http://ansinet.com/itj

ITJ

ISSN 1812-5638

# INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 9 (1): 98-106, 2010
ISSN 1812-5638


# The Use of Assignment Programming Activity Log to Study Novice Programmers' Behavior Between Non-Plagiarized and Plagiarized Groups

[1]H. Mohamad Judi, [2]S. Mohd Salleh, [2]N. Hussin and [2]S. Idris
[1]Department of Industrial Computing,
[2]Department of Computer Science, Faculty of Information Sciences and Technology,
Universiti Kebangsaan Malaysia, 43600 UKM Bangi Selangor, Malaysia

**Abstract:** The objectives of this study are to identify novice programmers' behavior pattern during program development and compare the pattern between non-plagiarized and plagiarized groups. To collect the data in this study, a surveillance software in an integrated development environment is used to record the programming activity log comprising of 19 programming behavior items. A laboratory test which involves two student groups: plagiarized and non-plagiarized novice programmers was conducted with 13 and 14 students in respective group. The study found that plagiarized group score higher in programming mark and spend less time in developing program compare to non-plagiarized group. In plagiarized group, a unique pattern was found in which students spend much more time to modify program to score higher marks. Students seldom compile their work and would be satisfied with the program once it is executable. In its counterpart, students score higher programming mark if they compile their program many times, spend long time in developing the program after last compilation and be able to successfully compile the program.

**Key words:** Plagiarism, programming behavior, novice programmer, program

## INTRODUCTION

Students in a programming course are usually regarded as novice programmers, due to their lack of experience in programming fields. Since, they are newly exposed to the programming skills, sufficient time is required for them to understand and gain the technical capability in developing a good software. They need to spend their learning time in three main activities in programming: writing, compiling and implementing programs. Programming assignments and projects are given to them especially to sharpen their talent and skills in programming field.

Unfortunately, increment in plagiarism cases in programming assignment were reported in many studies (Joy and Luck, 1999; Sheard and Dick, 2003). Students who involved in such activity should be given some advice, guidance and encouragement, otherwise they would be detrimental in their learning process. Necessary efforts should be taken to provide an environment that helps them study effectively in the course. Various softwares are available in the education field that help course instructors to detect plagiarism in program assignments. Course instructor could also monitor students' learning performance including their programming behavior and gather some information from their assignment. The information would be useful to identify novice programmers' behavior and learning style towards programming.

This study focuses on empirical research on programming activity conducted by novice programmers. The study aims at examining novice programmer activities in two groups: plagiarized and non-plagiarized. The study identifies a metric of novice programmer behavior, in terms of three main programming activities. This study presents the results of programming activity being observed automatically by one surveillance system using the identified metric. Correlation analysis on the generated data is presented and discussed in the study.

This study will discuss the results of programming activity being observed automatically by one surveillance system. The software is called Dwicoder, aims to monitor and record every programming activity during program development. This surveillance system is embedded in student programming environment and is implemented in Java programming language course.

Programming activity is recorded in hidden manner and the information is kept in a special file called activity

**Corresponding Author:** H. Mohamad Judi, Department of Industrial Computing, Faculty of Information Sciences and Technology, Universiti Kebangsaan Malaysia, 43600 UKM Bangi Selangor, Malaysia
Tel: 0060389216668 Fax: 0060389256732

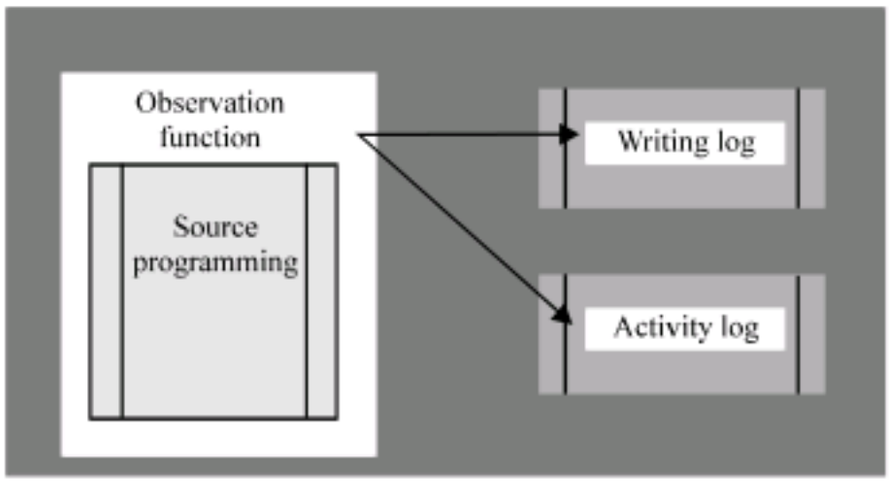


Fig. 1: Environment of Dwicoder

log file (*.log). These data are kept in separate files and are created uniquely for every program source. The data have a standard format and treated as raw data in programming study. Besides the activity log file, the surveillance system will also produce backup file. Backup file (*.bak) is created to save the program writing record. Figure 1 shows the environment of the surveillance system. Data on programming activity are used as main reference to detect plagiarized activity in the lowest level. This happened when program is copied entirely with minimum modification.

Plagiarized activity is detected based on very small program writing time and compilation number. When this case happened, program content similarity will be made through text comparison of program record file (*.bak) against any program file in investigation domain. However, detection using this artifact information becomes less effective if the whole program is typed personally by the suspected student. A support system must be used to detect plagiarism case.

Jplag, Moss and SID are examples of plagiarism detection systems. JPlag detection system is giving assessment in score value and this software could be achieved on-line. It is among the most popular softwares that detect plagiarism based on similarity in the program and could be used for text and also program (Prechelt *et al*., 2001). Bugarín *et al*. (2009), Spinellis *et al*. (2007), Joy and Luck (1999) and Daly and Horgan (2005) have identified various approaches in tackling programming plagiarism especially involving softwares. There is a weakness in using detection system based on program similarity in novice programmer environment. The similarity cases become too large and investigation procedure become so difficult because of restrictions in students' knowledge and skill in their learning domain.

In this study, experimental approach is implemented to answer several issues in study objective. Experimental study involves control experimental groups to study the effect of certain treatment applied to the experimental group. For research studies examining programming plagiarism and performance in educational environment, experimental study may be used as a methodology to collect the data especially when the studies involve observational data. In fact, several experimental designs aimed at examining the effect of certain factors have deliberately included observation on students and examining the differences between groups (Green, 2007; Tsai and Pohl, 2007; Xu and Chen, 2005).

A set of metric is used in this study that represents programmer oriented behavior. Altogether, there are 19 items in the metric. This metric falls into four categories that measure different aspects in the programming procedure: time profile, compilation and implementation activity, mistake measurement and program solution similarity.

Time profile measures development time and programming session number:

- DT ( Development time)
- NoS (Programming session No.)

Compilation and implementation category comprises of ten measurements:

- NoC (Compilation No.)
- NoE (Implementation No.)
- NoM (Modification No.)
- WT (Writing time-development time until the first time of compilation)
- DTfLC (Development time from last compilation)
- CI (Compilation interval)
- CiSD (Standard deviation of compilation interval)
- CoT (Compilation time-time needed to free the program from syntax error)
- MoT (Modification time)
- EoT (Implementation time)

Metric related to compilation mistake comprises of six items:

- NoFC (Failed compilation No.)
- NoSC (Successful compilation No.)
- FCP (Failed compilation percentage)
- SCP (Successful compilation percentage)
- FCNbDT (Failed compilation normalisation between development time)
- SCNbDT (Successful compilation normalisation between development time)

Finally, metric related to program similarity comprises of one measurement:

- Score (Percentage of program similarity from the imitated program or from the whole solution)

## MATERIALS AND METHODS

This study which takes two years from 2005 to 2007 intends to investigate the behavior of novice programmer in developing their programs. In specific, it examines the relationship between items in programmers' behavior metric that exist in non-plagiarized and plagiarized groups. To collect the data in this study, a surveillance software in an integrated development environment is used to record the data relating to 19 items in behavior metric for each participating subject during programming activity. Data collection is conducted by using observation technique via surveillance system.

An experimental approach is used in this study which involves two groups: plagiarized and non-plagiarized novice programmers. To make the experimental study environment very close to a normal programming exercise for novice programmer, a laboratory test was conducted to two groups of students. As many as 27 students (15 men and 12 women) were included in this study. In a limited computer laboratory environment, one test session could accommodate around 30 students only. These students are registered in an object oriented programming course offered by Selangor International Islamic University College (KUIS). This is a compulsory course offered to all diploma students majoring in Computer science. These students have taken C programming course in the previous semester. Students' age range between 18 to 24 years. All participating students in this study is randomly divided into two groups. These groups were supervised by one of the researchers herself.

Students were informed clearly from the beginning of the class regarding the Integrated Development Environments (IDE) that were used in their class. This surveillance system is activated only six weeks after the course begins to give familiarity and environment adaptation prior to research study implementation. Observation function that was developed behind the IDE scenes aims at recording students' programming activity and displaying graphical report (Fig. 2). They may serve as a student program evidence.

Correlation analysis was conducted that aims at examining significant relationship exist between novice programmer behaviors within plagiarized non-plagiarized atmosphere. These programmers' behaviors are represented by behavior metric being discussed previously. This study is not a real experimental design study in the sense that no manipulation activity was

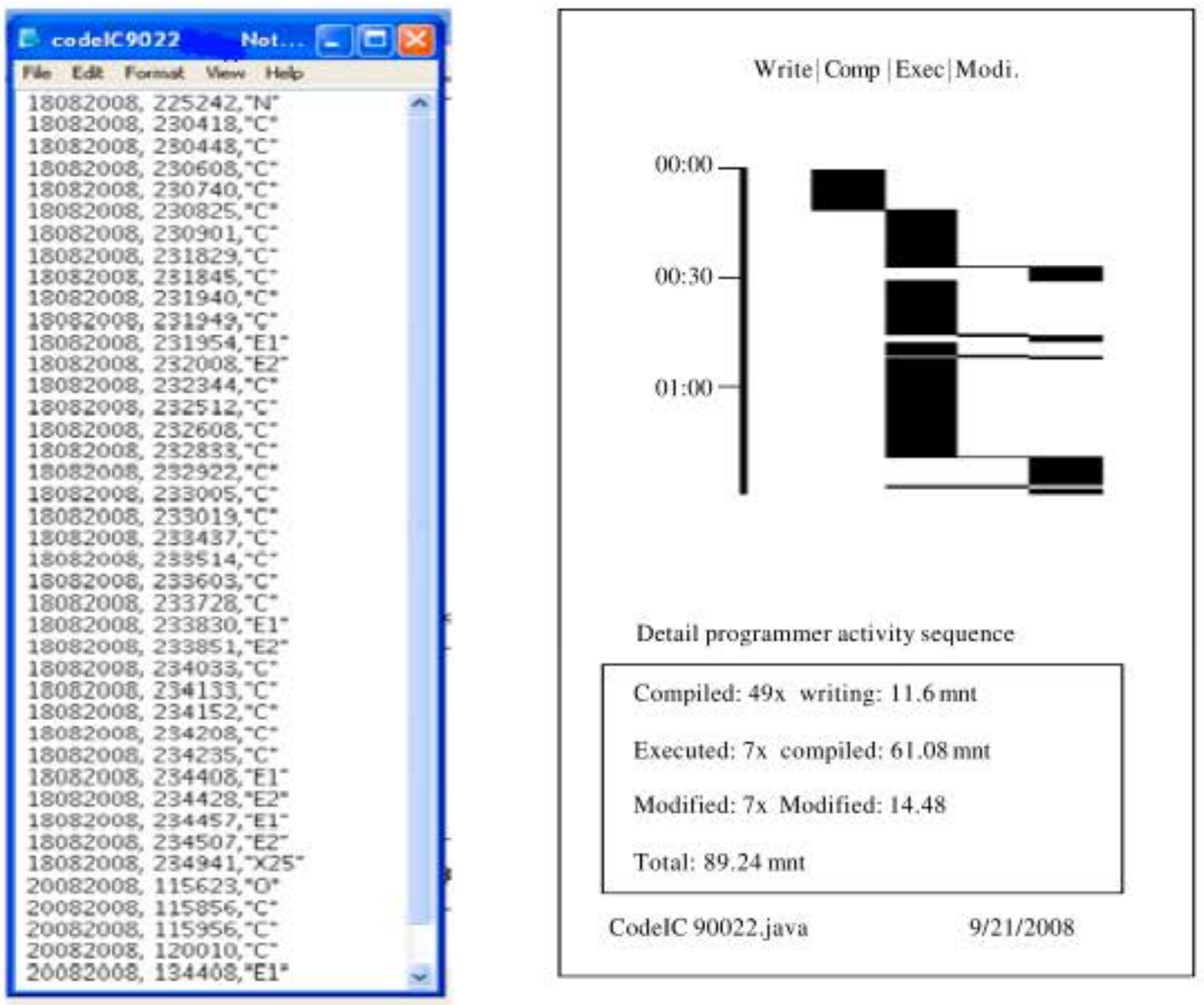


Fig. 2: Log content and graph display of activity

committed against any metric used. In the given assignment, students were asked regarding their knowledge and understanding on the course materials that were covered during first five weeks of this course. Students were expected to gain the necessary knowledge as many main concepts and techniques in the program were built in previous exercises, whether being discussed in class or resolved in earlier assignment. The first group is a control group whereas the second one as experimental group. These groups were separated in different rooms. Both groups received the same treatment. They should complete a programming assignment within 2 h and 30 min. In the first half hour, both groups were allowed to discuss in seeking and planning their solutions. In contrary to the first group, the second group is given solution sample after the first 30 min.

The first group involves non-plagiarized students in which they developed program themselves. They also placed next to their student partner so that they could collaborate maximally in this period in providing their solution problem in the first 30 min. Although, they could imitate the solution during this session, close supervision from course instructor would control students' activity. Students may ask for syntax design but they must produce their program themselves.

In the second group, students receive solution sample for the problem. Hence, they are treated as plagiarized group. Though students in this group were given a solution sample, they were requested to give dissimilar answer from the received one. Apart from that, the second group is exposed to the same environment as in the first group. Of both this group, they will be given programming score based on their ability to develop executable program and at minimum time. The main difference between both groups is students in the second group receive solution to questions in hard copy form. Of both this group, subjects do not know that they involved in plagiarism study.

Particular time is provided for both groups in terms of laboratory test to increase students' integrity and motivation in learning the programming skills. This was necessary because in a normal environment, students should prepare themselves before working on their programming assignment-whether by revising notes, referring to books etc. It is difficult to give equal preparation time for each individual in an experimental study. Programming assignment environment for both groups follows open test format in which students can bring any reference of necessity to develop their solution. However this reference should not involve digital format or program solution which might invite students to copy from the available program.

Data is collected by observation technique via surveillance system. The recorded data are saved as activity log file which contain 19 items in behavior metric. The data in this study is analyzed by using SPSS statistical software. Correlation analysis is conducted to measure the relationship between the items in the metric (Coakes, 2005). Independent samples t-test was also executed to examine significant difference between the groups. Prior to conduct the test, the data was checked for certain assumptions including independence of sample elements and normality of sample distribution. Independent assumption was assessed by observing the method to retrieve the data and sampling technique. Normality assumption can be assessed for each variable by examining the skewness and kurtosis (Hair *et al*., 2006). No serious violation of the assumptions occur, thus t-test was conducted. Homogeneity of variance assumption was tested by Levene's test. If the probability value is smaller than 0.05, there is violation of equal variances. Accordingly, the t-test results for equal variances not assumed will be used from SPSS output.

## RESULTS

There are twenty seven students participate in this study, with fourteen and thirteen students in the first and second group, respectively. However, only ten and nine cases in the respective groups were remaining in the study after examining their programming solution. Only solutions with executable programming and precisely answering the questions were taken into account.

The results of mean comparison between non-plagiarized and plagiarized groups are given in Table 1. It shows that mean comparison between two groups reveal significant difference in score, programming development time, compilation number, writing time, standard deviation of compilation interval, compilation time, implementation time and failed compilation number. Among 19 items, the top two in the list show important result. The programming score mean for the non-plagiarized group is 42.47, whereas 67.67 for plagiarized counterpart. The development time to the first group is higher, (Mean = 89.92) compared to second group (Mean = 64.96).

The Pearson's r correlation results between novice programmers' behavior metric for non-plagiarized group are displayed in Table 2. The results for plagiarized group are in Table 3.

Table 2 displays correlation results for non-plagiarized group. Among important results in Table 2 is regarding six relationships that involve Score. This item shows medium to high correlation with following items:

Table 1: Mean comparison between groups

| Behavior metric | Non-plagiarized group | Plagiarized group | Significance |
|---|---|---|---|
| Score | 42.47 | 67.67 | ** |
| DT (Development time) | 89.92 | 64.95 | ** |
| NoS (Session No.) | 2.80 | 2.89 | - |
| NoC (Compilation No.) | 35.60 | 27.44 | * |
| NoE (Implementation No.) | 14.00 | 11.44 | * |
| NoM (Modification No.) | 14.00 | 11.44 | * |
| WT (Writing time) | 32.03 | 24.16 | * |
| DTfLC (Development time from last compilation) | 1.69 | 1.96 | - |
| CI (Compilation interval) | 1.25 | 1.06 | - |
| CiSd (Standard deviation of compilation interval) | 7.97 | 5.96 | * |
| CoT (Compilation time) | 30.78 | 19.33 | ** |
| MoT (Modification time) | 18.97 | 17.11 | - |
| EoT (Implementation time) | 8.13 | 4.34 | ** |
| NoFC (Failed compilation No.) | 25.70 | 18.22 | * |
| NoSC (Successful compilation No.) | 9.90 | 9.22 | - |
| SCP (Successful compilation percentage) | 34.51 | 37.30 | - |
| FCP (Failed compilation percentage) | 65.49 | 62.70 | - |
| FCNbDT (Failed compilation normalisation between development time) | 0.27 | 0.28 | - |
| SCNbDT (Successful compilation normalisation between development time) | 0.12 | 0.15 | - |

**Significant at p<0.01. *Significant at p<0.05

Table 2: Relationship in non-plagiarized group

| Behavior metric | r | |
|---|---|---|
| NoC (Compilation No.) | Score | 0.721 |
| DTfLC (Development time from last compilation) | Score | 0.648 |
| CiSd (Standard deviation of compilation interval) | DT (Development time) | 0.794 |
| CoT (Compilation time) | DT (Development time) | 0.842 |
| | NoC (Compilation No.) | 0.782 |
| MoT (Modification time) | NoE (Implementation No.) | 0.648 |
| | NoM (Modification No.) | 0.648 |
| NoFC (Failed compilation No.) | Score | -0.718 |
| | DT (Development time) | 0.693 |
| NoSC (Successful compilation No.) | MoT (Modification time) | 0.713 |
| FCP (Failed compilation percentage) | Score | -0.811 |
| | DTfLC ((Development time from last compilation) | -0.665 |
| | CiSd (Standard deviation of compilation interval) | 0.707 |
| | CoT (Compilation time) | 0.744 |
| SCP (Successful compilation percentage) | Score | 0.811 |
| | DTfLC ((Development time from last compilation) | 0.665 |
| | CiSd (Standard deviation of compilation interval) | -0.707 |
| | CoT (Compilation time) | -0.744 |
| FCNbDT (Failed compilation normalisation between development time) | Score | -0.806 |
| SCNbDT (Successful compilation normalisation between development time) | MoT (Modification time) | 0.733 |
| | SCP (Successful compilation percentage) | -0.756 |
| | FCP (Failed compilation percentage) | 0.756 |

Table 3: Relationship in plagiarized group

| Behavior metric | r | |
|---|---|---|
| MoT (Modification time) | Score | 0.767 |
| | CoT (Compliation time) | -0.683 |
| NoFC (Failed compilation No.) | MoT (Modification time) | -0.891 |
| NoSC (Successful compilation No.) | DTfLC (Development time from last compilation) | -0.714 |
| | CI (Compilation interval) | -0.734 |
| | EoT (Implementation time) | 0.748 |
| FCP (Failed compilation percentage) | MoT (Modification time) | -0.895 |
| SCP (Successful compilation percentage) | MoT (Modification time) | 0.895 |
| FCNbDT (Failed compilation normalisation between development time) | MoT (Modification time) | -0.919 |
| SCNbDT (Successful compilation normalisation between development time) | DTfLC (Development time from last compilation) | -0.833 |
| | CI (Compilation interval) | -0.753 |
| DtfLC (Development time from last compilation) | NoE (Implementation No.) | -0.717 |
| | NoM (Modification No.) | -0.717 |
| | WT (Writing time) | 0.767 |

Compilation number (NoC) (r = 0.721), Development time from last compilation (DTfLC) (r = 0.648), NoFC (r = -0.718), FCP (r = -0.811), SCP (r = 0.811) and FCNbDT (r = -0.806).

Results for plagiarized group are presented in Table 3. Among important results involve Score and Modification time (MoT) items. Contrary to Table 2, this Table 3 only has one pair that involves Score, Score vs.

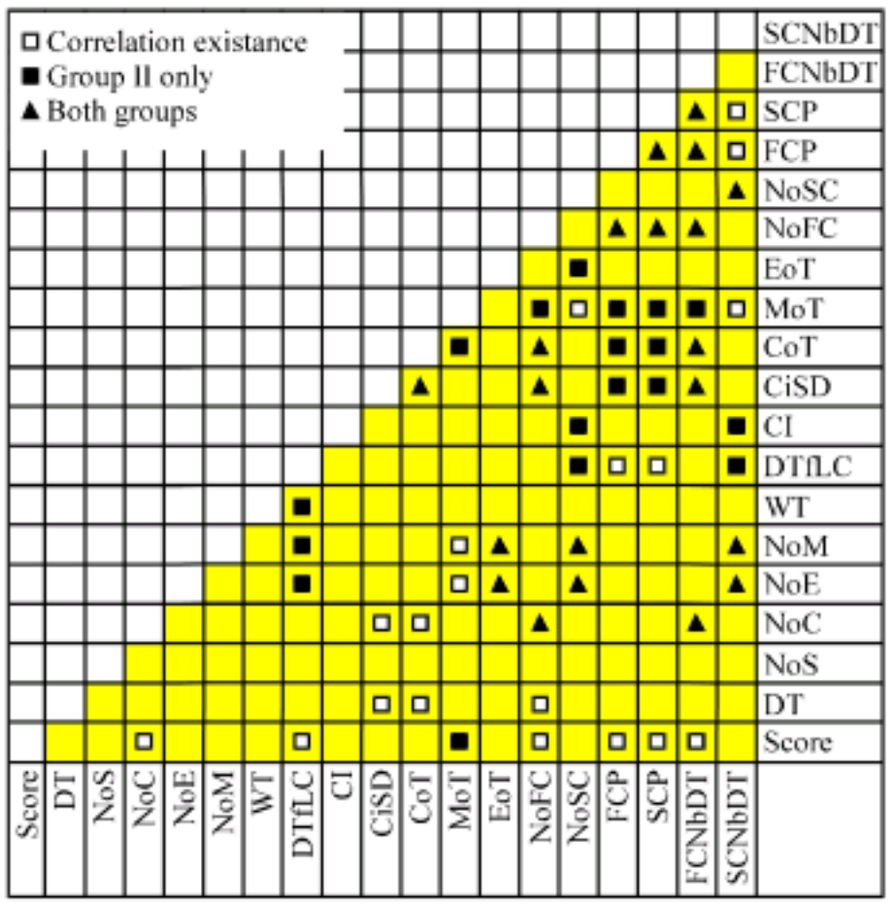


Fig. 3: High correlation among metric items

MoT (r = 0.767). There are four relationship pairs that involve MoT. Besides with Score, the other pairs are with NoFC (r = -0.891), FCP (r = -0.895), SCP (r = 0.895) and FCNbDT (r = -0.919).

The reason for focusing on non-plagiarized and plagiarized groups is due to the absence of clear pattern that can be observed regarding score if subjects from both groups are combined together. Only when subjects are segregated in their each plagiarized and non-plagiarized group, relationship between score and other items are found. This finding also justifies the use of experimental approach in this study. By comparing the control and experimental groups, the research could identify the effect of plagiarized activity on score.

Correlation results from Table 2 to 3 are summarised as in Fig. 3. The correlation graph shows significant relationship between items in behavior metric for non-plagiarized, plagiarized and both groups. These results are discussed shortly in the next section.

## DISCUSSION

The discussion starts with results from Table 1 (mean comparison between non-plagiarized and plagiarized groups). There is significant difference in programmers' score. This shows that plagiarized students have higher programming score based on program similarity to the solution than non-plagiarized students. Obviously plagiarized students take advantage from their counterpart by scoring higher in problem solution. To improve results is one of the reasons for cheating in programming (Sheard and Dick, 2003), but this score is not the only yardstick to assess novice programmer.

In terms of development time, students that did not plagiarize need longer development time compared to plagiarized students. In plagiarized group, students spend less time to develop a program but get higher score than their counterpart. Non-plagiarized students in the first group show diligence by scoring higher mean in many programming activities like writing, compiling, modifying and implementing program.

In general, the results show that mean of the first group (non-plagiarized group) exceeding the second group in most items in the metric. The exceptions are for score and Successful Compilation Percentage (SCP). However, the mean difference between the groups is not significant in some items such as successful compilation percentage and Failed compilation percentage (SCP and FCP). This findings explains that students' successful compilation or failed percentage is similar regardless they plagiarized or not.

Many results regarding programming score were sought from Table 2, that concern with non-plagiarized group. High correlation was found between score and Compilation number (NoC). Students in this group would score higher if the compilation number is high. Very high and positive relationship between Successful Compilation Percentage (SCP) and score, shows that student which possess high successful compilation percentage tend to have high scores. This finding is related to the behavior pattern suggested by Jadud (2005). If a student's typing event(s) reduce the number of errors after the initial compilation, the student is likely to end up with a running program.

On the other hand, very high and negative correlation between Failed Compilation Percentage (FCP) and Score means high failed compilation percentage lead students in this group to lower scores. This result is consistent with Jadud's (2005) observation: students who encounter more errors than a specified threshold are likely to end up with an unsuccessful program. The relationship between Development time from last compilation (DTfLC) and score shows that students who spend much time after last compilation can produce program similar to the programming solution.

All of six significant relationship pairs that involve score also concern with compilation related items i.e., Compilation number (NoC), Development time from last compilation (DTfLC), Failed compilation number (NoFC), Failed Compilation Percentage (FCP), Successful Compilation Percentage (SCP) and Failed compilation normalization between development time (FCNbDT). This result shows that among non-plagiarized students, they

should compile their program effectively to generate a good programming assignment and the successful percentage in this activity would result in good program. Students that spend more time after last compilation will likely be able to enhance the quality of their problem solution.

Results for plagiarized group are discussed by referring to Table 3. Modification time (MoT) has high and positive correlation with Score (r = 0.767). In plagiarized environment, students that spend much time modifying their program tend to have high programming score. This strong relationship indicate that if the modification time of a student is high, the chance to get high score assessment is also high. Those plagiarized students will spend a great deal of time to do modification on the available code. However, this relationship is unclear in the first group. Programming modification phase in the first group aims to increase program code and expand the program to suit the problem.

Observation in the second group (plagiarized group) shows high correlation between Modification time and compilation-related items. Very high and negative correlation was found between Modification time (MoT) and Failed compilation number (NoFC) (r = -0.891). Negative relationship gives interpretation that if modification time is high, failed compilation number will tend to be low. This happened because in plagiarism case, modification process may result in code modification or addition with determined impact, as such successful compilation of new or modified code is high.

This explanation is strengthened by a very strong and positive relationship between Modification time (MoT) and Successful Compilation Percentage (SCP) ( r= 0.895). In the second group, Successful compilation number (NoSC) and Compilation Interval (CI) have high negative association. This explains the situation in a plagiarism atmosphere; if compilation interval is extended, then successful compilation number tend to be low. Low compilation interval results in high successful compilation. This is synonym with novice programmers' behavior to adjust and compile the program upon finding some small mistakes. On the other hand, if the programmer corrects the whole error, the successful compilation number will be low. This situation is projected in plagiarized group but not in non-plagiarized group. In plagiarized group, increasing compilation number will increase successful compilation. Since, this relationship is closely related to successful compilation, we will see how number of successful normalization has rather same relationship with compilation interval.

In plagiarized group, strong negative relationship exists between Development time from last compilation (DTfLC) and Successful compilation number (NoSC) (r = -0.714). This pair is able to give some explanation because the trend is different between the groups. In plagiarized group, students with low successful compilation number tend to spend much time after the last compilation. On the other hand, high successful compilation number lead to low allocation time after last compilation. When students get some solution that they could copy, they would try to compare their own performance with the copied program code. If successful compilation is high, there is no need to test or check the total program.

This phenomenon is seen to be different in non-plagiarized group (r = 0.329). Although, the correlation value shows low relationship (not presented in Table 2), this result is discussed here because this pair shows different pattern between these groups. Positive relationship in the result suggests that non-plagiarized student still need to test the program design with the previous one although successful compilation is obtained. This situation may be contributed by their uncertainty on the proposed program and motivation factor which gives positive impact on their performance.

Results from correlation analysis that were shown in Fig. 3 give a clear picture about the relationship pattern. From the results on plagiarized group, we can see that students in this group score high in programming if they spend more time in modifying program. If they spend more time in modifying program, they tend to get low number in failed compilation. The more they spend in modifying program, their chance to compile successfully is high and to fail is low. Further, the more they modify the program, the less they will spend in development time from last compilation.

If their successful compilation number is small, their compilation interval is high. These students will spend long time in modifying the program that makes a big compilation interval. If their successful compilation number is high (and consequently execution number is high), they will not spend much time after compilation to develop the program more. This is also related to program execution, once the program is executable, the students will be satisfied with the answer that match the suggested solution and stop developing the program further. However, if their compilation fails, they will spend long time to develop their program, as much time as they spend in writing the program. These students compile their program minimally, in contrast to their tendency to modify their program maximally, as suggested by the negative direction in the relationship between these behaviors.

These patterns show us that plagiarized group will tend to modify the program and spend much time in this

activity to get their program works. Once they compile their work after spending quite long time at writing it, soon they found some syntax errors which lead them to modify the program. At this time, they are very sure that the program should have run as suggested by the provided solution. They will compile the program very seldom, as they give much attention and confidence to the suggested solution.

In non-plagiarized group, students get higher sore if they compile their program as many time as they could. They need a lot of effort and hard work to free the program from syntax errors. Program compilation and debug is one of the most efficient ways of learning programming. In this activity, novice programmers could learn the correct working code and see the programming implementation (Ahmadzadeh *et al*., 2007). Once students successfully compile the program and improve the syntax, they will be spending enough time to develop the program that fulfills the requirements in the question. The time they spend to develop the program from last compilation contributes to their programming score. Their score also depends on their ability to succeed in compiling the program including its normalization. After repeating the compilation and modification processes for a number of times, they will have a fair percentage of successful compilation over the attempts they have made, as much as their plagiarized students counterpart could get. The higher successful rate will contribute to higher mark in the programming score.

These non-plagiarized students obviously spend more time in developing the program, compare to their counterpart. Their programming activities involve a lot of program compilation. The more they repeat the compilation process, the longer compilation time that will be recorded from their whole programming session. The whole programming session that they get involved in is a good chance for them to observe the programming syntax, especially from their mistakes. Since they are newly exposed to the programming skills, they will be in a try and error mode in the learning process, in which their failure in compilation process lead to a longer time in program development. In future, when these students get used to the skills, they will compile less as they progress (Jadud, 2005).

From the above discussions on unique behavior pattern in each group, some learning pattern could be observed here. In plagiarized group, these novice programmers spend quite long time in writing the program, probably the whole program, as they have seen the complete solution. They are convinced with the given answer and will follow rigidly the suggested answer. Next, they compile the written program with a high hope that it will work. Unfortunately, they need to modify the program and improve the syntax. Again, they spend quite long time in modifying the program and keep referring to the solution to free some syntax error. They know that they can accomplish the goal, i.e., to run the program successfully. They do not need to compile the work many times, because once they follow the suggested solution, they can produce the executable program successfully.

In this group, students might consider developing their own solutions and try to come with their own idea. However, with the minimum time that they need to perform together with lack of experience, they have a very high tendency to copy (Sheard and Dick, 2003). This group will be following similar approach in problem solving and rigidly adhere to the available program. Though they learn some programming techniques and guidance from lectures, the solution copy make them loss their discretionary judgment.

Students in non-plagiarized group work in a normal novice programmer environment. The laboratory test to produce the assignment is a good chance to enhance their interest, knowledge and skills especially in a certain programming language. These students start trying to develop new program on their own. They have some difficulty in troubleshooting errors in the program which require a lot of compilation process during program development. They will write a small program at a time, compile it and gradually improve the syntax and logic which finally produce the expected results.

They try to use the programming guidelines gained from lectures yet they are not enough to understand the whole programming concept and practice. This led them to spend long time in program development. They are building confidence gradually by solving a small part in the program and move to another part which were integrated together. When they are experiencing this programming process, they would be ready to become an advanced novice programmer.

## CONCLUSION

This study examines novice programmers' behavior in developing program. Experimental study was conducted to compare behavior patterns during programming between subjects who plagiarized and who don't. The study found that plagiarized group score higher in programming and spend less time in developing program compare to non-plagiarized group. Though non-plagiarized group experience more compilation failure then their counterpart, their failed compilation percentage has not much difference from their counterpart, as non-plagiarized group has also more compilation number.

In plagiarized group, a unique pattern was found in which students spend much more time to modify program to score higher marks. Students seldom compile their work and would be satisfied with the program once it is executable. In its counterpart, students score higher programming mark if they compile their program many times, spend long time in developing the program after last compilation and be able to successfully compile the program.

Students who work own their own have to work harder to come with the solution, justify the programming logic and free the program from syntax error. Though they are facing difficulty in detecting and correcting syntax errors, they are able to develop their confidence gradually and expanding their familiarity, creativity and interest. On the other hand, copycat students might face dilemma to try their own idea within the limited time and knowledge to produce executable program. Their tendency to copy leads them to be rigid to one approach in problem solving and lack of creativity.

All students should be encouraged to expand their skills and interest in programming and the safest way is through practice. Findings from research studies like this one should be shared with novice programmer that they will be lag behind in important programming skills such as developing logical program and detecting and correcting syntax errors. Research studies to enhance the effectiveness of teaching and learning process in programming courses are needed. Such an example is study by Shuhidan *et al*. (2009).

## REFERENCES


Ahmadzadeh, M., D. Elliman and C. Higgins, 2007. The impact of improving debugging skill on programming ability. ITALICS, 6: 72-87.

Bugarín, A., M. Carreira, M. Lama and X.M. Pardo, 2009. Plagiarism detection using software tools: A study in a Computer Science degree. http://www.eunis.dk/papers/p90.pdf.

Coakes, S.J., 2005. SPSS Analysis without Anguish. John Wiley, Queensland, ISBN: 0470807369.

Daly, C. and J. Horgan, 2005. A technique for detecting plagiarism in computer code. Comp. J., 48: 662-666.

Green, T.B., 2007. A statistical analysis of the utilization effectiveness of a PERT program. Decis. Sci., 4: 426-436.

Hair, J.F., W.C. Black, B.J. Babin, R.E. Anderson and R.L. Tatham, 2006. Multivariate Data Analysis. 6th Edn., Prentice-Hall Inc., Upper Saddle River, New Jersey, USA.

Jadud, M.C., 2005. A first look at novice compilation behaviour using blue. J. Comp. Sci. Educ., 15: 25-40.

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

Prechelt, L., G. Malpohl and M. Phillippsen, 2001. Jplag: Finding plagiarism among set of program. Universitat Karisruhe, Germany. Technical Report 2000-01. http://page.mi.fu-berlin.de/prechelt/Biblio/jplagTR.pdf.

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, Jun. 30-Jul. 2, ACM, New York, pp: 45-49.

Shuhidan, S., M. Hamilton and D. D'Souza, 2009. A study of novice programmer responses in summative assessment. Proc. Aust. Comput. Educ. Conf., 95: 147-156.

Spinellis, D., P. Zaharias and A. Vrechopoulos, 2007. Coping with plagiarism and grading load: Randomized programming assignments and reflective grading. Comp. Appl. Eng. Educ., 15: 113-123.

Tsai, S.W. and N.F. Pohl, 2007. A comparative study of the effects of lecture and Computer-aided instruction on student achievement in computer programming classes. Decision Sci., 9: 291-309.

Xu, S. and X. Chen, 2005. Pair programming in software evolution. Proceedings of the Canadian Conference on Electrical and Computer Engineering, May 1-4, Laurentian University, Ontario, Canada, pp: 1846-1849.