# Software Plagiarism in Undergraduate Programming Classes

Samuel Abraham
sam@sienaheights.edu
Department of CIS
Siena Heights University
Adrian, MI 49221

Gregg Milligan II
milligangregg@yahoo.com
Department of CIS (Student)
Siena Heights University
Adrian, MI 49221

## Abstract

In recent years software plagiarism became a big concern for many programming instructors. The goal of this paper is to give its reader a better understanding of software plagiarism in undergraduate programming classes, the extent in which plagiarism is used in undergraduate classes, various methods used by the students to plagiarize, and some deterrents used by institutions to combat the problem of plagiarism. The paper also discusses some consequences faced by the students for their plagiarism activity.

**Key Words:** Cryptomnesia, Ghost phenomenon, Intellectual property, Natural language plagiarism, Plagiarism, Plagiarism detection, Plagiarism detection software , Plagiarism deterrents, Plagiarism methods, Program specification, Program units, Software plagiarism.

## 1. WHAT IS SOFTWARE PLAGIARISM?

In general, plagiarism is claiming, part or all of a body of intellectual property (written, cinematic or audible), was created completely independent of all other intellectual works when, in fact, it was copied or based from one or more separate intellectual works. Plagiarism includes paraphrasing material without giving credit to the author(s) of the paraphrased material. Software plagiarism is a form of plagiarism, which specifically applies to computer program source code. This type of plagiarism is becoming a big concern for institutions of higher learning.

## 2. PLAGIARISM & ACADEMIC INSTITUTIONS

In academic institutions, the most common reason for concern is that software plagiarism, performed by students, prevents these students from achieving the main goal of writing code: To learn how to write and interpret software code (Clough, 2000). For example, if a student plagiarizes source code, he or she does not benefit from the experience of writing code, which is the most effective way of learning and understanding this subject, when paired with the study of text-book or in-class instruction. Instead, a plagiarizing student copies part or all of a program from an outside source without fully understanding the copied material. The student may also arbitrarily alter the plagiarized code, in order to make it appear unique, but this still does not help the student fully understand the code. The core problem with students' software plagiarism is the fact that it impedes learning. Another reason for concern is that such plagiarism denies credit for those who deserve credit for their intellectual work.

## 3. HOW COMMON IS SOFTWARE PLAGIARISM?

Statistics on the frequency of plagiarism within universities are rare. Bowyer & Hall (2001) speculate that this is because many universities are reluctant to keep comprehensive records of plagiarism or to release these records because they are afraid that it will violate the privacy of the students. Another reason may be that universities are hesitant to release records, which may reflect poorly upon the institution's reputation. In a survey of 242 students at Duke University, nine percent (22 students) admitted to plagiarizing a programming assignment, "at least once." Of all students surveyed, 40 percent deemed software plagiarism as a "serious" problem (Bowyer & Hall, 2001). One flaw of this survey is that "serious" is not clearly defined. For example, "serious" could be interpreted as the seriousness of the problem where fewer instances of plagiarism result in a less serious problem than many instances of plagiarism. Alternately, "serious" could be interpreted as applying to the ethical weight of plagiarism. For example, a student may consider plagiarism not to be "serious" and, therefore, consider it to be an appropriate solution to completing and submitting assignments. An additional flaw of surveys, in general, is that in many cases, students may be hesitant to answer honestly concerning sensitive topics, such as plagiarism, even if the survey results are promised to be recorded anonymously.

The best methods for trying to pinpoint the prevalence of software plagiarism usually include careful personal observation in a position that works directly with students and their assignments, such as a tutor or, better still, the professor or assistant who grades the assignments and is mainly tasked with the detection of plagiarism. A tutor may only be able to infer about the prevalence of plagiarism since he or she usually does not grade the submitted assignments. However, experience as a tutor occasionally reveals suspected or blatant cases of plagiarism. For example, after a student receives help with a tutor and develops a good understanding of the assignment, his or her friends, who are still struggling with the assignment, may receive assignment solutions from the friend student without putting in the effort to develop a good understanding. This same situation also sometimes occurs when more than one student is being tutored at the same time and one student manages to develop an understanding more rapidly.

One inference is that software plagiarism is *relatively* more prevalent than "natural language" plagiarism (book reports, papers, etc); although fewer students write software code compared to those who write papers, it is inferred that a higher percentage of those who write code have plagiarized. The primary reason for this inference is that it is more difficult for most university students to write a sufficiently working program than to write a sufficiently intelligible paragraph. Programming is less forgiving when it comes to grammatical rules and less predictable in the logical meaning of its output. Because of this, it is inferred that more students are tempted to plagiarize software code and, therefore, end up plagiarizing.

Another factor, however, may suggest the opposite of this initial inference: Most programming students have chosen Computer Programming as their main focus of study. Therefore these students are likely to take computer programming seriously and are therefore less likely to plagiarize software code. In addition to this, there are only a few required programming classes in their chosen field of study compared to the greater number of required classes that involve writing. These two factors suggest that software plagiarism may be less prevalent than other types of plagiarism.

In one study that examined natural language plagiarism, 449 students were surveyed, 32 percent or "142" students responded that they never plagiarized. The other 68 percent or "307" students admitted to plagiarizing "through either traditional and/or Internet means" (Lester, Chaky, Diekhoff & George M, 2002). Although any specific conclusion cannot be made from the comparison between the software and natural language plagiarism study, it is evident that a higher ratio of students plagiarized natural language assignments compared to the ratio of students that plagiarized software assignments in the first study. These two studies cannot be accurately compared because of the difference in the approach they have taken to conduct the study.

## 4. PLAGIARISM METHODS

Students use different methods to plagiarize their code and a good understanding of these methods are essential to identify plagiarism and then to respond to these activities. The following list contains different factors that can be used to classify plagiarism methods:

**Who is being plagiarized**?

- How many sources are being plagiarized? For example, is the student plagiarizing content from one or more sources?
- Are they willing participants? For example, did the author(s) of the plagiarized code give this code to student(s) knowing that it would be plagiarized or did the student(s) plagiarize code without the author's knowledge? This will determine whether the author is penalized in response to the act of plagiarism. For example, like many other universities, the University of Western Australia differentiates between knowing and unknowing accomplices in plagiarism within its Computer Science and Software Engineering written policy (U of WA, 2003).
- What is their affiliation with the university? For example, is the author of the plagiarized code a graduate or an undergraduate student within the same class as the plagiarizing student? This may determine whether the author is penalized in response to the act of plagiarism (if collusion with the plagiarist occurred) and whether the plagiarism can be detected by comparing the plagiarized assignment to other assignments within the same class. For example, the "ghost phenomenon" refers to a situation where the original work is unavailable to a course instructor, making it more difficult to detect plagiarized work by comparing it to its original, remote or "ghost" source (Bowyer & Hall, 2001)

**Disguising the plagiarized work**? The following list outlines the common modifications plagiarists generally apply to plagiarized material. This list ranges from minimal modifications to the most drastic modifications, which are often the most difficult to manually detect as plagiarism. Examples of modifications, mentioned in the list, have been included to help better explain these different levels of modification:

- No modification. For example, a simple cut-and paste of program code (Bowyer & Hall, 2001).
- Comment modification (Clough, 2000). For example, deleting comments, adding comments or changing the wording of comments.
- Data type modification (Clough, 2000). For example, changing a floating-point number to a double data type.
- Text format modification. For example, getting rid of or adding indents within the code and condensing statements onto fewer lines. This can drastically change the look of the source code if it is examined only briefly.
- Selection structure modification (Clough, 2000). For example, changing a nested If statement to more than one separate If statements or converting a Switch statement to an If Else statement.
- Identifier modification (Clough, 2000). For example, changing the names / order of programmer-defined variables and functions.
- Adding superfluous statements / variables (Clough, 2000). For example, including variables or coded statements within the

program that are not needed or used for processing.
    - Mixing both plagiarized and unique statements in the same program (Clough, 2000). For example, writing part of the program, but then copy and pasting the other part.
    - Breaking logical units into smaller units, creating more logical units, or combining logical units. For example, converting one plagiarized function into two or more smaller functions. This is a more advanced type of plagiarism that requires sufficient knowledge of programming. As plagiarism techniques become more advanced, it becomes more logical to direct efforts towards writing legitimate code rather than plagiarizing.

**Why do they plagiarize?**
Some reasons why students plagiarize include:

- Seeking the easiest or most "economical" route. This stems from the perception that it is more beneficial to breeze through assignments than to learn from them. In order to change this perception, an instructor can stress the fact that the purpose of assignments is to benefit students by helping them develop knowledge and giving them practice that can lead to success, self-sufficiency and gaining important skills (Harris, 2004). Alternately, using plagiarism to breeze through assignments cheats the plagiarist of such benefits.
- Lack of interest and low priority. Low priority assignments that seem unappealing are usually neglected until plagiarism becomes a desperate option to complete the assignment on time. Instructors can reduce instances of such plagiarism by creating assignments that may be more appealing to students or letting the students choose their own assignment topics (Harris, 2004).
- Procrastination and underestimation of required time and effort. Instructors can reduce the prevalence of this type of plagiarism by setting "intermediate" due dates where small pieces of large assignments are due (Harris, 2004). This way, students are less motivated to plagiarize in order to meet deadlines that have become impossible to meet because of excessive procrastination.
- Low confidence in ability to adequately complete assignments. Instructors can ameliorate this type of motivation for plagiarism by "reassuring" students about additional sources of help, such as through tutors or other sources of out-of-class instruction (Harris, 2004). Instructors can also stress that excess criticism will never be given for original work.
- "Thrill" of breaking rules. Instructors can be conscious of emphasizing the positive aspects of avoiding plagiarism rather than "angrily condemning" the negative aspects of plagiarism; some students feel influenced to resist this kind of condemnation. One positive aspect of citing sources, rather than plagiarizing, is that citations show that the student has conducted research and taken the time to understand another author's opinion, findings *or method for solving a coding problem* (Harris, 2004).

## 5. METHODS TO DETER PLAGIARISM

The main methods for deterring software plagiarism include educating students about what constitutes plagiarism, modifying course grading structure and the ability to adequately detect plagiarism (many software applications exist, which help detect student software plagiarism) (Bowyer & Hall, 2001). Additional methods to deter plagiarism include requiring detailed code specifications and in-code comments and modifying course grade structure. Both of these methods help reveal students' understanding of their submitted assignment code and are difficult to fake when the student has a poor understanding of the assignment. Each of these plagiarism deterrent methods is discussed in more detail in the following paragraphs.

**Plagiarism Education**
According to Bowyer & Hall (2001) and Barry (2006) the most common excuse for plagiarism is ignorance. A student may claim that he or she was not aware of committing plagiarism. However, if students are clearly informed of what constitutes plagiarism, this excuse cannot be used. The proper method of citing code sources can be taught, so that students can cite sources, within code

comments, when necessary. However, students should also be aware that a greater percentage of their code should be original and not borrowed from other sources, similar to a written paper. Barry (2006) suggested that in order to combat this type of plagiarism some courses use assignments that are specifically designed to educate students about plagiarism. For example, students can be required to read an original source code document and then to read source code that plagiarizes from this original document. Next, the students are asked to identify the section of the code they believe to be plagiarized. "Feedback" is given to the students about their responses; students are given both definitions and examples of plagiarism and graded on the accuracy of their assignment responses. Such assignments are graded based on students' submitted description of plagiarism and their application of this description by avoiding plagiarism in their submitted work. In cases where this educational method was used, the scores of students were higher when issued a second similar assignment, which indicates that students developed a better understanding of plagiarism from the first assignment.

Carpenter (2002) suggested that an additional cause of plagiarism is an interesting phenomenon called cryptomnesia. Cryptomnesia occurs when a person is exposed to an idea, such as a programming solution or an idea expressed in natural language and then later recalls this idea without remembering that it came from an external source. The person then 'innocently' uses the idea as if it was original. Studies have shown that this phenomenon exists and is not always intentionally feigned. For example, participants in one study were asked to illustrate completely "novel" alien creatures after viewing example creatures. Participants who viewed examples, tended to add similar characteristics "such as four legs or antennae" in their own illustrations without consciously knowing how much their creature renditions were being influenced by the examples (Carpenter, 2002).

Personal experience with cryptomnesia includes unconsciously re-using similar programming solutions that have been employed or understood in the past. For example, when presented with a programming problem, programming ideas that have been previously used or understood seem to be the first to be evoked by the mind (whether the original source is consciously recalled or not). It is important to note that cryptomnesia is likely to occur less frequently in programming since producing code requires more conscious effort than, for instance, expressing an idea through natural language. While a person is exerting effort to understand code, it becomes harder to forget that the code solution stems from an external source. One article confirms this concept by expressing "when there [are] fewer perceptual and contextual cues--such as the distinctiveness of the voice associated with" expressing the idea, then it is easier for cryptomnesia to occur (Carpenter, 2002). The high amount of conscious effort, required to understand source code, is the "distinctiveness" that should make programming cryptomnesia infrequent.

**Modifying Course Grading Structure**

This deterrent method stems from the idea that removing students' reasons for plagiarizing is an effective way of dealing with plagiarism. For example, in a study of such a method, it was assumed that many students plagiarize because they want to achieve a certain grade without offering the required level of effort. Therefore, the grade-rewards for homework assignments were eliminated and the course grade depended only on in-class activities, such as exams. Although, there were "no incidents of plagiarism" the downfall of this approach proved to be that many students did not offer a desired level of effort on assignments; although these assignments helped students prepare for quizzes and other in-class activities, they were only *indirectly* determinant of grades (Bowyer & Hall, 2001).

**Requiring Detailed Documentation**

Both specifications and comments describe, in natural language, the purpose and function of coded statements. Without a good understanding of the code, it is hard to fake specifications and coded comments. Therefore, students may be less likely to plagiarize code, knowing that they must also be able to describe it, in their own words. Also, if an insufficient specification or no specification is provided along with completed code, this may help foster suspicion of plagiarism.

**Employ Plagiarism Detection Software**
Many software plagiarism detection applications exist, which can be used to automate much of the process of plagiarism detection. The main considerations that should be taken when choosing to use one of these applications are as follows: Cost / access; to what existing software code does the application compare the submitted code; length of time needed to detect for plagiarism; reliability of the application when met with different types of plagiarism methods; and is the detection software language specific? Some software plagiarism detection software can be accessed for free over the Web. An application called JPlag is one example of such software (Clough, 2000).

## 6. PLAGIARISM DETECTION SOFTWARE

**Reliability of the Detection Software**
For example, in the case of the "ghost" source, will the plagiarism detection tool be able to detect the plagiarism. This depends upon what the submitted code is compared to. Some detection applications only compare submitted code to other submitted code and do not build a long-term database of submitted code. Therefore, the user is likely to submit the code from only one class; if a student plagiarizes the work of a previous student, an application without records of this previous assignment will not detect the plagiarism. This limitation is found within the Measure of Software Similarity (MOSS) plagiarism detection software (Bowyer & Hall, 2001).

**Time Needed to Detect Plagiarism**
Depending upon the algorithm and number of comparisons processed by the plagiarism detector, the length of time needed to complete these comparisons varies. For example, one advanced application called CodeMatch can require hours or days to complete its detection processing. CodeMatch uses a combination of common plagiarism detection algorithms and is one of the more thorough detection applications, as indicated by its use in major business copyright infringement cases (Zeidman, 2004).

**Reliability of the Application**
Most plagiarism detection software can detect plagiarism sufficiently well, but some of these applications specialize in detecting certain kinds of plagiarism and are not as well suited in detecting other kinds. For example, MOSS is one application that ignores "comments and identifier names" during the detection process (Zeidman, 2004). The reason for this is, most likely, to prevent MOSS software from being mislead if a plagiarist tries to hide the plagiarism through altering comments and identifier names. However, in the cases where plagiarism can be detected through copied identifiers and comments, this quirk, of programs like MOSS, is not helpful.

Five main detection algorithms, used by CodeMatch software, include the following. *Word Matching* (matches between non-keyword words are counted); *Partial Word Matching* (matches between character sequences within non-keyword words are counted); *Source Line Matching* (matches between non-comment code lines are counted); *Comment Line Matching* (matches between comment code lines are counted); and *Semantic Sequence Matching* (similarities between code segment operations / purposes are counted regardless of how these operations are coded) (Zeidman, 2004). Case is ignored within all algorithms so altered case does not hinder the process of detecting plagiarized code.

**Language Specific Detection Software**
Some detection software is more language specific than other software while some software is designed to detect plagiarism within a wide variety of different languages. For example, MOSS is designed to detect plagiarism within C, C++, Java, Pascal, ADA and more (Bowyer & Hall, 2001). The reason for language specificity is that some detection software relies on the unique 'back-end' details of a specific language in order to detect plagiarism (Clough, 2000).

## 7. PLAGIARISM CONSEQUENCES

The most lenient and common consequence for software plagiarism is credit denied for the plagiarized assignment and a warning. Other consequences include an 'F' grade for the assignment / course; exclusion from the final test (U of WA, 2003); expulsion from the computer science program and expulsion from the university. Attempts at plagiarism may be recorded on a student's record becoming a hindrance when attempting to find employment (Bowyer & Hall, 2001).

Most universities are very careful about ensuring that plagiarism has, in fact, occurred and are thorough when deciding the best way to respond to these incidents. They recognize that plagiarism is a sensitive issue for both the University and the student and is not taken lightly when an accusation of plagiarism is made. For example, in all examined university policies on plagiarism, before the student is "proven guilty" he or she is required to (or given the opportunity to) present evidence that he or she did not commit plagiarism. The University of Western Australia's "Policy on Plagiarism" states that students' suspected of plagiarism will first be "interviewed" before culpability is determined (U of WA, 2003). In another university, suspected plagiarism is handled by, first arranging a student-professor meeting and requesting a "written summary of any information that might help in understanding why the programs were rated as highly similar" (Bowyer & Hall, 2001). Universities are usually very careful about taking precautions against falsely accusing students of plagiarism.

## 8. RECOMMENDATIONS

We discovered that deterring plagiarism is a very time consuming activity. As faculty we are very concerned about the negative effects of plagiarism on our students at the same time, to deter plagiarism, one has to go through a number of time consuming steps. We tried some of the deterring techniques described in the paper. We changed the course structure in such a way that the programming assignments have a very low percentage while the written and hands-on tests have a larger percentage in their final grade calculations. We did not like the outcome that much. We did not use any plagiarism detection software as we are a very small school with very limited funds and limited faculty time to experiment with it. We now use a combination of the techniques described in the paper. We try to educate the students about the problems associated with plagiarism and then inform them of the penalties for plagiarizing programs. We included this in our syllabi and then frequently discussed plagiarism with the students. We also modified our programming class grade structure so that programming assignments carry less weight than before while the written tests and hands-on tests carry a higher weight. We also require the students to develop extensive specification and design together with good comments on the body of the code. We are seeing some positive effects of these changes. Care should be taken when duplicating our effort in other situations as we are a small private institution and we can implement many things without much external interference.

## 9. CONCLUSION

Software plagiarism is a problem in many institutions of higher learning. This is because plagiarism provides a way for a student to complete assignments without gaining a sufficient level of knowledge of the course material. There are many methods and tools available to help reduce the prevalence of plagiarism. Although there are many methods available to detect plagiarized programs, the best way to combat this problem is through the usage of a combination of methods. Some of these methods are: convincing students that academic plagiarism cheats the plagiarist from gaining invaluable knowledge, modifying the course grading structure and requiring detailed documentation with the programs. It should be noted that the process of detecting and deterring plagiarism is a time consuming activity for faculty members.

## REFERENCES

Barry, E. S. (2006, Jun). Can Paraphrasing Practice Help Students Define Plagiarism? *The Pennsylvania State University Fayette*, Retrieved June 10, 2008, from Unknown URL

Bowyer, K., & Hall, L. (2001). Reducing Effects of Plagiarism in Programming Classes. *Journal of Inofrmation Systems Eduction*, *12(3)*, Retrieved May 27, 2008, from http://www.jise.appstate.edu/Issues/12/141.pdf.

Carpenter, S. (2002, Feb). Plagiarism or Memory Glitch? *Monitor on Psychology*, *33*, Retrieved Jun 10, 2008, from http://www.apa.org/monitor/feb02/glitch.html

Clough, P. (2000, Jul). Plagiarism in natural and programming languages: an overview of current tools and technologies. Department of Computer Science, University of Sheffield, Retrieved May 27, 2008, from http://64.233.167.104/search?q=cache:hVIhW9S4Yn4J:ir.shef.ac.uk/cloughie/papers/plagiarism2000.pdf+plagiarism+in+programming&hl=en&ct=clnk&cd=8&gl=us

Harris, R. (2004, Nov 17). Anti-Plagiarism Strategies for Research Papers. *Virtual Salt*, Retrieved Jun 10, 2008, from http://www.virtualsalt.com/antiplag.htm

Lester, Chaky M., Diekhoff, M. George (2002). A comparison of traditional and internet cheaters. b Net, Retrieved Jun 10, 2008, from http://findarticles.com/p/articles/mi_qa3752/is_200211/ai_n9166002/pg_2.

U of WA, (2003). Policy on Plagiarism. University of Western Australia, Retrieved May 27, 2008, from http://www.csse.uwa.edu.au/departmental/publications/policy.on.plagiarism.html

Zeidman, B. (2004, 01). Detecing Source-Code Plagiarism. *Dr. Dobb's Portal*, Retrieved May, 27, 2008, from http://www.ddj.com/architect/184405734