Predictors of Errors of Novice Java Programmers

Rex P. Bringula (Corresponding author)
College of Computer Studies and Systems, University of the East
2219 C. M. Recto Avenue, Manila, Philippines
Tel: 632-735-54-71 to 82 loc. 425 E-mail: rex_bringula@yahoo.com

Geecee Maybelline A. Manabat
College of Computer Studies and Systems, University of the East
2219 C. M. Recto Avenue, Manila, Philippines
Tel: 632-735-54-71 to 82 loc. 425 E-mail: gcee91805@yahoo.com.ph

Miguel Angelo A. Tolentino
College of Computer Studies and Systems, University of the East
2219 C. M. Recto Avenue, Manila, Philippines
Tel: 632-735-54-71 to 82 loc. 425 E-mail: miguelangelotolentino@yahoo.com

Edmon L.Torres
College of Computer Studies and Systems, University of the East
2219 C. M. Recto Avenue, Manila, Philippines
Tel: 632-735-54-71 to 82 loc. 425 E-mail: emon.torres@gmail.com

Received: October 15, 2011 Accepted: November 12, 2011 Published: February 1, 2012
doi:10.5430/wje.v2n1p3 URL: http://dx.doi.org/10.5430/wje.v2n1p3

This paper is funded by the University of the East.

Abstract

This descriptive study determined which of the sources of errors would predict the errors committed by novice Java programmers. Descriptive statistics revealed that the respondents perceived that they committed the identified eighteen errors infrequently. Thought error was perceived to be the main source of error during the laboratory programming exercises. Factor analysis showed that there were five categories for the types of errors committed. Four of them were symbol- or keyword-related errors (Invalid symbols or keywords, Mismatched symbols, Missing symbols, and Excessive symbols) and the fifth one was Naming-related error (Inappropriate naming error). Regression analysis showed that Sensorimotor and Habit errors, together with Knowledge error, were found to predict Mismatched symbols and Missing symbols errors, respectively. Knowledge error was found to be the consistent source of the five types of errors. Thus, the null hypothesis stating that sources of errors do not predict errors committed by novice Java programmers is partially rejected. The implications of the findings were also discussed.

Keywords: Error, Java educator, Java programmer, Novice programmer, Taxonomy of Error

1. Introduction

One of the most successful programming languages in the market is Java (Wong, 2002). Java is considered as a general-purpose programming language that reduces the compilation cycle and enables codes to be run on multiple operating systems on any certified Java Virtual Machine (Sun Microsystems, 2008). It is a robust language that can be used for a variety of applications such as client-side software that incorporates sophisticated graphical user interfaces (Benander et al., 2004). Application developers can use it and develop different application software or even different

web applications. As it becomes more popular these days, applications developed from it become in demand and so are the programmers and developers using it.

Seeing the demand for such skill, Philippine higher education institutions (PHEIs) offer degree and non-degree programs in Information Technology Education (Computer Science, Information Technology, Information System, and Associate in Computer Technology). Most PHEIs offer Java as their introductory course in programming. Programming from its simplest definition is an act of creating a program. However, learning to program is difficult (Robins et al., 2003; Pendergast, 2005). Inexperienced programmers, also called novice programmers, have experienced varied difficulties in learning a programming language.

It is important to study novice programming errors since the study "can lead to a better understanding of problem-solving strategies and will highlight the difficult aspects of programming and programming instruction" (Ebrahimi, 1994, p. 457). As a result, numerous studies have been conducted to identify the difficulties experienced by novice programmers and the errors associated with these difficulties (e.g., Mosemann and Weibendeck 2001; Jackson et al., 2005; Gobil et al., 2009).

However, none have yet studied *what predicts* the errors committed by novice programmers. Thus, this study has been conceived. Toward this aim, it sought answers to the following questions. (1) How do the respondents perceive the sources of errors in Java programming in terms of Knowledge error, Memory error, Thought error, Habit error, Sensorimotor error, and Omission error? (2) What are the Java programming errors committed by novice Java programmers? (3) Do the different sources of errors, singly or in combination, predict the novice Java programmer's errors?

2. Literature Review

2.1 Error and Taxonomy of Error

Broadly, error is non-attainment of a goal (Zapf et al., 1992). Specifically, human error is defined as "a deviation from normal or expected performance, the deviation being defined by the consequence" (Whittingham, 2004, p. 3). It is also defined as the discrepancy between the desired and the actual performance of a person in pursuit of the attainment of the goal (Meister, 1989). For instance, if a person presses keys at random (i.e., the action is not purposeful (Whittingham, 2004)), an error cannot be committed (Zapf, et al., 1992; Brodbeck et al., 1993).

Most accidents occur due to human error (Peters and Peters, 2006). Such error accounts for the accidents in aviation (Wiegmann and Shappell, 1997; Rantanen, 2006; Garrett and Teizer, 2009), in computer and utility system operations (Koval and Floyd II, 1998), driving (Stanton and Salmon, 2009), and in petrochemical, healthcare, construction, mining, and nuclear power industries (Garrett and Teizer, 2009). The effects of human error range from damaging equipment and property, disrupting scheduled system operation, or even causing injury or fatality (Koval and Floyd II, 1998). For this reason, it has been a widely used subject of research (e.g., Kim et al., 2004; Joyce et al., 2005; Itoh et al., 2009; Stanton and Salmon, 2009).

There are articles that attempted to explain human error. Poska (2009) said that it is associated with carelessness, lack of attention, misjudgment, forgetfulness, executing tasks out of sequence or doing things outside the required time frame. Meister (1989) described it further in terms of (1) the nature of the error (errors of omission, commission, sequence, slips, and mistakes), (2) the stages in which the error occurred (e.g., takeoff, landing, installation error, design error), (3) the function with which the error is associated (e.g., decision making, tracking), (4) the procedure in relation to which the error was made, (5) equipment and/or part of the facility with which the erroneous action is associated, (6) assumed cause of error (e.g., lack of motivation, lack of skill, inattention) and (7) the consequences of error (e.g., catastrophic effect, no effect at all). Whittingham (2004, p. 4) associated it with the cognitive processes of a person as it commented that "human error is a failure of the cognitive (or thinking) processes that went into planning an action or sequence actions, a failure in the execution of the action or a failure to carry out the action at all." This is called endogenous error (Whittingham, 2004).

Others attempted to explain human errors through error classification (taxonomy of errors). Phenotypes and genotypes (Hollnagel, 1993 cited in Sutcliffe and Rugg, 1998, and in Whittingham, 2004) is the broadest taxonomy of human errors. Genotype deals with the underlying cause of the error at the cognitive level while phenotype deals with the observable effect of the error (Sutcliffe and Rugg, 1998; Whittingham, 2004). Meanwhile, Rasmussen (1983 cited in Whittingham, 2004) proposed that errors can be explained through skill-, rule-, and knowledge-based behaviors. Whittingham (2004) explains that (1) skill-based behavior is used when a usual and routine task has to be carried out in an automated fashion without a great deal of conscious thought, (2) rule-based behavior is acquired from experience or through formal training and which is retrieved from memory when a task is executed, and (3) knowledge-based behavior

is executed when a new and unfamiliar situation is presented for which no existing rules are stored but requires a plan of action.

Zapf et al. (1992) and Brodbeck et al. (1993) classified errors in the field of human-computer interaction. In their studies in the office workplace using computers, they proposed two taxa. The first taxon is the functionality problem in which there is a mismatch between the work task and the computer (Zapf et al., 1992; Brodbeck et al., 1993). The second taxon is usability problem in which there is a mismatch between the user and the computer (Zapf et al., 1992; Brodbeck et al., 1993). The second taxon is subdivided further into knowledge-based for regulation, intellectual level of regulation, level of flexible action patterns, and sensorimotor level of regulation.

Knowledge error is the sole component of knowledge-based for regulation. Thought, Memory, and Judgment errors are the components of intellectual level of regulation. Level of flexible action patterns is composed of Habit, Omission, and Recognition errors. Sensorimotor error is the only element of Sensorimotor level of regulation.

Zapf et al. (1992) defined these errors as follows.

- Knowledge errors occur when a specific task is not executed because one does not know certain commands, function keys, rules, etc.
- Thought errors, on the other hand, occur due to inadequacy of the developed plan or wrong decisions are made in the assignment of plans or sub-plans even though the user knows what to do.
- Memory errors occur when the plan is forgotten and not executed although the goal and the plan are correctly identified.
- Judgment errors happen when the user cannot understand or interpret the computer message or feedback.
- Habit errors mean that a correct action is carried out in a wrong situation.
- Omission errors arise when a plan is not executed although the plan is done routinely.
- Recognition errors arise when an error message is unnoticed or is confused with another one.
- Sensorimotor errors are manifested when a wrong key is pressed or a wrong mouse click is executed.

2.2 Novice Programmer

Programming is one of the core skills for Information Technology Education (Computer Science, Information Technology, Information Systems, and Associate in Computer Technology) students. It is a very useful and rewarding career (Robins et al., 2003). However, learning to program is hard (Robins et al., 2003; Pendergast, 2005). It requires exceptional perfection (Perkins and Martin, 1986; Rogerson and Scott, 2010). Programming ability requires a strong foundation about knowledge on computers, programming languages, programming tools and resources, theory and formal methods (Robins et al., 2003). More specifically, programming involves putting the pieces together of a set of programming language instructions that solves a specific problem (Pennington and Grabowski, 1990).

Novice programmers are programmers who are inexperienced in the art of programming and usually taking up introductory programming course (Gobil et al., 2009). Similarly, Shuhidan et al. (2009) defined them as those who never had programming experience to those who may had some basic background to programming either attained formally or informally in a pre-university setup. Norman (1978 cited in Palumbo, 1990) believes it may require 5,000 hours to develop a complex skill such as programming. Simon (1980 cited in Palumbo, 1990) said that it may require 10,000 hours to build expertise in a particular area. But according to Winslow (1996 cited in Robins et al., 2003), it takes 10 years of experience to develop the skills of novices to become experts. Nevertheless, it is clear that "if participants in programming language/problem-solving research were novices at the beginning of the programming instruction, they would still be classified as non-experts at the conclusion of the instruction" (Palumbo, 1990, p. 70).

Novice programmers lack of programming knowledge (Robins et al., 2003) and strategies (Robins et al., 2003; Carbone et al., 2009). They are less skilled at using functional information in spite of the simplicity and appropriateness of the program to their own level (Wiedenbeck, 1986). They have difficulty tracing one or more variables in order to see how these are transformed in the program (data flow view) (Mosemann and Wiedenbeck, 2001; Carbone et al., 2009) due to their "line-by-line" approach to programming (Gobil et al., 2009; Winslow, 1996 cited in Robins et al., 2003). They have limited debugging skills (Carbone et al., 2009), and have incomplete understanding of language constructs such as variables, loops, arrays, and recursions (Gobil et al., 2009). Winslow (1996 cited in Robins et al., 2003, p. 140) also said that novices are "limited to surface and superficially organized knowledge, lack detailed mental models, fail to apply relevant knowledge, and approach programming "line by line" rather than using meaningful program chunks or structures."

In order to understand more about novices, they are usually compared to experts since "by definition novices do not have many of the strengths of experts" (Robins et al., 2003, p. 140). Experts are good at structuring changes in a program and know how to integrate these changes with the existing codes (Mosemann and Wiedenbeck, 2001). They are also good at recognizing, using, and adapting patterns or schemas (Robins et al., 2003). They employ problem-solving strategies such as modular decomposition, analogical reasoning, systematic planning, coding and debugging (Kurland et al., 1986).

Every novice programmer is confronted with a wide range of difficulties and deficits in programming (Robins et al., 2003). First, novices have difficulty in expressing natural language solutions into computer programming language solutions (Ebrahimi, 1994; Kelleher and Pausch, 2005). They might know the syntax and semantics of the individual statements of the programming language but they are unable to put them together into a valid program (Winslow 1996, cited in Robins et al., 2003). Second, they have problems in analyzing and designing mathematical expressions, naming variables and assigning suitable data types and structures to these variables, evaluating correctly output statements, arithmetic expressions, and relationship expressions (Gobil et al., 2009). Lastly, they also have difficulty in debugging loop conditions, conditional logic, arithmetic errors, and data initialization and update (Fitzgerald et al., 2008).

2.3 Research Paradigm

The foregoing review of related literature served as basis in the formulation of the research paradigm. The study was guided by the taxonomy of error proposed by Zapf et al. (1992). However, taxonomy of error was modified to fit the taxa in programming domains. The modified seven constructs were identified below.

<Figure 1 about here>

The following constructs served as the independent variables. These were defined as follows.

- Knowledge error refers to the acquired knowledge in programming of the students in the classroom lecture. The content of the lecture encompasses syntax, simple mathematical operators, logical operators, conditional statements, use of classes and methods in Java.
- Memory error refers to the errors committed due to failure to remember a) the syntax of Java programming language, b) lecture or instructions of the teacher, c) use of specific mathematical operator, and d) use of methods in Java.
- Thought error refers to the misconception of the acquired knowledge in Java programming. This was manifested by the presumption that the code was correct but once but once compiled a syntax error was found.
- Judgment error occurs when students cannot understand the compiler error message or cannot debug the error. Judgment and recognition errors were treated as one in this study.
- Habit error occurs when students tend to ignore the lecture or the instructions of the teacher.
- Sensorimotor motor refers to unintentional inclusion of unnecessary characters in the program text or unintentional key press.
- Omission error occurs when students skipped the instructions or reading the lecture given by the teacher.

The dependent variable (errors committed by novice Java programmers) was composed of eighteen (18) errors (e.g., No semi-colon at the end of a statement, Excessive semi-colon, Putting a period between the keyword *import* and java packages, etc.). Self-reporting method (Meister, 1989) was used to determine the frequency of errors committed. The measurement, validity, and reliability of the constructs were discussed in details in the following section.

In light of the research paradigm, it is hypothesized that sources of errors, singly or in combination, do not significantly predict errors committed by novice Java programmers.

3. Methodology

3.1 Research Design, Locale, and Subjects

The study used a descriptive design which employed a descriptive survey as the research instrument. The College of Computer Studies and Systems of the University of the East was selected as the research locale of the study. The study adopted the definition of Gobil et al. (2009) and Shuhidan et al. (2009) of novice programmers. Thus, first year students who were enrolled in Introductory Computer Programming Course in Java Programming (PROG1) of First Semester of School Year 2010-2011 were the respondents of the study.

3.2 Sampling Design

There were 598 students enrolled in PROG1. Eighty-one (81) students participated in the experiment (programming activity) and forty (40) in the pre-test of the questionnaire. This number of students (121) was deducted from the total population enrolled in PROG1. Thus, the true population considered in the study was 477. Using Sloven's formula with a sampling error of 0.05, the computed sample size was 217. Respondents were randomly selected through their class sections in PROG1.

The sample size was increased to 322 (a 48% increase) to accommodate low return rate. Two hundred fifty-three (253) forms were retrieved and these were all used in the study. The details of the distribution of the survey forms were given in Table 1.

<Table 1 about here>

3.3 Research Instrument and Data-Gathering Procedure

A survey form was utilized as the research instrument of the study. There were two (2) phases in the construction of the survey form. The first phase was to determine the items to be included in the survey form. An experiment, which involved a laboratory programming activity, was conducted in three (3) sections (with a total of 81 students) of PROG1.

Before the experiment was conducted, three experts validated the complexity, relevance, and appropriateness of the programming problem given (application of an *if-else* structure) in the activity. These experts were composed of two (2) professors with at least 10 years of teaching experience in programming and one (1) technical expert (Certified Java Programmer). They agreed that the problem could be solved within 60 minutes by novice programmers.

The programming problem was related to the lecture. The students were also asked to take screenshots of their compilations in order to log their common errors in programming. The screenshots were then pasted in a word processer. Errors committed were tabulated based on the screenshots.

The screenshots were used as basis in the formulation of questions with regard to errors committed by novice Java programmers. Eighteen questions were developed based on these screen shots. Data collected on the eighteen questions served as dependent variable of the study.

After the questionnaire was developed, it was pretested to 40 students who were excluded in the sample. This is the second phase of the survey construction. To determine the frequency errors committed, respondents could answer from 1 (Never) to 5 (Always). The weight and verbal interpretation of this construct are given below (Table 2).

<Tables 2-4 about here>

There were seven (7) constructs that could be attributed to the errors committed by novice Java programmers. Respondents were asked to rate each question on the hypothesized constructs from 1 (Highly disagree) to 10 (Highly disagree). Each question began with the statement "*I do not know…*", "*I forgot the …*", "*I thought …*", "*I cannot …*", "*I always do not…*", "*I accidentally …*", and "*I skipped …*" for Knowledge Error, Memory Error, Thought Error, Habit Error, Sensorimotor Error, and Omission Error, respectively. However, Cronbach's alpha analysis showed that Judgment error ($\alpha = 0.690$) (see Table 3) was not found to be a reliable construct (below the minimum criterion of 0.70) (Pallant 2001; George and Mallery, 2009). Thus, only six constructs were retained. The retained constructs were found to be reliable (above the minimum criterion of 0.70) and valid (factor loadings greater than 0.40) (Pallant 2001; George and Mallery, 2009).

3.4 Statistical Tools Used

The study used frequency count, percentage, means, and ranking to describe the data. Cronbach's alpha analysis and factor analysis were utilized to determine the reliability and validity of the constructs, respectively. Multiple regression analysis at 5% level of probability and 95% reliability was employed to determine which of the sources of error would predict the errors committed by novice Java programmers.

4. Results and Discussion

4.1 Java Programming Errors Committed

The respondents of the study are taking up degree programs in Information Technology (*f*=194, 77%) and Computer Science (*f*=54, 21%), and a non-degree program in Computer Technology (Associate in Computer Technology, *f*=5, 2%). Table 4 shows the factor analysis of the eighteen (18) novice Java programming errors.

Five types of errors (i.e., factors) are retained since their eigenvalues are greater than 1.00 (Dancey and Reidy, 2002). The cumulative percentage of variance is 56.346%. All variables loaded highly on each type of error (greater than 0.40).

The first type of error extracted is called **Invalid symbols or keywords** (overall mean rating = 1.98). It has the highest eigenvalue of 5.209. This is composed of errors such as No period between class name and method name, Capitalized keywords, Replacing (and) with < and > or [and] in output stream, and *else* without *if*. The second type of error is related to **Mismatched symbols** (eigenvalue = 1.445; overall mean rating = 2.42). These errors are due to Unmatched curly braces, Incorrect *greater than* or *equal to* sign, Cannot find symbol because of mismatched between the declared and used variable, and Cannot find symbol because of undeclared variable. Meanwhile, No semi-colon at the end of a statement, No close/open parenthesis on if condition, No parentheses on *if*-condition, and Unclosed literals are the errors under **Missing symbols** (eigenvalue = 1.281; overall mean rating = 2.03).

Inappropriate naming (eigenvalue = 1.135; overall mean rating = 2.22) is the fourth type of error that is composed of errors such as Inappropriate casing of method names, Inappropriate casing of class names, and Splitting a class name by putting a space. **Excessive symbols** (eigenvalue = 1.072; overall mean rating = 2.11) are the last type of errors. These are composed of Excessive semi-colon, Putting a period between the keyword *import* and java packages, and Putting a semi-colon after the *if*-condition. All types of errors are below the lower half of the scale (Invalid symbols or keywords, over all mean rating = 1.98; Mismatched symbols, overall mean rating = 2.42; Missing symbols, overall mean rating = 2.03; Inappropriate naming, overall mean rating = 2.22; and Excessive symbols, overall mean rating = 2.11).

The findings of this study are similar to those of the studies of Jadud (2005), Jackson et al. (2005), and Gobil et al. (2009). Jadud (2005) reported that out of 1,926 errors encountered by the students, more than half of all errors generated by students while programming are missing semi-colons, unknown symbol-variable, illegal start of expression, bracket expected, and unknown symbol-class. Jackson et al. (2005) also reported that novice programmers also encountered the following errors: cannot resolve symbol, semi-colon expected, illegal start of expression, class or interface expected, <identifier> expected,) expected, incompatible types, int, not a statement, and } expected. Gobil et al. (2009) also found out that the most common errors are basic mechanic symbols (braces, brackets and semi-colons, formatting outputs and indenting), and incorrect and irrelevant naming of variables or constants. The findings of Jadud, Jackson et al., and Gobil et al. are extended by classifying them into types of errors committed shown in Table 4.

It must be noted that the study was conducted in a laboratory experiment. The findings therefore have profound implications on educating the novice programmers. First, the finding stressed the importance of teachers (Rogerson and Scott, 2010) as they play an important role in "curing" these errors during laboratory exercises. The findings provide Java programming teachers an insight on understanding the errors committed by novice Java programmers. Shuhidan et al. (2009, p. 148) commented that "improved understanding of novice errors will also better inform educators about alleviating the difficulties experienced by novices at commencement." Thus, teachers can expect that their students (i.e., novice Java programmers) will always encounter these errors and therefore can be readily equipped to correct these mistakes.

The second implication is that, since more than 50% in the variation (cumulative percentage of variance = 56.346%) of the types of errors committed is accounted to the five factors, lecture hours and laboratory exercises can be focused on these types of errors. Thus, strong focus should be given on the discussion on these errors since it could greatly enhance the performance of the students in programming.

Consequently, Introductory Java programming syllabus can be designed in a way that tackles these topics. This could elevate the teaching standards of Java teachers. This is very important since teaching standards can influence the outcomes of courses that teach programming (Linn and Dalbey, 1985). This is the third implication of the study.

4.2 Sources of Errors Committed

Table 5 shows the sources of Java programming errors committed.

<Table 5 about here>

Thought error has the highest mean rating of 5.18. It is composed of six (6) questions such as "(1) I thought my syntax was correct, (2) I thought that the mathematical operator I used was correct, (3) I thought that the logical operator I used was correct, (4) I thought that the conditional statement I put in an *if*-statement was correct, (5) I thought that I used a correct class of Java, and (6) I thought I used a correct method of Java." Meanwhile, Habit error got the lowest mean rating of 3.23. Sensorimotor (mean = 4.34), Omission (mean = 3.78), and Memory (mean = 3.72) errors were rated at the lower half of the scale. The overall mean (overall mean = 3.92) shows that the six errors were rated below half of the scale.

It can be noted that Thought error refers to misunderstanding of the learned lesson in Java programming. This is manifested by writing a syntax that they assumed to be a correct syntax but in reality it is syntactically incorrect. The solution to such misconception is of course to correct it. This implies that students should be given enough, proper, and

rigorous instructions on how to translate their solutions represented in natural language statements into programming language statements.

4.3 Regression of Errors Committed by Novice Java Programmers Committed on Sources of Errors

Table 6 shows the regression of errors committed by novice Java programmers on sources of errors. Knowledge error (beta = 0.395, $p < 0.05$) is found to be the significant predictor of Invalid symbol errors. A fifteen percent (15%) (Adj. $R^2 = 0.153$) variation on invalid symbol errors of the students is attributed to Knowledge error. The result is unlikely to have arisen from sampling error (F-value = 45.931, $p < 0.05$).

<Table 6 about here>

Mismatched symbol is predicted through Knowledge (beta = 0.317, $p < 0.05$) and Sensorimotor (beta = 0.128, $p < 0.05$) errors. Knowledge error is a stronger predictor than Sensorimotor error since the former has a larger beta coefficient. Together, the predictors are accounted to 13% (Adj. $R^2 = 0.127$) in the variation of Mismatched symbol errors. The F-value of 19.17 with an associated probability of 0.000 shows that the result of the prediction is unlikely to have arisen from sampling error.

Knowledge error and Habit error predict Missing symbol type of error. Missing symbol type of error is predicted more by Habit error (beta = 0.176, $p < 0.05$) than by Knowledge error (beta = 0.134, $p < 0.043$). However, the explanatory power of both predictors is only about 6% (Adj. $R^2 = 0.058$). Nevertheless, the result is unlikely to have arisen from sampling error (F-value = 8.58, $p < 0.05$).

It must be noted that Habit error refers to the reason of error resulting from persistent inattentiveness of students on the lecture of the teacher. According to Wiegmann and Shappell (1997), error due to attention is one of the reasons of accidents in aviation. This finding shows that in the field of programming, inattentiveness of students frequently yielded to avoidable and simple error such as a "missing semi-colon." This implies that it is important to listen to the lecture of the professors.

Knowledge error (beta = 0.332, $p < 0.05$) also predicts Inappropriate naming with an associated explanatory power of about 11% (Adj. $R^2 = 0.106$). The F-value of 30.408 with an associated p-value of 0.000 shows that the result is unlikely to have arisen from sampling error. Similarly, Knowledge error (beta = 0.183, $p < 0.004$) predicts Excessive symbol type of error. Three percent (3%) (Adj. $R^2 = 0.029$) in the variation of Excessive symbol type of error is accounted to Knowledge error. This is unlikely to have arisen from sampling error, as shown by *F*-value = 8.423 with an associated probability of 0.004.

As can be seen in Table 6, Knowledge error is found to be the consistent predictor of all types of errors. The predictive power of Knowledge error ranges only from 3 to 15%. It can be noted that in this study, Knowledge error *only* refers to the knowledge that the students *know* about Java syntax, mathematical and logical operators, conditional statement formulation, and classes and methods usage. Thus, other factors can be attributed to committing the types of errors discussed since "knowledge is only part of the picture" (Davies, 1993, cited in Robins et al., 2003, p. 141).

Novice programmers, according to Perkins and Martins (1986), Robins et al. (2003), and De Raadt (2007), have fragile knowledge. This knowledge is can either be partial, inert, misplaced, or conglomerated. Perkins and Martins (1986) elaborated this knowledge as follows: (1) Partial knowledge refers to the straightforward case of an impasse since knowledge is not retained nor learned. (2) Inert knowledge is possessed by a novice programmer but fails to retrieve that knowledge during programming. (3) Misplaced knowledge is manifested when a student uses commands structures which do not fit to its intended purpose. (4) Conglomerated knowledge signifies situations where several unrelated codes are put together in a syntactically or semantically anomalous way.

In this study, only partial knowledge was investigated. Future studies can include the other three forms of knowledge discussed by Perkins and Martins (1986).

The finding of the study has also an educational implication and it also supports the study of Robins et al. (2003). Robins et al. (2003) noted that in their experience, typical introductory programming courses are "knowledge-driven," i.e., typical programming textbooks are focused on presenting knowledge about a particular language through detailed samples and exercises. Whitfield et al. (2007) also found out that previous textbooks used by the undergraduate degree were not suited for them since these textbooks assumed knowledge of problem-solving skills. The result of the regression shows that knowledge-driven textbooks are relatively effective since it can only predict 3 to 15% of the error that the students can commit during programming laboratory exercises. In other words, it can be expected that these textbooks can only avoid 3 to 15% of the errors during programming laboratory sessions.

While it is important to teach the constructs of the language, however, teachers should not only focus on the "knowledge" given by such textbooks but also on other factors that could lead to error-free programming. Davies (1993 cited in Robins et al., 2003) advocated the focus on problem-solving strategies. Programming knowledge of a declarative nature is the ability to state *how* a for loop works while programming strategies are the *ways* knowledge is used and applied (e.g., a for loop is appropriately used in a program) (Davies, 1993 cited in Robins et al., 2003). De Raadt (2007, p. 211) also commented that "programming knowledge and programming strategies, while related, need to be identified separately in curricular materials and assessment." Labuscagne (2008) reported that many teachers believed that problem-solving should be the focus of teaching to novice programmers while the programming language itself is secondary.

Winslow (1996 cited in Robins et al., 2003) showed that novice programmers use general problem-solving strategies instead of problem-specific or programming-specific strategies. A flawed strategy therefore could lead to programming errors (see Robins et al., 2003).

Furthermore, caution should be taken into consideration when writing textbooks. It cannot be denied that students should be familiarized with the syntax and semantics of the programming language. Nevertheless, textbook writers should also emphasize the strategy on how to solve problem on hand and direct the students to put the pieces together. Doing this will greatly encourage novices to become effective programmers (Robins et al., 2003).

5. Summary, Conclusions, and Recommendations

On the errors presented earlier, it was found out that respondents perceived that they only committed these errors infrequently. When these errors were grouped together, five categories were found. These were Invalid symbols or keywords, Mismatched symbols, Missing symbols, Inappropriate naming, and Excessive symbols. These errors contributed to more than 50% of the errors committed during laboratory programming exercises.

Knowledge error was found to be the consistent predictor of novice Java programmers' error. Sensorimotor error and Habit error, together with Knowledge error, were found to be significant predictors of Mismatched symbol and Missing symbol, respectively. Thus, based on the findings presented, the null hypothesis stating that the sources of errors do not predict errors committed by novice Java programmers is partially rejected.

In the light of the limitations, findings, and conclusion presented, the following recommendations are set forth. First, a study of the relationship between errors committed and the affect states and behavior while programming may be initiated. For example, it was found that there are two types of novice programmers: stoppers and movers (see Robins et al., 2003). Stoppers are those who, as the word implies, stop on programming when faced with difficulty on the task. On the other hand, movers are those who keep on trying to solve the problem on hand and, perhaps, eventually will achieve a working program.

Therefore, questions can be raised from these situations:

(1) What types of errors do stoppers and movers encounter during programming?

(2) Are sources of errors mentioned in this study attributed to the errors they committed?

(3) Is there a difference, in quantity and in nature, between the errors encountered by stoppers and those by movers? (4) What problem-solving and debugging strategies employed by movers?

(5) As educators, what can be done to help stoppers be engaged in solving the problem?

Future research can answer these questions.

Second, other variables not investigated in this study can be included in future research. The types of errors used in this study are only focused on syntax errors. Thus, semantic errors can also be investigated. Also, other forms of knowledge (inert, misplaced, or conglomerated) can be included as independent variables.

Third, Java educators should be readily equipped with the knowledge on how to avoid or to cure the types of errors found in this study. It is important to teach students the knowledge about a specific programming language but Java educators should not forget that it should be taught along with effective problem-solving strategies. Robins et al. (2003, p. 162) also recommended that "initial course material should be simple, and this should be expanded on systematically as the students gain experience." In this connection, Java textbooks should not only be "knowledge-driven" but it should also be "strategy-driven" (Perkins and Martins, 1986; Robins et al., 2003).

Robins et al. (2003, see p. 164) proposed a programming framework that summarized the programming processes. This gives a clear and vivid picture of the whole programming processes. Guided by this framework, errors committed by novice Java programmers on each programming process can be investigated.

Fourth, there were anecdotal evidences that some students preferred some programming languages (e.g, Visual Basic or C++) over Java. Hence, two further studies can be initiated – (1) a study on the factors considered in choosing programming languages, and (2) a study on the difficulties faced by novice programmers in using Java.

Finally, the big challenge for Java educators is to help Java novice programmers translate the latter's solution into an error-free program. With the help of an appropriate textbook, teachers could deliver the content of the course with great precision and caution since "a course well-experienced will leave students with good programming habits, the ability to learn on their own, and a favorable impression of programming as a profession" (Pendergast, 2006, p. 491). Doing this will greatly "educate the younger generation that an error, even a small one, means that the program is incorrect. After all, even a minor error can cause a disaster and result in a major expenditure of human life" (Kolikant and Mussai, 2008, p. 148).

References

Benander, A., Benander, B., & Sang, J. (2004). Factors related to the difficulty of learning to program in Java – An empirical study of non-novice programmers. Information and Software Technology, 46, 99–107. http://dx.doi.org/10.1016/S0950-5849(03)00112-5

Brodbeck, F. C., Zapf, D., Priimper, J., & Frese, M. (1993). Error handling in office work with computers: A field study. *Journal of Occupational and Organizational Psychology*, *66*, 303–331. http://dx.doi.org/10.1111/j.2044-8325.1993.tb00541.x

Carbone, A., Hurst, J., Mitchell, I., & Gunstone, D. (2009). An exploration of internal factors influencing student learning of programming. In Md. J. Nordin, K. Jumari, M. S. Zakaria, & Suwarno (Eds.). *Computing Education 2009: Proceedings 11th Australasian Computing Education Conference (ACE 2009), Conferences in Research and Practice in Information Technology (CRPIT), 95* (pp. 25–34). Sydney, Australia: Australian Computer Society, Inc.

Dancey, C. P., & Reidy, J. (2002). *Statistics without maths for Psychology: Using SPSS for Windows 2nd edn.* England: Pearson Education Limited.

De Raadt, M. (2007). A review of Australasian investigations into problem solving and the novice programmer. *Computer Science Education*, *17*(3), 201–213. http://dx.doi.org/10.1080/08993400701538104

Ebrahimi, A. (1994). Novice programmer errors: Language constructs and plan composition. *International Journal of Human-Computer Studies*, *41*, 457–480. http://dx.doi.org/10.1006/ijhc.1994.1069

Fitzgerald, S., Lewandowski, G., McCauley, R., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: Finding, fixing and flailing, a multi-institutional study of novice debuggers. *Computer Science Education*, *18*(2), 93–116. http://dx.doi.org/10.1080/08993400802114508

Garret, J. W., Teizer, J. (2009). Human factors analysis classification system relating to human error awareness taxonomy in construction safety. *Journal of Construction Engineering and Management*, *135*(8), 754–763. http://dx.doi.org/10.1061/(ASCE)CO.1943-7862.0000034

George, D., & Mallery, P. (2009). *SPSS for Windows step by step: A simple guide and reference 16.0 update 9th edn*. Boston: Pearson Education.

Gobil, A. R. M., Shukor, Z., & Mohtar, I. A. (2009). Novice difficulties in selection structure. In Md. J. Nordin, K. Jumari, M. S. Zakaria, & Suwarno (Eds.). *2009 International Conference on Electrical Engineering and Informatics, 2,* (pp. 351–356). doi: 10.1109/ICEEI.2009.5254715. New Jersey, USA: IEEE Computer Society. http://dx.doi.org/10.1109/ICEEI.2009.5254715

Itoh, K., Omata, N., & Andersen, H. B. (2009). A human error taxonomy for analysing healthcare incident reports: Assessing reporting culture and its effects on safety performance. *Journal of Risk Research*, *12*(3/4), 485–511. http://dx.doi.org/10.1080/13669870903047513

Jackson, J., Cobb, M., & Carver, C. (2005). Identifying top java errors for novice programmers. *Frontiers in Education Conference*, *1*, T4C-T4C27. http://dx.doi.org/10.1109/FIE.2005.1611967

Jadud, M. C. (2005). A first look at novice compilation behavior using BlueJ. *Computer Science Education*, *15*(1), 25–40. http://dx.doi.org/10.1080/08993400500056530

Jenkins, T., & Davy, J. (2001). Diversity and motivation in introductory programming. *Innovations in Teaching and Learning in Information and Computer Sciences*, *1*(1). Retrieved January 9, 2011, from http://www.ics.heacademy.ac.uk/italics/issue1/tjenkins/003.PDF

Joyce, P., Boaden, R., & Esmail, A. (2005). Managing risk: A taxonomy of error in health policy. *Health Care Analysis*, *13*(4), 337–346. http://dx.doi.org/10.1007/s10728-005-8129-x

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys*, *37*(2), 83–137. http://dx.doi.org/10.1145/1089733.1089734

Kim, J. W., Jung, W., & Ha, J. (2004). AGAPE-ET: A methodology for human error analysis of emergency tasks. *Risk Analysis*, *24*(5), 1261–1277. http://dx.doi.org/10.1111/j.0272-4332.2004.00524.x

Kolikant, Y. B-D., & Mussai, M. (2008). "So my program doesn't run!" Definition, origins, and practical expressions of students' (mis)conceptions of correctness. *Computer Science Education*, *18*(2), 135–151. http://dx.doi.org/10.1080/08993400802156400

Koval, D. O., & Floyd II, H. L. (1998). Human element factors affecting reliability and safety. *IEEE Transactions on Industry Applications*, *34*(2), 406–414. http://dx.doi.org/10.1109/28.663487

Kurland, D. M., Pea, R. D., Clement, C., & Mawby, R. (1986). A study of the development of programming ability and thinking skills in high school students. *Journal of Educational Computing Research*, *2*(4), 429–458. http://dx.doi.org/10.2190/BKML-B1QV-KDN4-8ULH

Labuscagne, C. (2008). How to teach programming to novices. *Communications of the ACM*, *51*(6), 11. http://dx.doi.org/10.1145/1349026.1349029

Linn, M. C., & Dalbey, J. (1985). Cognitive consequences of programming instruction: Instruction, access, and ability. *Educational Psychologist*, *20*(4), 191–206. http://dx.doi.org/10.1207/s15326985ep2004_4

Meister, D. (1989). The nature of human error. *Global Telecommunications Conference and Exhibition*, *2*, 783–786. http://dx.doi.org/10.1109/GLOCOM.1989.64071

Mosemann, R., & Wiedenbeck, S. (2001). Navigation and comprehension of programs by novice programmers. *Proceedings of the 9th International Workshop on Program Comprehension: IWPC '01* (pp. 79–88). http://dx.doi.org/10.1109/WPC.2001.921716. Washington, DC: IEEE Computer Society.

Pallant, J. (2001). *SPSS survival manual: A step by step guide to data analysis using SPSS for Windows version 10*. Buckingham: Open University Press.

Palumbo, D. B. (1990). Programming language/problem-solving research: A review of relevant issues. *Review of Educational Research*, *60*(1), 65–89.

Pendergast, M. O. (2005). Teaching Java to IS students: Top ten most heinous programming errors. *Americas Conference on Information Systems 2005 Proceedings*, 656–667. Retrieved May 3, 2011, from http://aisel.aisnet.org/amcis2005/241

Pendergast, M. O. (2006). Teaching introductory programming to IS students: Java problems and pitfalls. *Journal of Information and Technology Education*, *5*, 491–595. Retrieved January 8, 2011, from http://jite.org/documents/Vol5/v5p491-515Pendergast128.pdf

Pennington, N., & Grabowski, B. (1990). The tasks of programming. In J.M. Hoc, T.R.G. Green, R. Samurçay, & D.J. Gilmore (Eds.), *Psychology of programming* (pp. 45–62). London: Academic Press.

Perkins, D. N., & Martin F. (1986). Fragile knowledge and neglected strategies in novice programmers. In E. Soloway & S. Iyengar (Eds.), *Empirical studies of programmers, First Workshop* (pp. 213–229). Norwood, NJ: Ablex.

Peters, G. A., & Peters, B. J. (2006). Human error: Causes and control. Boca Raton, Fl: CRC Press/Taylor & Francis.

Poska, R. (2009). Human error and retraining: An interview with Kevin O'Donnell, Ph.D., Irish Medicines Board. *Journal of GXP Compliance*, *13*(4), 47–60.

Rantanen, E. M., Palmer, B. O., Wiegmann, D. A., & Musiorski, K. M. (2006). Five-dimensional taxonomy to relate human errors and technological interventions in a human factors literature database. *Journal of the American Society for Information Science and Technology*, *57*(9), 1221–1232. http://dx.doi.org/10.1002/asi.20412

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, *13*(2), 137–172. http://dx.doi.org/10.1076/csed.13.2.137.14200

Rogerson, C., & Scott, E. (2010). The fear factor: How it affects students learning to program in a tertiary environment. *Journal of Information Technology Education*, *9*, 147–171. Retrieved April 8, 2010, from http://jite.org/documents/Vol9/JITEv9p147-171Rogerson803.pdf

Rountree, N., Rountree, J., & Robins, A. (2002). Identifying the danger zones: Predictors of success and failure in a CS1 course. *Inroads (the SIGCSE Bulletin)*, *34*, 121–124.

Shuhidan, S., Hamilton, M., & D'Souza, D. (2009). A taxonomic study of novice programming summative assessment. In M. Hamilton and T. Clear. *Computing Education 2009: Proceedings 11th Australasian Computing Education Conference (ACE 2009), Conferences in Research and Practice in Information Technology (CRPIT)*, *95* (pp. 147–156). Sydney, Australia: Australian Computer Society, Inc.

Stanton, N. A., & Salmon, P. M. (2009). Human error taxonomies applied to driving: A generic driver error taxonomy and its implication for intelligent transport systems. *Safety Science*, *47*(2), 227–237. http://dx.doi.org/10.1016/j.ssci.2008.03.006

Sun Microsystems. (2008). Java programming language student guide. California, U.S.A.: Sun Microsystems.

Sutcliffe, A., & Rugg, G. (1998). A taxonomy of error types for failure analysis and risk assessment. *International Journal of Human-Computer Interaction*, *10*(4), 381–405. http://dx.doi.org/10.1207/s15327590ijhc1004_5

Whitfield, A. K., Blakeway, S., Herterich, G. E., & Beaumont, C. (2007). Programming, disciplines and methods adopted at Liverpool Hope University. *Innovations in Teaching and Learning in Information and Computer Sciences*, *6*(4), 145–168. Retrieved April 8, 2011, from http://www.ics.heacademy.ac.uk/italics/vol6iss4/Whitfield.pdf

Whittingham, R. B. (2004). *The blame machine: Why human error causes accidents*. Elsevier Butterworth-Heinemann: Burlington, MA.

Wiedenbeck, S. (1986). Organization of programming knowledge of novices and experts. *Journal of the American Society for Information Science*, *37*(5), 294–299.

Wiegmann, D. A., & Shappell, S. A. (1997). Human factors analysis of postaccident data: Applying theoretical taxonomies of human error. *The International Journal of Aviation Psychology*, *7*(1), 67–81. http://dx.doi.org/10.1207/s15327108ijap0701_4

Wong, W. (2002). Self-inflicted wounds may scar Java. Retrieved May 10, 2011, from http://www.zdnet.com/news/self-inflicted-wounds-may-scar-java/121683?tag=content;search-results-rivers

Zapf, D., Brodbeck, F. C., & Frese, M. (1992). Errors in working with office computers: A first validation of a taxonomy for observed errors in a field setting. *International Journal of Human-Computer Interaction*, *4*(4), 311–339. http://dx.doi.org/10.1080/10447319209526046

Table 1. Number of Survey Forms Distributed and Retrieved

Section[a]	**No. of Forms Distributed (No. of Students)**	**No. of Forms Returned**
A	34	34
B	38	21
C	35	22
D	37	35
E	34	25
F	35	21
G	39	34
H	30	28
I	40	33
TOTAL	322	253

[a]Class sections were changed to protect the privacy of the respondents.

Table 2. Five-Point Scale/Weight, Mean Range, and Verbal Interpretation

Five-Point Scale/Weight	Verbal Interpretation
5	Always
4	Often
3	Sometimes
2	Seldom
1	Never

Table 3. The Independent Variables

Independent Variables	No. of Questions	Cronbach alpha	Factor Loading
Knowledge error	6	0.799	0.772
Memory error	5	0.906	0.910
Thought error	6	0.851	0.474
Habit error	2	0.896	0.880
Sensorimotor error	2	0.913	0.457
Omission error	2	0.844	0.829
Judgment error	2	0.690	-

Table 4. Factor Analysis of Novice Java Programmers' Errors Committed

Types of error (Errors Committed)[b]	Eigenvalue	Factor Loading	Mean
Invalid symbols or keywords			
No period between class name and method name	5.209	0.754	1.94
Capitalized keywords		0.524	2.11
Replacing (and) with < and > or [and] in output stream		0.647	1.80
else without *if*		0.539	2.06
Overall mean			**1.98**
Mismatched symbols			
Unmatched curly braces	1.445	0.585	2.34
Incorrect *greater than* or *equal to* sign		0.592	2.30
Cannot find symbol because of mismatched between the declared and		0.631	2.48
Cannot find symbol because of undeclared variable		0.741	2.55
Overall mean			**2.42**
Missing symbols			
No semi-colon at the end of a statement	1.281	0.602	2.14
No close/open parenthesis on *if*-condition		0.739	1.91
No parentheses on *if*-condition		0.629	1.92
Unclosed literals		0.540	2.14
Overall mean			**2.03**
Inappropriate naming			
Inappropriate casing of method names	1.135	0.713	2.40
Inappropriate casing of class names		0.743	2.18
Splitting a class name by putting a space		0.732	2.09

Overall mean			**2.22**
Excessive symbols			
Excessive semi-colon	1.072	0.639	1.79
Putting a period between the keyword *import* and java packages		0.787	2.11
Putting a semi-colon after the *if*-condition		0.523	2.42
Overall mean			**2.11**

[b]cummulative % of variance = 56.346

Table 5. Sources of Errors Committed

Source	Mean	Rank
Thought error	5.18	1
Sensorimotor error	4.34	2
Omission Error	3.78	3
Memory error	3.72	4
Knowledge error	3.28	5
Habit error	3.23	6
Overall Mean	**3.92**	

Table 6. Predictors of Errors Committed by Novice Java Programmers

Types of Error	Predictor(s)	Beta	Sig.	Adj. R^2	F-value	Sig.
Invalid symbol	Knowledge error	0.395	0.000	0.153	45.931	0.000
Mismatched symbol	Knowledge error	0.317	0.000	0.127	19.17	0.000
	Sensorimotor error	0.128	0.035			
Missing symbol	Knowledge error	0.134	0.043	0.058	8.58	0.000
	Habit error	0.176	0.008			
Inappropriate naming	Knowledge error	0.332	0.000	0.106	30.408	0.000
Excessive symbol	Knowledge error	0.183	0.004	0.029	8.423	0.004

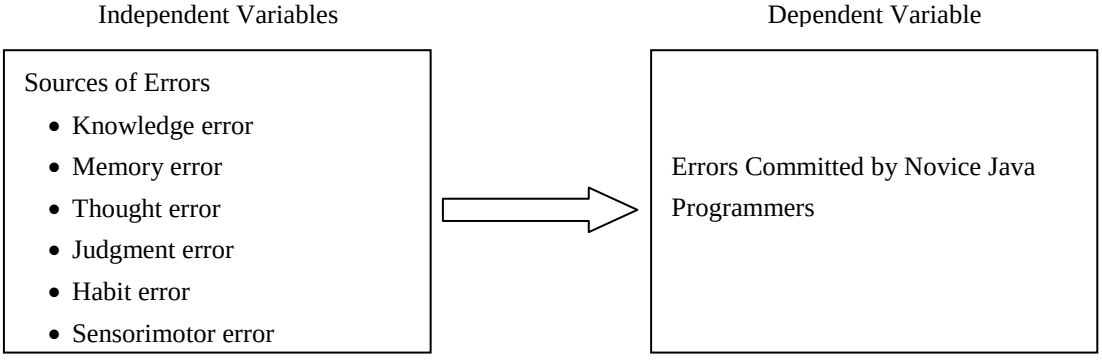

Figure 1. The Research Framework Showing the Sources of Errors That Might Predict Errors Committed by Novice Java Programmers