# An Analysis of the Errors Made by Novice Programmers in a First Course in Procedural Programming in Java

Nelishia Pillay
School of Computer Science
University of KwaZulu-Natal
Pietermaritzburg Campus
Pietermaritzburg, KwaZulu-Natal
South Africa
E-mail: pillayn32@ukzn.ac.za

Vikash R. Jugoo
Department of Computer Science
Mangosuthu Technikon
Umlazi, KwaZulu-Natal
South Africa

E-mail: Vikash@julian.mantec.ac.za

## Abstract

*Novice programmers usually experience a number of learning difficulties. The main aim of the study reported in this paper, is to identify these problem areas for first time procedural programmers. The paper describes the overall methodology employed to determine these learning difficulties. The study revealed that the most frequently occurring problems are poor planning and problem solving ability, a lack of knowledge of the programming language, a lack of understanding of the application domain, and a lack of conceptualization of the execution of the program. In addition to this the study has also identified the incorrect transfer of knowledge, a lack of understanding of control structures, incorrect identification of control structures needed, and inefficient problem solving approaches as the causes of errors in student programs. Furthermore, programming topics which students experienced the most difficulty with were modularization and iteration. Future work will involve developing instructional strategies to assist novices in overcoming these difficulties.*

*Keywords: novice programmers, learning difficulties, procedural programming*

## 1. Introduction

According to Jenkins (2002), "few students find learning to program easy". Hence, there is a need to identify the problems experienced by learners and develop instructional strategies to help reduce the learning curve. The study presented in this paper investigates the causes of errors made by novice procedural programmers in a first course in Java programming. The paper firstly examines the results of previous studies conducted to identify problems experienced by first time programmers. The paper then reports on the study conducted by the authors to identify causes of learning difficulties experienced by Computer Science students in a first year Java programming course at two South African tertiary institutions, namely, University of KwaZulu-Natal and Mangosuthu Technikon . A comparison of these results with those obtained in the previous studies reported is provided. The main contributions of this paper are the detection and categorization of errors made by South African novice procedural programmers and the identification of those areas which learners experience most difficulties with.

The next section provides a brief overview of previous work conducted in this domain. **Section 3** describes the methodology employed to identify the causes of errors made by novice programmers and a discussion of these causes is presented in **Section 4**. **Section 5** summarizes the problems experienced by first time programmers and future extensions of this study are presented.

## 2. Previous Work

The main causes of learning difficulties reported by previous studies include poor problem solving ability, insufficient understanding of programming constructs, an inability to determine the effect of a program during execution of the program and egocentrism.

Research into the difficulties experienced by novice programmers was initiated in the 1980's. Studies conducted in this era include that of Perkins et al. (1986) to determine the problems experienced by high school pupils enrolled for a first course in BASIC programming. The study revealed that student difficulties could be attributed to the students' lack of problem solving skills and student's possessing "fragile" programming knowledge. The authors describe fragile knowledge in terms of missing knowledge, inert knowledge, misplaced knowledge and conglomerated knowledge. Missing knowledge refers to the students' lack of knowledge, for example, of the programming language. Inert knowledge is knowledge that the student does have, but does not realize that it is applicable to a particular context. Misplaced knowledge results in the learner using programming constructs in places in the code where they are not needed, while conglomerated knowledge refers to those cases where the student code contains syntactical and semantic errors.

Investigations were carried out by Spohrer et al. (1986) to identify errors made by novice Pascal programmers at Yale University. Spohrer et al. found that errors made by novices can be attributed to the lack student's of understanding of the semantics of the different programming concepts (the data-type inconsistency, natural language, human interpreter and coincidental ordering problems); misinterpretation of the application domain (related knowledge interference, duplicate tail-digit, boundary and expectation and interpretation problems) and an inability to visualize the effect of code during execution (plan-dependency problem).

Hristova et al. (2003) categorize errors made by first time Java programs as syntax, semantic and logic errors. Syntax errors essentially refer to typographical errors in programs including the incorrect ordering of keywords. Semantic errors arise due to the learners' incorrect conceptualization of how certain programming constructs are interpreted while logic errors occur due to incorrect knowledge or understanding of concepts, e.g. including a return statement in a void method. The same classification of errors is reported by Amadzadeh et al. (2005). A majority of the errors made by students in the latter study were semantic errors. The topic that students appeared to make the most errors with was methods.

According to Fergusson et al. (2004) and Lahtinen et al. (2005) students are unable to break a problem down into sub-problems and identify the correct programming structures that should be used to implement solution algorithms to these sub-problems. This is attributed to the students' poor problem solving ability and an insufficient understanding of the functioning of the different programming control structures, e.g. a loop.

Mostrom et al. (1998) state that students usually experience difficulties in forming the correct mental models of the functioning of these constructs. The importance of formulating the correct mental models of programming concepts is further emphasized by Lui et al. (2004) and Ramalingam et al. (2004). In addition to this students attempt to code a solution before planning how the program can be represented using the different programming constructs.

Novice programmers are unable to determine what their program does, i.e. the effect of each programming instruction during execution of the program and hence experience difficulties in debugging their programs (Amadzadeh et al. 2005; Bailey et al. 2005; Chmiel et al. 2004; Edwards 2004; Lahtinen et al. 2005). This leads to students making random changes to their programs in an attempt to debug the program. This in turn results in more errors being introduced into the program.

Fergusson et al. (2004) report that first time programmers often have the expectation that the computer or compiler knows what to do, e.g. automatically initialize variables, expect code further down in the program to be executed. This is referred to this as egocentrism (Fergusson et al. 2004).

Studies conducted by Ginat (2003) and Haberman et al. (2005) revealed that students also experience difficulties in developing efficient computer programs in terms of the running times of the program. Ginat (2003) attributes this to the "design-by-keyword" syndrome which results in novices attempting to design solutions directly from the keywords in the problem rather than a deeper analysis or understanding of these terms.

These studies have also revealed that the programming concepts that novices usually experience difficulties understanding are variables, the *for* and *while* loops and functions/methods.

## 3. Methodology

This section provides a brief overview of the methodology employed to identify problems experienced by students taking a first course in procedural programming. The study was carried out at two tertiary education institutions, namely, the University of KwaZulu-Natal and Mangosuthu Technikon. At both institutions the programming course used to carry out this investigation was a first course in Java programming which covered the procedural aspects of programming. Both these courses focused on the following programming concepts: variables, data types, simple input and output, selection, repetition and nested repetition. Approximately 61 first year Computer Science students were enrolled for the University course and 30 students for the Technikon course. Instruction was in the form of lectures and practical sessions. During practical sessions students were required to implement solution algorithms, in Java, to the assigned programming tasks. Based on the methodologies reported in the literature surveyed, the following sources of data were analyzed in order to identify learning difficulties experienced by novice programmers:

- Observations during practical sessions – Errors made by students and learning difficulties encountered during practical sessions were documented by lecturers and tutors.
- Practical tests and examinations – The programming errors made by students in tests and examinations were also documented.
- Mental models – According to George (2000), Lui et al. (2004) and Ramalingam et al. (2004), mental models of programming constructs represent the programmers understanding of the functioning of the construct. Incorrect mental models of programming control structures, has been reported as one of the causes of the problems experienced by novice programmers. The mental models of students struggling to program were studied for the different programming control structures in an attempt to identify student misconceptions. Traces were used to represent mental models. An example of a trace is illustrated in **Figure 2**. This is a trace of the Java code fragment in **Figure 1**.

```
int sum=0;
for(int count=2;count<=6;count=count+2)
{
   System.out.println(count);
   sum+=count;
}
System.out.println("Sum is "+sum);
```

*Figure 1: Java Code Fragment*

| SCREEN | MEMORY |
|--------|--------|
| 2<br>4<br>6<br>Sum is 12 | sum = 0<br>count =2  sum = 2<br>count =4  sum = 6<br>count =6  sum = 12<br>count = 8 |

*Figure 2: Trace of the Java code fragment in Figure 1*

According to Lahtinen et al. (2005) the use of student interviews and student questionnaires did not prove to be an effective means of collecting data as students were not able to clearly identify their difficulties and often provided subjective opinions. Thus, these instruments were not used in this study.

## 4. Learning Difficulties

The collected data was analyzed as follows:

- A general list of errors made in practicals was created.
- A list of misconceptions in students' mental models for the different programming concepts was constructed.
- For each test and examination, a list of errors was maintained for each student.

These errors were then placed into categories as defined in the literature survey. It was decided that the classification systems used by Hristova et al. (2003) and Amadzadeh et al. (2005) for classifying errors in terms of syntax, semantic and logic was too general and the more specific categories were used. In some cases a new category had to be created. The following 8 categories were established:

- **D1: Incorrect Transfer of Knowledge** - In this case students incorrectly apply knowledge that they have acquired from previous programming examples to a programming problem, e.g. assigning initial values that were used in previous examples to accumulators and counter variables but are incorrect in the current context; the use of variables names from previous examples that are not relevant to the problem at hand; prompt the user to enter values when this is not required; based on previous examples assume that the result of each iteration of a loop is automatically output.
- **D2: Lack of Understanding of the Application Domain** - Errors arise as a result of students either not possessing knowledge of the processes they have to write an algorithm for, or misinterpreting the description of the process provided in the question. For example, not knowing what the factorial of a number is; when calculating the variance each $x_i$ is interpreted to be in the range 1 to the number of numbers instead of the actual number; values illustrated in examples are hard-coded in programs. Similar errors were detected by Spohrer et al. (1986).
- **D3: Lack of Understanding of Control Structures** - Errors occur due to the student incorrectly conceptualizing the functioning of a control structure, e.g. updating the counter variable in a *for* loop after the condition is checked, testing the condition of a *dowhile* loop at the beginning of the loop; including the calculation of the average of numbers in the loop for. This lack of understanding of control structures was also detected in the previous studies discussed in **Section 2**.
- **D4: Lack of Conceptualization of the Execution of the Program** - The student is unable to conceptualize the execution of the program, i.e. the effect of the implementation of each statement.

As a result of this the student makes errors such as overriding stored values; accessing variables not within scope of the current method; execution of statements in the incorrect order; including commands in a loop which should be executed after the loop such as outputting or initializing an accumulator; displaying introductory headings after values have been read in; not outputting the result of the program. This lack of conceptualizing what the program is actually doing has also been identified as a learning difficulty in other such studies (Amadzadeh et al. 2005; Bailey et al. 2005; Chmiel et al. 2004; Edwards 2004; Lahtinen et al. 2005).

- **D5: Problem Solving Ability** - The student develops a partially correct algorithm or is unable to derive an algorithm to solve the problem, e.g. swapping of values; calculating the variance of a list of numbers. Perkins et al. (1986) , Fergusson et al. (2004) and Lahtinen et al. (2005) have also identified the novice's lack of problem solving ability as one of the main causes of errors in student programs. Beaubouef et al. (2001) state that is not that students cannot solve computing problems specifically, but they cannot solve problems in general.

- **D6: Lack of Knowledge or Understanding of the Programming Language** – This category is similar to the logic category described by Hristova et al. (2003) and Amadzadeh et al. (2005). The student does not have the necessary declarative knowledge, e.g. the syntax of the programming language; the functioning of an operator such as the *ceil* or *floor* operator; how to perform a particular function such as the logical *or* operation.

- **D7: Incorrect Identification of the Control Structure Needed** - The incorrect control structure is used to perform a particular function, e. g. a conditional loop instead of an if-statement is used to implement an error check; a *while* or *dowhile* is used where a *for* loop should be used; the use of if-statements instead of if-else statements; the need for an iterative control structure is not identified.

- **D8: Inefficient Problem Solving Approach** - In this case the student is able to derive a solution, however the solution is inefficient in that the execution time should be less; less memory should be used or the length of the program (i.e. number of lines of code) could be shorter. For example, performing unnecessary conditional checks; declaration of unnecessary variables; using separate loops when loop limits can be one of two values.

**Appendix A** lists some of the errors commonly made for each of the programming concepts taught. The pie chart in **Figure 3** illustrates the frequency of each type of difficulty encountered by students in the final practical examination for the University course. The examination consisted of two tasks for which the students had to write Java programs, and one requiring them to correct the semantics of a program. A poor problem solving ability and insufficient knowledge and understanding of the programming language appear to be the most common problems experienced by students. Other frequently occurring problems include the lack of understanding of the application domain and lack of conceptualization of the execution of the program.

**Figure 4** depicts the distribution of errors made by students in the final practical examination over the different programming concepts, namely, variables and typing (E1), arithmetic operations (E2), logical operations (E3), input and output (E4), conditional control structures (E5), iterative control structures (E6), and modularization (E7). Errors for each of these topics include both syntax and semantic errors. It is evident from pie chart in **Figure 4** that students experienced the most difficulty with modularization , i.e. concepts related to including methods in the program, followed by iteration.
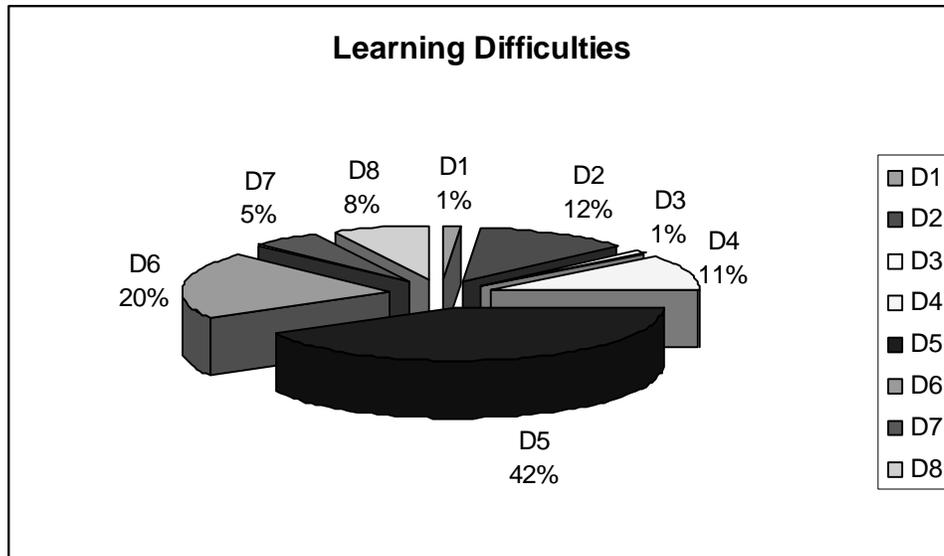
**Figure 3:** *Learning Difficulties*
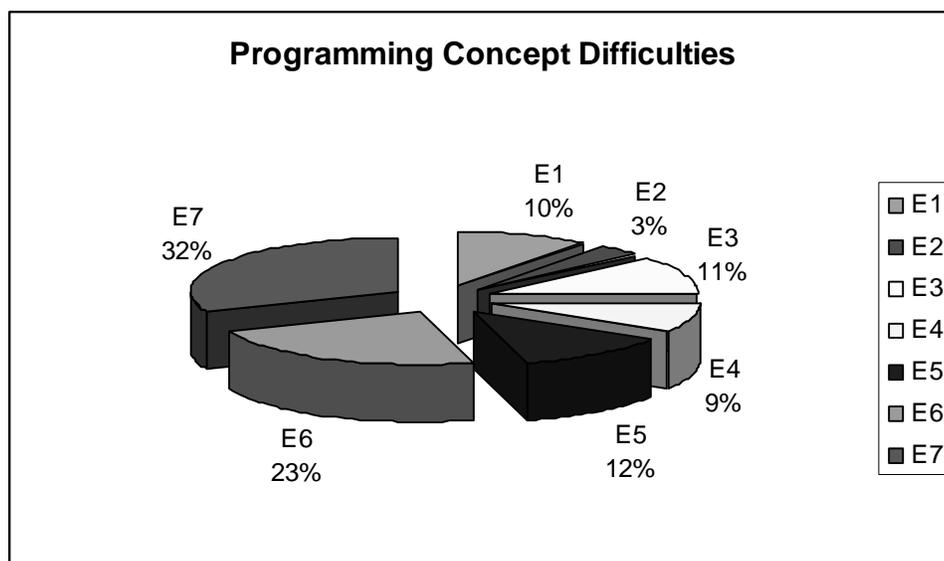


**Figure 4:** *Distribution of Errors for the Different Programming Concepts*

### 5. Conclusion and Future Work

This paper reports on a study conducted at two tertiary institutions, namely, University of KwaZulu-Natal and Mangosuthu Technikon, to determine the difficulties experienced by novice programmers. Consistent with the literature reviewed, this study has also revealed poor problem solving ability, lack of conceptualization of the execution of the program, lack of knowledge and understanding of the programming language, lack of understanding of control structures and an inefficient problem solving approach as difficulties experienced by first time programmers . In addition to these other problems identified by the study include the incorrect transfer of knowledge, the lack of understanding of the application domain, and incorrect identification of the control structure needed.

A detailed study of the errors made by students was conducted for the final practical examination for the University course. The main causes of errors made by students in the examination were a poor problem solving ability and the lack of knowledge and understanding of the programming language. Furthermore, students experienced the most difficulty with modularization and iteration. This is also consistent with the results reported in previous such studies.

Future extensions of this project will focus on developing instructional strategies to assist novice programmers in these 8 problem areas. For example, students with a poor problem solving ability will be given additional tutorials to help improve their problem solving skills by assigning them tasks like those described in (Levitin 2005). Methods for teaching each of the programming concepts will also be revised so as to take into consideration the errors identified by the study for each of the topics.

## 6. References

1.  Ahmadzadeh M., Elliman D., & Higgins C. (2005). An Analysis of Patterns of Debugging Among Novice Computer Science Students. In *SIGCSE Bulletin inroads, ITiCSE 2005 Proceedings*, 37, 3, pp. 84 – 88.
2.  Bailey M. W. (2005). IronCode: Think-Twice, Code-Once Programming. In *SIGCSE Bulletin-inroads, Proceedings of the SIGCSE' 05*, 37, 1, pp. 181-185.
3.  Beaubouef T., Lucas R., Howatt J. (2001). The UNLOCK System: Enhancing Problem Solving Skills in CS-1 Students. In *SIGCSE Bulletin – inroads*, 33, 2, pp. 43 – 46.
4.  Chmiel R., & Loui M. C. (2004). Debugging: From Novice to Expert. In *SIGCSE Bulletin inroads, SIGCSE '04 Proceedings*, 36, 1, pp. 17 – 21.
5.  Edwards S. H. (2004). Using Software Testing to Move Students from Trial-and-Error to Reflection-in-Action. In *SIGCSE Bulletin inroads, SIGCSE '04 Proceedings*, 36, 1, pp. 26 – 30.
6.  Fergusson, K., Darllenfyd (2004). Novice Programming Archive, http://ccse.monash.edu.au/~kef/research/readings/archives/cat_novice_programming.html
7.  George, C. E. (2000). Evaluating a Pedagogic Innovation: Execution Models and Program Construction Ability, http://www.ics.1tsn.ac.uk/pub/conf2000/Papers/george.htm.
8.  Ginat D. (2003). The Novice Programmers' Syndrome of Design-by-Keyword. In *SIGCSE Bulletin inroads, ITiCSE 2003 Proceedings*, 35, 3, 154 - 157.
9.  Haberman B., Averbuch H., & Ginat D. (2005). Is it Really and Algorithm – The Need for Explicit Discourse. In *SIGCSE Bulletin inroads, ITiCSE 2005 Proceedings*, 37, 3, pp.74 – 78.
10. Hristova M., Misra A., Rutter M., & Mercuri R. (2003). Identifying and Correcting Java Programming Errors for Introductory Computer Science Students. In *SIGCSE Bulletin inroads, SIGCSE '03 Proceedings*, 35, 1, pp. 153-156.
11. Jenkins, T. (2002). On the Difficulty of Programming, http://www.psy.gla.ac.uk/~steve/localed/jenkins.html.
12. Lahtinen E., Ala-Mutka K., & Jarvinen H. (2005). A Study of the Difficulties of Novice Programmers. In *SIGCSE Bulletin inroads, ITiCSE 2005 Proceedings*, 37, 3, pp.14 – 18.
13. Levitin, A. (2005). Analyze That: Puzzles and Analysis of Algorithms. In *SIGCSE Bulletin inroads*, 37, 1, pp. 171 – 175.
14. Lui A. K., Kwan R., Poon M., & Cheung Y. H. Y. (2004). Saving Weak Programming Students: Applying Constructivism in a First Programming Course. In *SIGCSE Bulletin inroads*, 36, 2, pp. 72 – 76.
15. Mostrom, J.E., Carr, D. A. (1998). Programming Paradigms and Program Comprehension by Novices, http://www.ida.liu.se/~davca/postscript/parvsserial.pdf.
16. Perkins D. N., & Martin F. (1986). Fragile Knowledge and Neglected Strategies in Novice Programmers. In E. Soloway, & Iyengar S., (Eds.), *Empirical Studies of Programmers*. (pp. 213 – 229). Ablex Publishing Corporation.
17. Ramalingam V., LaBelle D., & Wiedenbeck S. (2004). Self-Efficacy and Mental Models in Learning to Program. In *SIGCSE Bulletin inroads, ITiCSE 2004 Proceedings*. 36, 3, pp. 171 – 175.
18. Spohrer J.G., Soloway E. (1986). Analyzing the High Frequency Bugs in Novice Programs, In Soloway E., Iyengar S.(Eds.), *Empirical Studies of Programmers*. (pp. 230 – 251). Ablex Publishing Corporation.

# 7. Appendix A: Summary of Errors Made for Each of the Programming Concepts

## 7.1 Variables, Typing and Constants

- Calculations performed as part of variable declarations.
- Assigning more than one calculation to a variable.
- Variables declared to be of the incorrect type.
- Accumulators not initialized.
- Incorrect initial value assigned to an accumulator.
- Declaration of unnecessary variables.
- Unnecessary initialization of variables.
- Undeclared variables.
- Reference to variables out of scope.
- Type of variable not specified in declaration.
- Use of variables that have not been assigned values only declared.
- Placement of declaration of variables and constants, e.g. before the main method.

## 7.2 Arithmetic Operations
- Calculations placed in inverted commas.
- Incorrect syntax of equations.
- Incomplete equations.
- The output format function used instead of the round function.
- Incorrect use of a function, e.g., the *ceil* function.
- Difference between / and % not understood.
- Integer division performed when not required.
- Operation of the ++ operator not understood.

## 7.3 Logical Operations
- Incorrect syntax of Boolean equations
- = used instead of ==.
- & used instead of &&.
- !== used instead of !=.
- Equality of more than one value tested with == instead of a combination of == and ANDs.
- ANDs used where ORs should be used and vice versa.
- Separate if-statements used instead of using the OR operator.

## 7.4 Input and Output Operation
- Result of program not output.
- Each possible value of input read in and stored instead of using one variable and testing the value read in.
- Assumption that spaces in program will appear on the screen.
- Output not formatted or incorrectly formatted.
- Output statements extending over more than one line.
- Missing inverted commas in string output.
- Syntax errors in the combination of variable and string output.

## 7.5 Conditional Control Structures
- Syntax errors such as missing semicolons.
- Block of statements to be executed not contained in { }, thus entire block not executed if condition is met.
- If-else used instead of an else for the last option.

- Condition not specified.
- A number of else's without if's.
- "if else" used instead of "else if".
- Single if-statements used instead of if-else statements and vice versa.
- If-statement used instead of a conditional loop and vice versa.

## 7.6 Iterative Control Structures
- Syntax errors, e.g. missing semicolons; incorrect *for* loop declaration.
- Incorrect type of loop used, e.g. *while* instead of *dowhile*.
- Conditional loop used instead of if-statement and vice versa.
- No content in loop.
- Infinite loops.
- Counter variable changed in loop.
- Incorrect conditions for conditional loops.
- Counter variable used as accumulator.
- Accumulator initialized in the loop.
- Calculations that should be performed after the loop are performed in the loop.
- Loop update not performed for conditional loops.
- Incorrect update performed in loop.
- Functioning of the different loops not understood.
- Scope of variables used in conditional loops not understood.
- Scope of the counter variable used in the *for* loop not understood.

## 7.7 Modularization
- Syntax errors e.g. return statement specified in (), semicolon after method declaration.
- Method returning a value declared as void.
- Value returned by method not stored or used.
- Variable to store method output passed to method.
- Name of method returned.
- Method defined after class definition.
- Method not used although question requires it.
- Method included in main method.
- Typed method does not return a value, outputs or stores the value.
- Values not passed to method.
- Return statement used to call up function in main method.
- Method performs the incorrect function.
- Method name does not reflect the function being performed.
- Method stores values passed to method in new variables instead of using dummy variables (refer to the parameters specified as part of the method declaration).
- Value of the incorrect type returned.
- Dummy variables declared in method and method declaration.
- Scope of variables not understood, e.g., dummy variables (not declared in main method) used in function call to method; variables used in method declared in main method (dummy variables not used).
- Attempts to return two values from a method.
- Values passed to method overridden.