Journal of Computer Science 6 (11): 1341-1346, 2010
ISSN 1549-3636
© 2010 Science Publications

Activity Surveillance and Hawthorne Effect to Prevent Programming Plagiarism

Hairulliza Mohamad Judi, Syahanim Mohd Salleh, Norijah Hussin and Sufian Idris
Faculty of Information Science and Technology,
University Kebangsaan Malaysia, 43600 Bangi, Malaysia

Abstract: Problem statement: Course instructors are facing serious problems in dealing with students who plagiarize programs especially when the number of students in the course is high. Among the proposed approach to handle this problem is by using automatic detection of plagiarism in programming projects. Preventive action is required rather than curing the problem so that programming students get the right message from the beginning. **Approach:** To address this problem, a surveillance system was proposed to record every programming activity. It is developed in an integrated development environment so that programming activity profile in Java format is created when students are developing their Java program. A non-intrusive and non-experimental setting approach was applied in which hidden data collection is conducted to observe students' behavior in natural programming setting. Experimental study effect i.e., Hawthorne effect and effect of expectation on subject behavior was exploited as prevention on plagiarism. Surveillance system produces two file types: Activity log to keep programming activity log information and Backup file to save the program writing record. **Results:** The proposed programming activity surveillance system, DwiCoder presented a programming activity report at the end of each programming session. Students can assess their own progress in developing a program in these three activities: Compilation, execution and modification. The report was presented in a simple and meaningful way to encourage student spend their own time in programming activity. **Conclusion:** By using DwiCoder, student's programming activity is continuously monitored and their behavior is under control. This system provides an effective prevention method in tackling plagiarism.

Key words: Programming activity profile, behavior monitoring, plagiarism prevention

INTRODUCTION

In programming course, programming assignments and projects are conducted to asses students' performance. One of the main problems in this assessment is in dealing with students who copy and modify programs. This problem is especially out of control when the number of students in the courses is high, as it will be very difficult to detect this plagiarism.

With rampant increment in plagiarism phenomena (Joy and Luck, 1999; Sheard and Dick, 2003) necessary efforts should be taken to provide a conducive environment that encourages students to develop their programming skills (Cogan and Gurwitz, 2009; Sani *et al*., (2009). Since it is so easy to copy and edit a computer program, students would find it is tempting to get involve in plagiarism activity.

This study suggests a method that observes students progress in building program in their programming environment. A surveillance system is proposed in an Integrated Development Environment (IDE). This study discusses issues on plagiarism, Hawthorne effect and some approaches to deal with plagiarism. Then the design and architecture of the system is brought forward. Some of the system interface and results will be discussed. Finally the research concludes the study.

Related work: Researchers (Joy and Luck, 1999; Daly and Horgan, 2005; Spinellis *et al*., 2007) have identified plagiarism issues in many perspectives. Unacknowledged copying of documents or programs is considered as an act of plagiarism. The widespread plagiarism scenario faced by instructors' at all educational levels is due to two reasons: Increasing facilities that students have for accessing to on-line resources and the huge number of work, project or report-based assessment of courses that need to be evaluated by instructors.

Corresponding Author: Hairulliza Mohamad Judi, Faculty of Information Science and Technology, University Kebangsaan Malaysia, 43600 Bangi, Malaysia Tel: 00603-89216180 Fax: 00603-89256732

Stewart-Gardiner *et al*. (2001) raise interesting issues regarding collaboration and plagiarism. These include the breaking point between collaboration and plagiarism. While many educators feel that collaboration belongs only in a very few upper division programming courses, others have experience to show that early collaboration broadens the learning of students, to become more effective professional individuals. They find out that a blend of the two styles is best for students and can reduce plagiarism.

Spinellis *et al*. (2007) agree with the blended approach in dealing with plagiarism. They do not only evaluate students' work in terms of originality, but also understanding, learning, fairness, difficulty, fun and interest as a result of collaboration with other students. They propose Jarpeb, a system that creates individually randomized assignments, grades the students' programs and allows students to submit their grade through the web. The results indicate that the system contributes to the reduction of plagiarism, increases the understanding and learning of the course subject while also increasing the perceived fairness, fun and interest of the learners.

Hawthorne effect is a phenomenon where a study subject's behavior or study outcomes are altered as a result of the subject's awareness of being under observation (Mangione-Smith *et al*., 2002). This phenomenon was originally identified at the Hawthorne Works Plant of the Western Electric Company in Chicago. Several studies were conducted at this plant between the years 1924 and 1932 in order to identify working conditions that would increase the productivity of the employed by the plant. The investigators found that worker productivity increased regardless of working conditions when the workers knew they were under observation. For example, both more light and less light in the workroom resulted in improved performance when workers were aware that their productivity was being measured.

Approach in surveillance system: The approach to deal with plagiarism in programming assignment needs suitable methodology and technique in which different modes will be used to gather data and suitable to various states and conditions. With the availability of more complex technique of data collection, observation study especially via embedded system is seen as more effective (El-Mousa and Al-Suyyagh, 2010; Rath and Meher, 2006; Rath and Dehuri, 2006). The development of instrument to deal with plagiarism in this study considers some important factors in physical specifications. The approach will cover both observational and prevention aspects.

Observational approach requires observational technique that monitors the programming activity conducted by students which fulfill the following features. First, a student programming environment: the system should consider the study environment including attentive to issues on students' learning capacity and their learning style. Student needs an easily-used and maintained programming environment. As such, the development of programming activity information collection function in an embedded Integrated Development Environment (IDE) will be one advantage due to the capacity and availability of the software itself. The surveillance software in a Java programming environment is intended to record the identified programming activity during students' Java program development.

Next, a non-experimental setting: experimental setting emphasizes variable manipulation which results-in experimental environment constraint. Among the major flaws in experimental setting is that subjects' behaviors are distorted from their normal ones. On the other hand, non-experimental setting explores the phenomenon being studied and is very suitable if the information regarding the study are very limited. By using this setting, the study findings could be generalized into a bigger group instead of certain group representation. The integrated observation function in an IDE environment also allows for exploratory data collection.

Then, observational data collection: data collection technique plays the most critical part and identifies the purpose of study (Taylor-Powell and Steele, 1996). In this context, observation is made when the students are developing their program. In this technique, the response is not determined by subject's needs and capability in giving information.

Finally, non-intrusive methodology: intrusive observation would influence subject's performance whereas manual data collection is not efficient at all. Therefore, a non-intrusive approach is applied. Hidden data collection is conducted to observe programming students' behavior. Among the recorded information is the number of compilation they are conducting and this information is gathered without them knowing.

In preventive approach, the software development takes into account the encouragement effect on student attitude. Instead of compulsory attitude change, students are encouraged to perform well in their assignment and fulfill their responsibility in spending sufficient time and effort in programming activity. The prevention approach is conducted by monitoring students' programming activity. They realize that their action is recorded and controlled. Two aspects are considered in this approach: effect towards observation and Hawthorne effect.

First, effect towards observation: students need to be given full information on study planning and the intended result of the prevention tool. The psychological impact would be created based on what the subjects think on real world. Subject's knowledge or their own expectation on the study purpose and their own desire to be seen as good subject in researcher's eye would result in action that follows researcher's needs. In this situation, the students will commit in normal programming activity such as typing, debugging and implementing their program. It is the role of course instructor in disseminating information regarding the study planning clearly at the beginning of the course.

Second, Hawthorne effect: an open programming activity pooling technique through observation would influence subject's behavior. Hawthorne effect is exploited in student's environment to maximize the surveillance system function as a preventive approach on plagiarism activity. Although the recording of process information is hidden, a report is generated at the end of each programming session that informs student development process. The report is presented in a simple and meaningful way to encourage student spend their own time in programming activity.

MATERIALS AND METHODS

The surveillance system is implemented in such a way that no significant change is observed in the available working environment. Surveillance system will produce two file types: activity log (special file) and backup file. Activity log file is created to keep programming activity log information as reference purposes if more information is needed. Backup file (*.bak) is created to save the program writing record. The surveillance function is developed by fulfilling certain criteria as summarized in Table 1.

In the proposed approach, each student in the programming course is supplied with surveillance system. It is developed in a Java programming environment which enables recording the identified programming activity during program development. The software is equipped with programming activities supervision function. Each time students are working on the programming assignment, the system will be able to record the activity. Even though the observation function has been implemented in the system's environment, basically no obvious changes have been made on the present programming interface environment. The function was built transparently with recording, analytic and programming activity sequence display functions. Figure 1 shows the environment of the surveillance system called DwiCoder.

Table 1: Requirements in developing surveillance function

Requirement	Description
Ubiquity	Enable monitoring of each activity in developing source program.
Quantitative	Each recorded activity should contain quantitative information which could be used in further analysis
Not intrusive	Activity recording is not made clearly
Transparent	Programmer should be able to interact naturally with the developing environment. Log record should not limit of his/her choice.
Objective	Analysis of log activity should be repeated using similar criterion
Individual	Each log should be individual, i.e., it is unique for one file only.
Hidden	Recording function should be implemented indirectly
Output	Programming activity output should be displayed in sequence
Writing record	The programming codes should be kept.
Data integrity	Recorded data should have high integrity

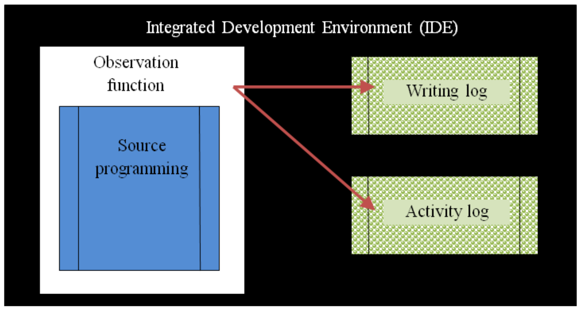
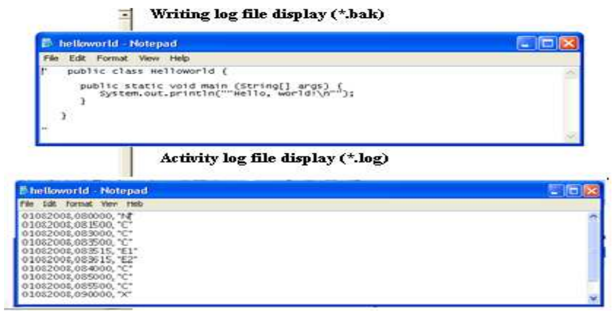


Fig. 1: Environment of DwiCoder


Writing log file display (*.bak)
helloworld - Notepad
File Edit Format View Help
Activity log file display (*.log)
helloworld - Notepad
File Edit Format View Help

```
    public class Helloworld {

      public static void main (String[] args) {
         System.out.println(""Hello, world!\n"");
      }

   }
```

```
01082008,080000, "N"
01082008,081500, "C"
01082008,083000, "C"
01082008,083500, "C"
01082008,083515, "E1"
01082008,083615, "E2"
01082008,084000, "C"
01082008,085000, "C"
01082008,085500, "C"
01082008,090000, "X"
```

Fig. 2: Model of the system

Programming activity is recorded in hidden manner and the information is kept in a special file called log file. These data are kept in separate files and are created uniquely for every program source. They have a standard format and treated as raw data in programming study. Observation function also produces one writing text file which stores program text when it is compiled for the first time. This file is built as a backup file if source file suffers some damage. Figure 2 displays the model of the whole system.

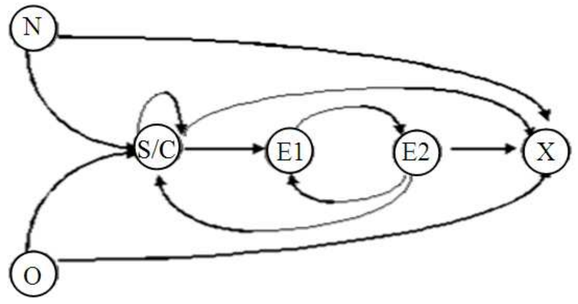


Fig. 3: Flow of programming activity

In the observation function, programming activity record is created based on a standard format. Three main attributes are provided: Date, time and programming code. This requirement is important because one programming session might pass a time frame with changing dates. Data is recorded following to the treatment of activity, state transition in document program and main programming activity (Fig. 3). In the sequence, program source would first be built (state "N"), followed by compilation and save process (state "C" and "S"). Next, program will be executed (state "E1") while state "E2" marks the end of program execution. Program file will be closed (state "X") when program session ends and program file will probably be opened (state "O") again if the programmer want to resume his/her programming process. Compilation, save and execution process perhaps will occur repeatedly in every programming session. Programmer may end each session in four main situations: After compilation or save, after implementation process, when a document is created and when a document is opened.

RESULTS

DwiCoder presents the programming activity log in a sequence format. The format allows for displaying as much information as possible that can be analyzed according to the content of activity log. The observation function captures specific and general observation of students' programming behavior based on their programming activity basis.

A data set that represents a metric of programmer oriented behavior is generated by the system. This metric fall into four categories that measure different aspects in the programming procedure: Time profile, compilation and implementation activity, mistake measurement and program solution equality. Table 2 lists the metric.

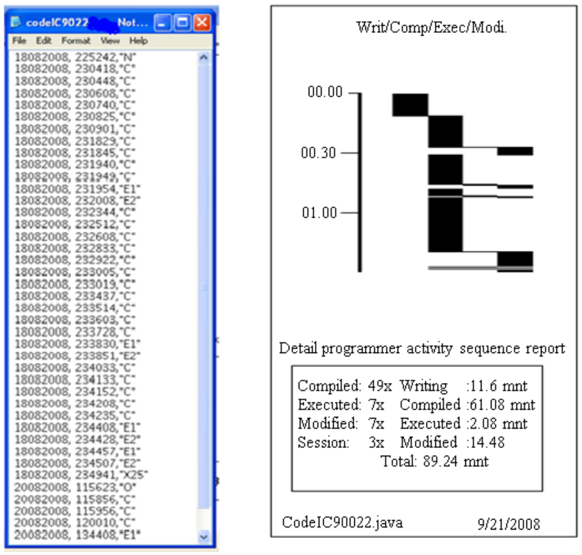
Table 2: Programmer oriented behavior measurement

Time profile	
DT	Development time
NoS	Programming session number
Compilation and implementation activity	
NoC	Compilation number
NoE	Implementation number
NoM	Modification number
WT	Writing time-development time until the first time of compilation
DTtLC	Development time after the last time of compilation
CI	Compilation Interval
CiSD	Standard deviation of compilation interval
CoT	Time needed to free the program from syntax error
MoT	Modification time
EoT	Implementation Time
Compilation of mistake	
NoFC	Failed compilation number
NoSC	Success compilation number
FCP	Failed compilation percentage
SCP	Success compilation percentage
FCNbDT	Failed compilation normalization between development time
SCNbDT	Succeed compilation normalization between development time
Solution equality	
Score	Percentage of program equality from the whole solution

The system generates a time graph, one of its display tools to show activity sequence which occurred. All information of necessity is displayed in the sequence of time. Graphical display format (Fig. 4) is produced so that the recorded data could be interpreted easily to follow four main phases: Writing, compilation, implementation and modification. Definition of each phase is adapted fully from program compilation, implemented and modification classification (Takada *et al*., 1994).

This report generating function is one of DwiCoder's menu choices so that student can see their programming progress and print their program as an evidence purpose. Figure 4 also shows an example of the report for a program. With the richness and systematic recorded information, the report is expected to give an encouragement for students to be more responsible on their work. The report shows that one of the expectations in the programming assignment is sufficient time and compilation number was fulfilled in developing the program in these three activities: Compilation, execution and modification. This report display is not only useful for research use on programming even may serve as a self reporting mechanism. Students are able to monitor their own progress and check whether they fulfill the assessment criteria.

This surveillance system has the ability to record any activity based on programmers' selection on related menus only. It is more difficult to consider programmer's mind which is not related to programming activities or leave his/her programming environment without closing the programming session.



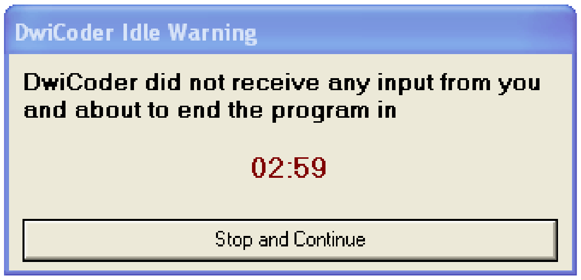
```
18082008, 225242,"N"
18082008, 230418,"C"
18082008, 230448,"C"
18082008, 230608,"C"
18082008, 230740,"C"
18082008, 230825,"C"
18082008, 230901,"C"
18082008, 231829,"C"
18082008, 231845,"C"
18082008, 231940,"C"
18082008, 231949,"C"
18082008, 231954,"E1"
18082008, 232008,"E2"
18082008, 232344,"C"
18082008, 232512,"C"
18082008, 232608,"C"
18082008, 232833,"C"
18082008, 232922,"C"
18082008, 233005,"C"
18082008, 233019,"C"
18082008, 233437,"C"
18082008, 233514,"C"
18082008, 233603,"C"
18082008, 233728,"C"
18082008, 233830,"E1"
18082008, 233851,"E2"
18082008, 234033,"C"
18082008, 234133,"C"
18082008, 234152,"C"
18082008, 234208,"C"
18082008, 234235,"C"
18082008, 234408,"E1"
18082008, 234428,"E2"
18082008, 234457,"E1"
18082008, 234507,"E2"
18082008, 234941,"X25"
20082008, 115623,"O"
20082008, 115856,"C"
20082008, 115956,"C"
20082008, 120010,"C"
20082008, 134408,"E1"
```

Detail programmer activity sequence report

```
Compiled: 49x  Writing   :11.6 mnt
Executed: 7x   Compiled :61.08 mnt
Modified: 7x   Executed :2.08 mnt
Session:  3x   Modified :14.48
            Total: 89.24 mnt
```

CodeIC90022.java 9/21/2008

Fig. 4: Log content and graph display of activity



DwiCoder Idle Warning

DwiCoder did not receive any input from you and about to end the program in

02:59

Fig. 5: Display of security message

As such, a security mechanism is built behind the scenes in DwiCoder to identify keyboard activity. It will close the programming session automatically if there is no reaction from programmer when a warning message appears, as in Fig. 5. It is found to be effective in reducing time recording for activities not belong to programming log.

DISCUSSION

DwiCoder is a surveillance system that employs observation and prevention approach. The former approach enables for monitoring the programming activity conducted by students that consider their learning capacity in an integrated development environment. Hidden data collection is conducted to observe the programmers' behavior. The later approach takes into account the encouragement effect on student attitude. Instead of compulsory attitude change, students are encouraged to perform well in their assignment and fulfill their responsibility in spending enough time in programming activity.

The Hawthorne effect phenomenon is a result of a careful setting to the environment. Assuming an increment in student performance resulting merely from instructor attention to them is among the mistakes in the usage of the term (Gottfredson, 2005). Remarkable improvements in students' performance might be contributed by various factors such as conducive and enjoyable laboratory environment in which students are willing to spend long time to complete the assignment, students' personal goals of obtaining their degree with flying colors, or careful attention to their performance since the first day in that semester.

To achieve the desired result in programming course, course instructor plays an important role. Not only depending on the surveillance system, it is important for instructor to keep varying the course to keep it from getting stale. Such changes might involve instructor's approach in presenting material and collaborating with other students (Spinellis *et al*., 2007; Stewart-Gardiner *et al*., 2001). The new and innovative method in marking students' assignment would probably change and improve student's learning style and behavior to spend enough time to develop the program. A reasonable change would cause an improvement in student learning especially when they perceive that the instructor is giving them attention that seems special.

With the ability to record complex data in programming, the surveillance system gives an advantage to implement observational study as compared to experimental study previously done. The inclusion of data collection function in present environment brought to by DwiCoder is not a new approach, tools such as AESOP and Mother are able to gather information to monitor student's behavior on-line (Kivi *et al*., 1998).

There are various tools and systems for automatic detection of plagiarism in programming projects. JPlag and Turnitin use similarity index as an evidence of plagiarism. Daly and Horgan (2005) present a technique for detecting plagiarism in computer code, which has the advantage of distinguishing between the originator and the copiers and handling a large group studying programming in an automated learning environment.

DwiCoder appears to be a better method for collecting information on subject's behavior due to its ability to be carried out in a non-experiment environment and does not require subject intervention. The programmer's action and response in the programming environment such as pressing the keyboard or calling on certain menus could be detected.

This approach implies a self reporting technique which might expose some element of bias. In future, monitoring on some non-programming activity domains like eating, drinking or opening email could be included as an enhancement of the system.

CONCLUSION

This study proposes a surveillance system called DwiCoder to handle plagiarism problem in programming projects. It applies observation and preventive approach to monitor students' programming activity and to promote good practice in spending enough time in programming activity.

DwiCoder provides a learning environment in which student's programming activity will be continuously under observation and their behavior is under control; therefore the environment is an effective prevention method in tackling plagiarism. Apart from that, this system affords to give enough evidence on student's effort to resolve their assignment.

REFERENCES

Cogan, E. and C. Gurwitz, 2009. Memory tracing. J. Comput. Sci., 5: 608-613. DOI: 10.3844/.2009.608.613

Daly, C. and J. Horgan, 2005. A technique for detecting plagiarism in computer code. Comput. J., 48: 662-666. DOI: 10.1093/comjnl/bxh139

El-Mousa, A.H. and A. Al-Suyyagh, 2010. Embedded systems education for multiple disciplines. J. Comput. Sci., 6: 186-193. DOI: 10.3844/.2010.186.193

Gottfredson, G.D., 2005. Hawthorne Effect. In: Encyclopedia of Statistics in Behavioral Science, Everitt, B.S. and D.C. Howell (Eds.). John Wiley and Sons, New Jersey, ISBN: 0470860804, pp: 784-785.

Joy, M. and M. Luck, 1999. Plagiarism in programming assignments. IEEE Trans. Educ., 42: 129-133. DOI: 10.1109/13.762946

Kivi, M.R., T. Gronfors and A. Koponen, 1998. MOTHER: System for continuous capturing of display stream. Behav. Inform. Technol., 17: 152-154. DOI: 10.1080/014492998119517

Mangione-Smith, R., M.N. Elliott, L. McDonald and E.A. McGlynn, 2002. An observational study of antibiotic prescribing behavior and the hawthorne effect. Health Serv. Res., 37: 1603-1623. DOI: 10.1111/1475-6773.10482

Rath, A.K. and P.K. Meher, 2006. Design of a merged DSP microcontroller for embedded systems using discrete orthogonal transform. J. Comput. Sci., 2: 388-394. DOI: 10.3844/.2006.388.394

Rath, A.K. and S.N. Dehuri, 2006. Non-dominated sorting genetic algorithms for heterogeneous embedded system design. J. Comput. Sci., 2: 288-291. DOI: 10.3844/.2006.288.291

Sheard, J. and M. Dick, 2003. Influences on cheating practice of graduate students in IT course: What are the factors? Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education, June 30-July 02, ACM Press, Thessaloniki, Greece, pp: 45-49. DOI: 10.1145/961511.961527

Sani, N.F.M., A.M. Zin and S. Idris, 2009 Implementation of CONCEIVER++: An object-oriented program understanding system. J. Comput. Sci., 5: 1009-1019. DOI: 10.3844/.2009.1009.1019

Spinellis, D., P. Zaharias and A. Vrechopoulos, 2007. Coping with plagiarism and grading load: Randomized programming assignments and reflective grading. Comput. Appli. Eng. Educ., 15: 113-123. DOI: 10.1002/cae.20096

Stewart-Gardiner, C., D.G. Kay, J.C. Little, J.D. Chase and J. Fendrich *et al*., 2001. Collaboration Vs plagiarism in computer science programming courses. Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education, ACM Press, Charlotte, North Carolina, United States, pp: 406-407. DOI: 10.1145/364447.364790

Takada, Y., K. Matsumoto and K. Torii, 1994. A programmer performance measure based on programmer state transitions in testing and debugging process. Proceeding of the 16th International Conference on Software Engineering, May 16-21, IEEE Computer Society, Sorrento, Italy, pp: 123-132. DOI: 10.1109/ICSE.1994.296772

Taylor-Powell, E. and S. Steele, 1996. Collecting evaluation data: An overview of sources and method. University of Wisconsin-Extention. http://www.worldbridgeresearch.com/files/Methods.pdf