings applicable to the kind of programs that students are expected to write for introductory programming assignments.

Code to which refactorings such as the ones listed in Table 1 was applied, is deemed plagiarised unless the original source is properly cited and is accompanied by a valid change log justifying each of the refactorings in terms of an improvement as suggested in our coding standards.

3.3 Code reuse

Code reuse, also called software reuse, is the use of existing software, or software knowledge, to build new software [4]. The key idea in code reuse is that parts of a computer program written at one time can be or should be used in the construction of other programs written at a later time [20]. To this end programmers define templates, functions, and procedures that are generic enough to be used in new situations without the need to change the original code. These are often published in public or propriatry software libraries.

The C++ language was designed to encourage reuse [16]. In your very first program you reused code that is in the **iostream** library. You simply include the library and use the code that is in the libary as if it native to your source code. You will soon learn how to use more libraries and later how to build your own.

Software reuse include the design and implementation of software libraries to extend the code base of reusable code, domain engineering methods to promote the

Table 1: Simple refactorings

Number	Type	Description
1	Change comments	Editing, removing and/or adding comments.
2	Renaming of identifiers	variable, constant and function names are changed.
3	Change scope	Move local variables to global scope and *vice versa*. Also on deeper levels of programming blocks, such as moving a variable that was local to the body of a loop, out of the loop and *vice versa*
4	Use alternative	Using a different construct to achieve the same results, for example; using a different mathematical expression in a conditional statement, using a different loop type, using a different data structure or data type, using an alternative function for I/O, etc.
5	Change order	Swop statements that may be executed or listed in a different order for example variable declarations, parameter lists, function definitions, etc.
6	Change modules	The modular design of a program is changed. Moving the body of a function to the position where the function was called and moving a block of code into the body of a function as well as moving functions from one class to another are examples.
7	Add or remove variables or constants	Elimination of variables and/or constants as well as the introduction of constants or helper variables.

development and deployment of reusable code and enhance the reusability of code in systems, as well as the development of tools to support software reuse.

Proper code reuse avoids “copy and paste programming”. The following are basic forms of code reuse within a small program. The code that is reused is written in a general fashion in order to be able to reuse it in different specific ways.

- If the same (or very similar) code appears in more than one place in the program, put it in a function that can be called more than once.
- If a number of consecutive lines of code are the same (or very similar) find a way to specify the operation performed by the code using a loop structure.

It should be clear that hacked code (see Section 3.1) is not a form of proper code reuse.

3.4 When is code plagiarised and when not?

When only looking at the end product it is close to impossible to distinguish whether the code is plagiarised or not. Only the person who created the submitted version of the code will know if the work that was done can be classified as *deliberate practise* or as *an action to get marks with minimal effort*. The latter borders on plagiarism while the former is an honest action of learning.

Many students seem to be under the impression that as long as the work that is submitted can not be clearly identified as plagiarism, then it is acceptable. There may be students who feel that in certain situations it may be justified to copy code or hack code with the wrong intentions. This is in sharp contrast with the value based attitude towards learning we hope to instill.

For this reason students are required to write documentation in their code files to provide additional information to enable assessors to determine the truth. We are not oblivious to the fact that students may lie in their documentation to deceive the assessor – at least the onus is on them. Students who lie and cheat are unlikely to succeed.

4 Preamble

A preamble should be included in each file. It is a comment block at the top of the file that contains the following information

- The name(s) and student number(s) of the author(s).
- The date of last edit.
- A short paragraph describing the purpose of the code in the file.
- If applicable, a description of how this file relates to other files in the submission with inclusion of the file names of related files.
- A declaration.

Use the proper tags for author and date as specified by the documentation generator of your choice. We recommend Javadoc[2] for Java code and Doxygen[3] when writing C or C++ code. Most of the tags for these documentation generators are the same.

If you completed the assignment on your own without any assistance the declaration should state that the content of the file is the author's own work.

If you received guidance from someone or used resources such as on-line tutorials or your textbook you should mention this in the declaration. These resources should be properly cited as described in Section 5.

[2] http://www.oracle.com/technetwork/articles/java/index-jsp-135444.html
[3] http://www.stack.nl/~dimitri/doxygen/index.html

If the file contains code that is a variation of existing code, the declaration should state this clearly. The original resource should be properly cited as described in Section 5. In addition the submission must be accompanied by a file containing a change log. The declaration must state the file name of the required change log. The requirements for the change log are described in Section 6.

If you reuse code libraries other than those in the standard template library, you have to cite the URL where the libraries can be found or include the libraries in your submission.

5 Citation of resources

The aim of citing resources is to provide enough information for the readers to locate the original resource. Preferably citations should be given in a standard style like APA[4] or Harvard[5]. The following specifies the detail that must to be included for each of the different types of resources:

Person The name of the person and the role of the person. The role specifies the relation between you and the person for example: tutor, friend, dad, etc. If the person is someone in the class, his/her student number is required. Also give the specific date(s) during which you were assisted by this person.

Own code base Detail about when, where and why the code was originally written. Ideally the code should be made publicly available, for example on StackOverflow, Codeshare, your blog or your facebook page.

On-line Resource The Author, the agency, the year of publication, the title of the page, and the complete URL. If unknown the author or the agency may be omitted, but you may not omit both.

Book The title of the book, the author, year of publication, publisher, and the page numbers.

Journal Article The title of the article, the author, year of publication, journal name, volume number, issue number, and page numbers.

Conference Paper The title of the paper, the author, year of publication, conference name and abbreviation, and page numbers.

If some of the required facts are unknown, it should be stated explicitly. For example if the year of publication is unknown you should state “year: unknown” in the citation.

[4] www.apastyle.org/

[5] openjournals.net/files/Ref/HARVARD2009%20Reference%20guide.pdf

6 Change log

Table 2: Code Changes

Number	Type	Description
1	Refactor	Apply one or more of the refactorings described in Table 1 to improve the quality of the code in terms of the guidelines specified in our coding standards
2	Add functionality	If the source need to be extended to suit the requirements of an assignment, new functionality is inserted.
3	Remove functionality	The source code contain code that is not required for the assignment. Redundant functionality has to be removed.

In order to be able to evaluate the level of understanding achieved by a student who has applied some changes to existing code, it is expected that a change log is included.

The aim of the log is to be able to identify the portions of the code and algorithms that are original. It helps the assessor to gauge the amount and quality of the programming work that was done by the student who submits the code. Instead of determining the value of the code base as a whole and awarding a mark for the correctness and quality of the complete code base, the assessor will assess only the student's contribution to the code base.

A chance log should describe the changes that were made and give a reason for applying each of the alterations. Possible changes are described in Table 2.

The following are examples of reasons that may be offered for applying some of the refactorings mentioned in Table 1:

- Add or reword comments to enhance clarity.
- Change identifier names to be more descriptive of their purpose.
- Remove redundant comments as the code is clear enough without.
- Change the scope of a global variable to be local to the function to improve modularity.
- Simplified an expression to remove complexity and/or improve performance.
- Replace a `for`-loop with an equivalent `while`-loop to enhance the elegance of the solution.
- Change the order of function prototypes to match the logical order in which the functions are implemented in the program to enhance clarity of the code.

The change log should be specified in a separate file. The file name of the change log should match the file name specified for it in the preamble of the submitted code.

For each of the changes:

- Describe the detail of the change. For example when renaming identifiers, it is required that a list stating the old name and the new name for each renamed identifier is given.
- Locate the change — i.e. Specify the line numbers in the code where the changed code can be found.
- Justify the change — i.e. Link the change to adherence to an item in our coding standards or give full detail why the change was needed.

7 Working together or assisting a class mate

When students are required to work in teams or pairs, it is expected that the team submit only one code base for assessment. The names and student numbers of all participants should appear in the preamble. The same requirements as when submitting an individual task hold. In this case an extended declaration is required. The declaration should describe the participation of each of the students in the team or pair.

If an assignment is meant to be completed individually, working in teams or pairs may be problematic, especially when the contribution of the individuals who participate are not equal. It is, however, not prohibited. We encourage our students to help one another. The value of cooperative learning is widely accepted [7, 15, 17]. Rohani [13] reports that students may experience a feeling of greater competence when instructed by a peer as opposed to when instructed by a teacher. Moreover, pair programming is acknowledged to have potential to enhance learning [11, 6].

If you received assistance while working on an individual assignment or had worked together towards a solution for a task that is assessed on an individual basis, all students who were involved should acknowledge the involvement of one another in the their preambles. An extended declaration similar to the one required for group projects is required. The declaration should stipulate the level of cooperation and describe the contributions of the other involved students and clearly outline the contribution of the individual who submits the code as described in Section 4.

8 Evaluation

When a submission contains code that originates from somewhere else, the submission is evaluated based on the quantity and quality of the reported changes

that were made at the discretion of the assessor. If no changes were made or the reasons for the changes are weak, a zero mark may be awarded.

When students acknowledge that they have worked together, it is to the discretion of the assessor to award marks to each of the individuals based on his/her opinion to what extent the individuals have learnt while collaborating and to what extent the individuals understand the code they submitted.

Disciplinary action will be taken against a student who submit code that was originally created by someone else without acknowledging the original code or without logging all the changes.

Student may be prosecuted for plagiarism if attempts to deliberately deceive is observed. This may include submitting unoriginal code and neglecting to cite the original source or not logging all alterations. Applying code changes that appear to be geared towards to fooling anti-plagiarism software are also frowned upon.

9 Conclusion

Wagner [19] states that it is important to eradicate plagiarism one way or the other because it undermines learning. Shaw *et al.* [14] is convinced that the prevention of cheating is better than to expend effort in establishing that cheating has occurred and taking action against students who did indeed plagiarise. Prevention entails reducing opportunities to cheat, minimising temptations to cheat as well as fulfilling our educational responsibility to create a learning culture and instil ethical values. Kourie [9] emphasises that curricula should aim to transmit an agreed-upon value-system, using skills training as a context to achieve this. Lecturers should constantly reinforce values relating to professionalism, responsibility, ethics, etc. This can be done by maximising opportunities to assimilate these values. The core message that should be brought to our students is aptly described in the words of the Hollywood screenwriter Scott Alexander:

> *All good is hard. All evil is easy. Dying, losing, cheating, and mediocrity is easy. Stay away from easy* [1].

References

[1] Alexander S (n.d.) ThinkExist – Scott Alexander quotes, http://en.thinkexist.com/quotation/all_good_is_hard-all_evil_is_easy-dying-losing/324937.html. [Online] accessed 2014-03-20.

[2] Collberg C and Kobourov S (2005) Self-plagiarism in Computer Science, *Communications of the ACM*, 48(4):88–94.

[3] Fowler M (1999) *Refactoring: Improving the Design of Existing Code*, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

[4] Frakes W and Kang K (2005) Software reuse research: status and future, *IEEE Transactions on Software Engineering*, 31(7):529–536.

[5] Gibson JP (2009) Software Reuse and Plagiarism: A Code of Practice, in: *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '09, 55–59, New York, NY, USA: ACM, URL http://0-doi.acm.org.innopac.up.ac.za/10.1145/1562877.1562900.

[6] Govender D and Govender T (2014) Using a Collaborative Learning Technique as a Pedagogic Intervention for the Effective Teaching and Learning of a Programming Course, *Mediterranean Journal of Social Sciences*, 5:1077–1086.

[7] Johnson DW and Johnson RT (1987) *Learning together and alone: Cooperative, competitive, and individualistic learning*, Englewood Cliffs, NJ, US: Prentice-Hall, Inc, 2nd edn.

[8] Kotov V (2011) Reuse in Software Development Organizations in Latvia, *Scientific Journal of Riga Technical University, Computer Sciences*, 41(1):90–96.

[9] Kourie DG (2004) Values in an Introductory Computer Science Curriculum, Poster presentation, *9th International Academic Forum*, Barcelona.

[10] Martins VT, Fonte D, Henriques PR and da Cruz D (2014) Plagiarism Detection: A Tool Survey and Comparison, in: Maria João Varanda Pereira ASo José Paulo Leal (Ed.) 3^{rd} *Simposium on Languages, Applications and Technologies (SLATE'14)*, 143–158, OASICS Schloss Dagstuhl.

[11] McDowell C, Werner L, Bullock H and Fernald J (2002) The effects of pair-programming on performance in an introductory programming course, in: *ACM SIGCSE Bulletin*, 34-1, 38–42, ACM.

[12] Pieterse V (2008) Reflections on coding standards in tertiary Computer Science education : festschrift : dedicated to Derrick Kourie, *South African Computer Journal*, 41:29–37.

[13] Rohani A (2014) Cooperative Learning, Motivational Effects, and Students Characteristics on Academic Performance, *Applied Psychology*, 8:20–29.

[14] Shaw M, Jones A, Knueven P, McDermott J, Miller P and Notkin D (1980) Cheating Policy in a Computer Science Department, *SIGCSE Bulletin*, 12(2):72–76, URL http://0-doi.acm.org.innopac.up.ac.za/10.1145/989253.1165253.

[15] Slavin R (2011) Cooperative learning, in: Aukrust V (Ed.) *Learning and Cognition in Education*, 160–166, Boston: Elsevier Academic Press.

[16] Stroustrup B (1996) Language-technical Aspects of Reuse, in: *Proceedings of the 4th International Conference on Software Reuse*, ICSR '96, 11–, Washington, DC, USA: IEEE Computer Society, URL http://0-dl.acm.org.innopac.up.ac.za/citation.cfm?id=525583.853465.

[17] Tran VD (2014) The Effects of Cooperative Learning on the Academic Achievement and Knowledge Retention, *International Journal of Higher Education*, 3(2):131–140.

[18] Tung YH, Chuang CJ and Shan HL (2014) A framework of code reuse in open source software, in: *Network Operations and Management Symposium (APNOMS), 2014 16th Asia-Pacific*, 1–6.

[19] Wagner NR (2004) Plagiarism by student programmers, http://www.cs.utsa.edu/~wagner/pubs/plagiarism.html. [Online: accessed 12-January-2013].

[20] Wikipedia (2014) Code reuse — Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/w/index.php?title=Code_reuse&oldid=633151701. [Online; accessed 15-February-2015].