The Influence of Class Structure on Program Comprehension

Ahmed Alardawi	Babak Khazaei	Jawed Siddiqi
Sheffield Hallam University	Sheffield Hallam University	Sheffield Hallam University
Aalardaw@my.shu.ac.uk	B.Khazaei@shu.ac.uk	J.I.Siddiqi@shu.ac.uk

Abstract

We report on a research that aims to investigate the effect of class structure on program comprehension. The subject groups are novices and the treatments are simple programs without class structure versus the equivalent programs with classes present; they are termed respectively as: *Non-Class* based programs and as *Class* based programs. Data was collected from three different sets of studies comprising of a total of 211 undergraduate first year computer science students from different institutions.

Some findings of these three sets of studies are put together and reported, in particular the overall results indicate that *Class* based programs were more understandable, readable, and accessible than the corresponding *Non-Class* based programs. Our findings align with and support those works that claim the cognitive benefits of the OO paradigm. Limitations and directions for future research are highlighted.

Keywords: POP-II A. novices, B. program comprehension, POP-IV A. object-oriented design, Pop-V.B. Questionnaire

1. Introduction

Pfleeger (2006) defines OO paradigm as “an approach to software development that organise both problem and its solution as a collection of discrete objects; both data structure and behaviour are included in the representation”. She also identifies the OO representation by seven characteristics: identity, abstraction, classification, encapsulation, inheritance, polymorphism, and persistence (Pfleeger 2006). These characteristics have changed the nature of software development; however, they have set a considerable debate about their appropriateness from both human factors and software engineering perspective. Good understanding of OO characteristics will positively affect the programmers’ skills. We are especially interested in the acquisition of programming skills by novice programmers. Our particular emphasis is on program comprehension since it forms the underpinnings for many programming activities.

Class structure represents one of the essential concepts of Object-Oriented paradigm and therefore, a good understanding of this concept will positively affect the effectiveness of novice programmers. Comprehension underpins many programming activities such as program design and program implementation. In this context, the comprehension represents a mental model approach that involves interesting theoretical frameworks of program comprehension. Our starting point is Burkhardt, et.al (2006) cognitive model for OO program comprehension that considers two distinct but interacting models: program and situation. Our focus does not rely primarily in distinguishing between these models, but use both of them to assess the influence on novices of class structure on program comprehension.

There is a wealth of literature that put forward a wide range of theories and models devising an account of text and program comprehension (Brooks 1978; Kintsch et al 1978; Johnson-Laird 1986; Van Dijk and Kintsch 1983; Pennington 1987a, b, Burkhardt, et.al 2006). Program comprehension represents in this context a mental model approach that involves interesting theoretical frameworks of program comprehension.

Section 2 provides a background to the psychology of programming studies of program comprehension. Section 3 is a brief report of the 3 sets of experiments which were replication of the same study materials at different institutions. Some conclusions and observations for future work are given in section 4.

2- Background to the Research

Empirical study of object-oriented programming style began to appear as early as 1995. The mental model approach has been used to explain the comprehension of procedural programs (Pennington 1987a, b) and more recently OO programs (Burkhardt, et.al 2006).The studies applied these cognitive models primarily in areas of comparing comprehension of programs written in different programming paradigms, and investigating the influence of certain tasks on program comprehension.

Wiedenbeck, Ramalingam, Sarasamma, and Corritore (Corritore and Wiedenbeck, 1999; Wiedenbeck and Ramalingam, 1999; Wiedenbeck et al, 1999) have completed a series of studies similar to the earlier Pennington's study. These studies attempted to compare mental representations constructed by OO programmers with those for procedural programmers. The overall results of these studies point out that novice comprehending programs written in OO style form stronger situation model than program models.

Khazaei and Jackson (2002) have also conducted an empirical study to investigate the program comprehension differences between event-driven and object-oriented styles for novice programmers. Interestingly, results show that information related to situation model was more accessible than information related to program model in both styles. Although they agreed that Pennington model had provided a good framework to investigate comprehension differences between programming styles, it was limited in the case of these two styles as it did not cover graphical representation and/or advanced OO concepts.

It is difficult to claim that the OO paradigm is not a "natural" way of conceptualising and modelling real world situation when the exercise being used is a simple OO program. If the measures used for verifying the mental representation are not accurately reflecting the object-oriented mental representation then these types of claims are difficult to support too (Sajaniemi and Kuittinen 2007; Alardawi, et.al 2010).

Burkhardt, et.al (2006) study of OO program comprehension aimed to examine how the mental model can be affected by the programmer expertise, programming task, and the development of comprehension over time, Burkhardt, et.al (2006) claimed that Pennington model has several limitations with relation to OO paradigm. Firstly, Pennington model does not examine representations about problem classes and objects or even data structures. Since objects are central entities in OO programs, the construction of the representation of objects should be taken into account in a model of OO program understanding, Burkhardt, et.al (2006) assume that the representation of objects is part of the situation model in as much as it reflects the objects of the problem situation. Secondly, Pennington model

accounts for understanding of short programs but does not scale up easily to larger programs. Two important OO aspects are not accounted for: the representation of delocalised plans and the representation of text macrostructure. Pennington assumes that plan representations of a program are primarily based on data flow. However, in the case of long programs, particularly in OO programs, it happens that many plans are delocalised. According to Rist (Rist 1996), plans and objects are orthogonal, a plan is a set of actions that, when placed in a correct order, achieves some desired goal. The actions in a plan are encapsulated in a set of routines, and the routines are divided among a set of classes and connected by control flow. Détienne (2006) claims that this can reflect the real world, where a plan can use many objects and an object can be used in many plans. In the Burkhardt, et.al (2006) OO model of program comprehension, they take the view that the construction of these complex delocalised plan representations is primarily based on client-server relationships, in which one object processes and supplies data needed by another object. Pennington also account for the representation of elementary operations as part of the text microstructure. However, the macrostructure of long programs which consisting of the representation of larger text units such as routines is not accounted for in her model. Burkhardt, et.al (2006) OO model considers that the representation of the macrostructure is based on the elementary functions of the program model. In summary, Burkhardt, et.al mental model takes into account the nature of OO programs such as classes and objects, message passing, and the size of OO programs (For more information about these categories see Burkhardt et al 2006).

Since we are assessing the effect of the class structure on OO program comprehension, the investigation framework should take into account the problem class category as well as the other 5 categories used by Pennington. Table 1 gives a summary of these six category knowledge. We have used all these categories to formulate our comprehension questions (see appendix A for a sample of 6 out of 19 questions representing the categorise).

Category Knowledge	Knowledge structures	Mental representation	Model
Elementary operations	Text structure knowledge	Dynamic and functional views	Program model
Control flow	Text structure knowledge	Dynamic view	Program model
Data flow	Plan knowledge	Dynamic and functional views	Situation model
Function	Plan knowledge	Functional view	Situation model
Problem Classes	Problem and Plan knowledge	Object view	Situation model
State	Plan Knowledge	Dynamic and functional view	Prog/Situ model

Table 1: the 6 categories of Knowledge for Comprehension Questions

The following is an explanation of each category listed in table 1 above:

Elementary operations knowledge: it forms part of the text microstructure, constitute basic text units usually consisting of one or few lines of code. The feature of this category is that it is directly available in the source code.

Control flow knowledge: it forms part of the text microstructure, constitutes the links between text units, which is in the simplest case sequential or in complex situations involves looping or calls to subprograms, thus this category is procedural in nature.

Data flow knowledge: it relates to Communication between variables, corresponds to data flow relationships connecting units of local plans within a routine and also changes that occurs to data variables while they pass through the program. The transformations of the data are, thus, at the heart of whatever useful action a program achieves. For this reason, data flow information is considered to be very closely related to a program's functions and goals and to form a part of the situation model.

Functions knowledge: explains the goal of the whole program, what the program accomplishes in terms of the problem situation it addresses. Function information expresses what the program does in terms of entities, relationships, and actions in the world, this information is usually not directly available in a program text, but must be inferred from the program text in combination with knowledge of the real world problem domain of the program.

Problem Classes Knowledge: it forms a part of problem and plan knowledge, these classes directly model classes of the problem domain. This information directly reflects the understanding of class structure in the program.

State knowledge: comprises the state of all aspects of the program at the time a given action occurs in a program.

Burkhardt, et.al (2006) reported that OO paradigm facilitates the construction of the situation model most strongly. Although the model could support the claim about the naturalness of OO paradigm, the generality of the proposed comprehension model is to be questioned. The situation model is more likely notation-independent whilst the program model is mostly depends on the notation. Therefore, replicating Burkhardt, et.al experiment across different OO languages and different problem domains will most likely help in investigating further for this claim. The work reported in this paper will build on Burkhardt, et.al (2006) experiment with focusing specifically on the class structure.

Although there are reasonable numbers of studies assessing the novices' program comprehension of OO concepts, the focus on specific concept of OO is very limited. It is very difficult to include all the OO features and concepts in a single study. Our earlier work (Alardawi, et. al 2010) identified encapsulation in the form of class structure and the hierarchy in the form of inheritance as possible areas for conducting further empirical study. This study is specifically focused on the influence of class structure on novices' program comprehension. Section 3 will focus on reporting the conduct of our 3 sets of experiment based on the same material.

3- Report of the 3 Sets of Experiments

For the purpose of this paper we have combined our data of the 3 sets of the experiments conducted at different institutions based on the same material.

3.1 Aim

The overall aim of this research is to investigate the influence of class structure on program comprehension. The research investigates the mental representations constructed by novices during comprehension of a *Class* based OO program in contrast to a *Non-Class* based OO program. We are focusing on the relevant category knowledge of the previous study as represented in table 1 of section 2.

3.2 Subjects

211 undergraduate first year computer science students from three institutions participated in the study. Demographic data were collected via a background questionnaire to highlight any significant differences among subjects. This revealed that the participants' gender ratio was 42% males and 58% females and their average age was about 20.5 years. The majority of the participants had no previous experience in object-oriented programming and the only significant current experienced programming languages encountered were Java and/or Visual Basic. All the 3 sets of subjects were however studying programming when the experiment took place.

3.3 Materials

Following on from the previous related empirical work, we modified and used the car problem scenario, which was used by (Ramalingam and Wiedenbeck 1997; Wiedenbeck and Ramalingam 1999; Wiedenbeck, et.al 1999; Khazaei and Jackson 2002). We modified the car program to emphasis or de-emphasis class structures for our experimental materials. Appendix A presents the two Visual Basic (VB) versions used in our studies. The program first allows the user to create a new car with specific engine and type. The program includes two variables referring to the speed and the number of passengers. The program then outputs different messages according to the speed of the car. This exercise is well known as beginners' pedagogy example and the problem knowledge used is considered as familiar to participants at this level.

The *Class* based versions (there were two, one in VB and another in Java) contained three classes, each class consisting of private data member(s) and public interface containing declarations of member functions. The execution starts in the main function, which begin by declaring objects of the classes engine and body. The engine and body are composed in Car class. The main function calls the 3 classes' functions in which the principal computations were carried out. The *Class* based versions is using OO features of classes, objects, encapsulation, and composition. The modification to car program of the previous studies is that we have introduced engine and body as distinct classes. As a result, the program size has increased from 28 lines of code into 65 lines of code in VB and 66 lines of code in java.

The *Non-Class* versions of the car programs did not use objects, classes or message passing. The VB program version consists of a graphical user interface and it uses variables

engine and type. Both VB and java versions initialise variables, and then they carry out the principal computations of the programs. All the other aspects of the two treatments were made as similar as possible. We have used similar names for the functions and variables of the two treatments. The *Non-Class* versions were slightly shorter than the *Class* based versions since they did not contain the overhead of classes' definitions, the numbers of lines of code in the VB version was 29, and the numbers of lines of code in the Java was 48. There were a set of 19 comprehension questions based on the category of knowledge of table 1 section 2 for each of the programs.

3.4 Procedure

We used first year undergraduate students who were learning VB at Sheffield Hallam University in UK as our first set here by labelled as 'set1'. For 'set2' we used first year undergraduate students who were also learning VB at the Faculty of Electronics in Libya. For 'set3' the subjects were first year undergraduate computer students who were learning java at the Faculty of Computer Technologies in Libya. The aim was to gather data from a large number of subjects.

The studies were carried out as paper-based exercises. At the beginning of each session, the subject sets were verbally informed about the procedure and explained that they were participating in an experiment. The subjects were assured that they were not being assessed.

There were two phases for each set of study. In the first phase, each participant was asked to fill out a background questionnaire; this phase was done to gather demographic data and to highlight any significant differences among subjects. In the second phase, the participants were divided into two matched groups. This division was based on the teachers' prior assessments marks of the subjects on the courses they were attending. Each participant was presented with a hard copy of either a *Non-Class* based program or a *Class* based program. Either a VB or a JAVA program was supplied depending on the course studied by the subjects. A set of corresponding comprehension questions with option of three kinds of responses for each question (YES, NO, DON'T KNOW) was also supplied. The start time at the beginning of the study and the end time for each participant were also recorded.

We refer to the means of the total correct responses to the 19 comprehension questions as a subject's total performance. We also refer to the means of the total correct responses of corresponding category knowledge questions as a subject's category performance.

The stated null hypotheses of the experiment are:

H01: There is no significant difference in terms of total performance in program comprehension between *Non-Class* based and *Class* based treatments.

H02: There is no significant difference in terms of category performance in program comprehension between *Non-Class* based and *Class* based treatments.

3.5 Results

For logistical reasons, we used different programming languages, as our subjects were covering different programming languages. In the first two sets of studies, VB was used to

represent the two treatments. In the last set of study, we used Java. The effect of the syntactical differences between the languages used on the level of comprehension questions were minimised as much as possible.

Preliminary analysis was done to determine whether there was a significant difference in total performance among the sets (set1, set2, and set2). A one-way ANOVA was run with “study set” as the independent variable and “subjects' total performance” as the dependent variable. The result was not significant. Therefore, the experiment 'set' was not included as a variable in further analysis. Another preliminary analysis which was also done to determine whether there was an effect from the programming languages. A one-way ANOVA was run with “programming language” as the independent variable and the “subjects’ total performance” as the dependent variable. The result was not significant, thus "programming language" was not included as a factor in the further analysis. After these analyses we felt justified that our further data analysis could combine the data.

Since we are not investigating the interaction between different independent factors, one-way ANOVA was considered as an appropriate statistical test to be used. We took appropriate advice from statisticians on this.

At the first level of analysis, the analysis was accounted for the effect of the six knowledge categories (table 1) on different sets of subjects. For all subjects, one-way ANOVA was used. The independent variable was the “knowledge category”. The dependent variable was the “category performance" of all the subjects. The ANOVA was significant ($F(5,1260) = 10.506$, $p<0.05$). Newman-Keul's test was run as a follow-up. It showed that there was significantly higher category performance on state knowledge than all other categories ($p<0.05$).

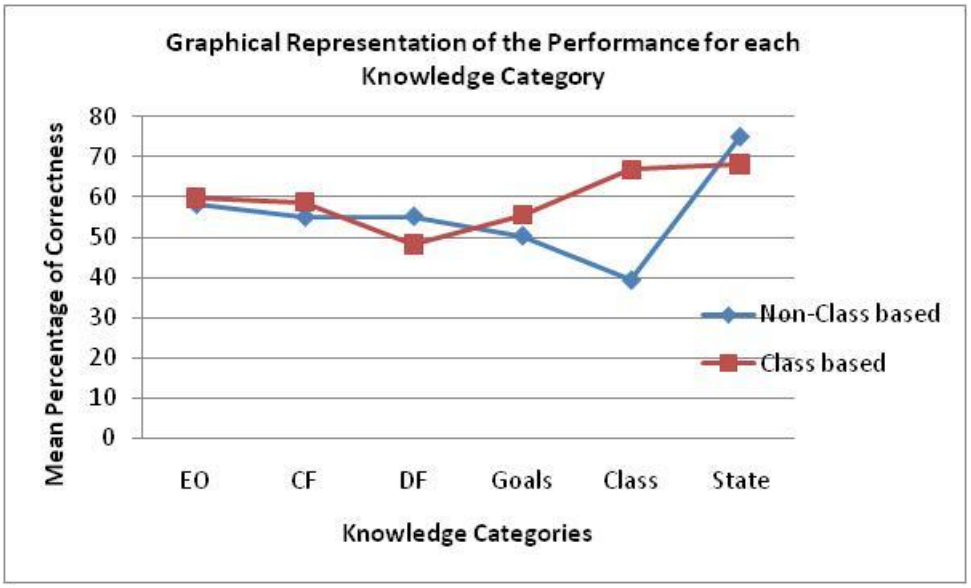
For the *Non-Class* based group, one-way ANOVA was used. The independent variable was the “knowledge category”. The dependent variable was the “category performance of the *Non-Class* based group”. The ANOVA was again significant ($F (5,635) =14.121$, $p<0.05$). Newman-Keul’s test was run as a follow-up. It showed that there was significantly higher category performance on state knowledge than on all other categories knowledge ($p<0.05$). Furthermore, the category performance of the class knowledge was significantly lower than on operations, control flow, and data flow knowledge ($p<0.05$).

For the *Class* based group, one-way ANOVA was also used. The independent variable was the “knowledge category”. The dependent variable was the “category performance of the *Class* based group”. The ANOVA was significant ($F (5,629) =5.075$, $p<0.05$). Newman-Keul’s test was also run as a follow-up. It showed that the category performance of the data flow knowledge was significantly lower than on the class and state knowledge ($p<0.05$).

In the second level of analysis, we focused on assessing the effect of the *Class/Non-Class* treatments on the subjects' total performance. One-way ANOVA was run. The independent variable was the "*Class/Non-Class*" treatments. The dependent variable was the “subjects' total performance”. The ANOVA result was significant ($F (1.201) =11.768$. $p<0.05$), it showed that there was a significant effect of the *Class/Non-Class* treatment on the subjects’ total performance. This could reject the first null hypothesis.

Considering the significant results of the effect of treatment on the subjects’ total performance between the subjects and the significant results of the effect of knowledge

categories on the subjects, further analysis was accounted for possible treatment effect between subjects' category performance. Figure 1 shows the category performance for *Non-Class* based and *Class* based subjects broken down by the knowledge categories.



Key to Mental Representation:
EO: Elementary Operations CF: Control Flow DF: Data Flow
GOALS: Program Goals CLASS: Problem Classes

Figure 2: Graphical Representation of the Performance for each knowledge Category

A one-way ANOVA was run. The independent variable was "treatment". The dependent variables were the "subjects' category performance" in each knowledge category. The test showed that the only significant effect of the treatment between subjects was on the class knowledge ($F(1,210) = 53.725$, $p<0.05$). However, test showed that there was no significant treatment effect between subjects in the other knowledge categories. Therefore, hypothesis H02 can be rejected only in the case of class knowledge category.

The third level of analysis was done to assess the effect of the programming languages on the *Class* based subjects' total performance. This was done to whether there is an effect of the way in which classes are represented in these languages. People might expect that representation of classes in java would have different results to representation of classes in VB. One-way ANOVA was run. The independent variable was "programming language": Java or VB. The dependent variable was the "*Class* based subjects' total performance". The test showed that no significant effect of the programming language on the *Class* based subjects' total performance.

3.6 Discussion

This investigation has focused on whether introducing class structure can influence the novices' comprehension of programs and the mental representations formed. In terms of number of classes used, and the program size, we have used larger programs compared to those used in prior related studies (Ramalingam and Wiedenbeck 1997; Wiedenbeck, et.al 1999; Wiedenbeck and Ramalingam 1999; Khazaei and Jackson 2002). However, the sizes we used are still small compared to most OO applications. In designing the treatments, the intention was made to minimise the effect of domain knowledge. Thus we feel no special

domain knowledge is needed for the car problem as we have kept the scenario as simple as possible, and the selected domain is well within the normal experience of our subjects. One potential problem for further investigation of OO program comprehension is the size of the OO programs especially for novice subjects and a design of a control experiment can become very difficult. Furthermore, for the car problem that we have used in our experiment, there is only one kind of semantic hierarchy in the problem in that a car consists of an engine and body. For many scenarios for OO systems, the hierarchy can be a lot more complicated and the OO structures could often suffer because there are competing natural hierarchies or other structures. For example, the parts hierarchy of a car would conflict with a different hierarchy say car types, seating, fuel system etc, where an OO hierarchy cannot usually represent both. When there is competition, there could be problems in program comprehension. Our results might have been different if there had been competition and therefore we need to be aware of these other factors.

Comparing the performance for *Non-Class* based and *Class* based showed interesting results. The performances of *Class* based subjects were generally better than the performance of the *Non-Class* based subjects. This may be explained as the *Class* based programs can make the program more readable and accessible than the *Non-Class* based programs. Détienne (2006) claims that the reverse mapping between the problem domain and the programming domain is more easy and straightforward in the OO paradigm than in the procedural paradigm. As program comprehension is based on the hypothesis of mapping from the program domain in to problem domain, our results could support the claims about the cognitive benefits of the OO paradigm. The mapping from program domain in to problem domain involves identifying the problem domain objects or entities and the associations between their structures and functionality. This activity is assumed to be driven much more by the programmer's knowledge about the real world structure than the knowledge about a particular software domains or programming knowledge.

Considering the performance in the different knowledge categories on the *Non-Class* based and *Class* based programs, we find clear evidence of differences in mental representation between the two. Results show that the patterns of response to comprehension questions on the two versions were very distinct. For the *Non-Class* based version, the state category was dominant in the mental representation. The high performance in this category can be accounted to high readability nature and clarity of the program structure that notation provides (Pennington 1987). For the *Class* based version, both class and state categories were dominant in the mental representation. The idea behind the class related knowledge was to reflect on subjects' ability in identifying problem entities. However, *Class* based subjects were introduced to class structure for the first time. Our results are similar to Burkhardt, et.al (2006), where novices scored highest in the class knowledge category among other knowledge categories. In the case of this study, the characteristics of the problem type used have facilitated highlighting the information related to this category. As problem type can be classified as non competing hierarchy, the classes in the *Class* based treatment are natural and already existing in the real world, car, body, and engine. This could facilitate the mapping from the program domain to the problem domain easier, thus identifying the problem classes used was easier. Moreover, the hierarchical solution structure in the *Class* based version (see appendix A) was also essential in the treatment and thus also played as a cue and helps highlighting the used classes. It would be fruitful to look at other types of problems for future experiments. In particular, currently we are looking

at Jackson's classification of problem types in order to choose the next problem for our next set of studies (Bray 2002).

4- Conclusion and Further Works

The presented study is one of what should eventually be an ensemble of empirical studies of the influence of class structure concept on program comprehension for novice OO programmers. The study was able to show that the *Class* based program can be more comprehendible than the *Non-Class* based program for novices. Introducing class structure concept can facilitate the program comprehension. It appears that the OO paradigm, with its emphasis on objects and relationships of objects, may result in a construction of a strong mental representation for a certain type of a problem. This is consistent with the naturalness claims of advocates of the OO paradigm. We can judge that our findings support the argument about the cognitive benefits of the OO paradigm and their affect on the novice comprehension. The study was able to confirm that the *Class* based programs can be more comprehensible than the *Non-Class* based programs for novices on the "car programs".

The study also suggests that the *Class* based treatment facilitates the reverse mapping from the program domain to the problem domain, especially for programs which are best understood in the *Class* based form. These programs' entities already exist in the real world. In order to determine whether this flow actually occurs, further research is required by using programs with different problem characteristics. We have identified Jackson as a good classification of problem types and we will using a different problem types for our next study.

The study finds clear evidence of differences in mental representation between the *Class* based and *Non-class* based programs. Considering the performance in the knowledge categories illustrated in table 1 on the *Class* based and *Non-Class* based treatments, we have only found differences in the class category. However, incorporating all the Burkhardt, et.al models' categories will be on the expense of the program size and complexity and will be beyond the scope of the proficiency level of our target subjects. We have briefly mentioned the pitfalls and problems of conducting large size and/or competing hierarchal structure problems for the future study.

Acknowledgement

We would like to thank all the students and the lecturers at Sheffield-Hallam University and Faculty of Electronics and Faculty of Computer Technologies in Libya participated in the 3 set of studies reported here.

5- References

Alardawi, A., Khazaei, B., & Jawed, S. (2011). Empirical study of novice comprehension of object-oriented OO programs. *PPIG2010 Work in Progress,*

Bray, I., & Bray, I. K. (2002). *An introduction to requirements engineering* Addison Wesley.

Brooks, R. (1978). Using a behavioural theory of program comprehension in software engineering. *Proceedings of the 3rd International Conference on Software Engineering,* 196-201.

Burkhardt, J. M., Détienne, F., & Wiedenbeck, S. (2006). Mental representations constructed by experts and novices in object-oriented program comprehension. *Arxiv Preprint cs/0612018,*

Détienne, F. (2006). Assessing the cognitive consequences of the object-oriented approach: A survey of empirical research on object-oriented design by individuals and teams. *Arxiv Preprint cs/0611154,*

Johnson-Laird, P. N. (1986). *Mental models: Towards a cognitive science of language, inference, and consciousness* Harvard Univ Pr.

Khazaei, B., & Jackson, M. (2002). Is there any difference in novice comprehension of a small program written in the event-driven and object-oriented styles? *Human Centric Computing Languages and Environments, 2002. Proceedings. IEEE 2002 Symposia on,* 19-26.

Kintsch, W., & Van Dijk, T. A. (1978). Toward a model of text comprehension and production. *Psychological Review, 85*(5), 363.

Pennington, N. (1987). Comprehension strategies in programming. *Empirical Studies of Programmers: Second Workshop,* 100-113.

Pennington, N. (1987). Stimulus structures and mental representations in expert comprehension of computer programs* 1. *Cognitive Psychology, 19*(3), 295-341.

Pfleeger, S. L., & Atlee, J. M. (2006). *Software engineering: Theory and practice* Prentice hall.

Ramalingam, V., & Wiedenbeck, S. (1997). An empirical study of novice program comprehension in the imperative and object-oriented styles. *Papers Presented at the Seventh Workshop on Empirical Studies of Programmers,* 124-139.

Sajaniemi, J., & Kuittinen, M. (2007). From procedures to objects: What have we (not) done. *Proceedings of the 19th Annual Workshop of the Psychology of Programming Interest Group,* 86–100.

Van Dijk, T. A., & Kintsch, W. (1983). Strategies of discourse comprehension.

Wiedenbeck, S., & Ramalingam, V. (1999). Novice comprehension of small programs written in the procedural and object-oriented styles. *International Journal of Human-Computer Studies, 51*(1), 71-87.

Wiedenbeck, S., Ramalingam, V., Sarasamma, S., & Corritore, C. L. (1999). A comparison of the comprehension of object-oriented and procedural programs by novice programmers. *Interacting with Computers, 11*(3), 255-282.

Appendix A

A1- Class based Program Version

```
Class Engine ' Beginning of Engine class
    'Declare Engine class Attributes
    Private Power As Integer
    ' Class Methods and behaviour
    Public Sub Set_Engine()
    Console.WriteLine("Enter the engine's power")
    ' Assigne the Power value of the engine
    Power = Val(Console.ReadLine())
    End Sub
    Public Sub Engine_Describe()
        Console.WriteLine("Engine power is ="&Power)
    End Sub
End Class ' End of class Engine
```

```
Class Body ' Beginning of Body class
    ' Declare Body class Attributes
    Private Brand As String
    ' Class Methods and behaviour
    Public Sub Set_Body()
    Console.WriteLine("Enter the Body's Brand")
    ' Assigne the Brand value of the engine
    Brand = Console.ReadLine()
    End Sub
    Public Sub Body_Describe()
      Console.WriteLine("Car Brand is: " & Brand)
    End Sub
End Class ' End of class Body
```

```
Class Car ' Beginning of Car class
    Private Passengers, Speed As Integer 'Declare Car class Attributes
    Private CEngine As New Engine ' Creates new instant of class Engine
    Private CBody As New Body ' Creates new instant of class Body
    'Class Methods and behaviour
    Public Sub Set_Car()
        CEngine.Set_Engine() 'Instantiate Engine object
        CBody.Set_Body() 'Instantiate Body object
    End Sub
    Public Sub Car_Describe()
        CEngine.Engine_Describe()
        CBody.Body_Describe()
    End Sub
    Public Sub Car_Status()
        Console.WriteLine("Enter the No.of.Passengers")
        Passengers = Val(Console.ReadLine())
        If Passengers = 0 Then
            Console.WriteLine("Car is Stopping")
        Else
            Console.WriteLine("Enter the Car Speed")
            Speed = Val(Console.ReadLine())
            If Speed > 50 Then
                Console.WriteLine("Over Speed")
            Else
                Console.WriteLine("Within Normal Speed")
            End If
        End If
    End Sub
End Class
```

```
'the main program start here
Module Module1
    Sub Main()
        Dim CCar1 As New Car 'Create new instance
        CCar1.Set_Car()
        CCar1.Car_Describe()
        CCar1.Car_Status()
        Dim CCar2 As New Car 'Create new instance
        CCar2.Set_Car()
        CCar2.Car_Describe()
        Console.ReadLine()
    End Sub
End Module
```

Comprehension Questions:

1. Does the user assign a value to variable "Body"? (Elementary Operations)
2. In "Car_Status" method in class "Car", does "Speed" value assigned in the case of "Passengers" =zero? (Control Flow)
3. Does the value of "Passengers" affect the value of "Speed"? (Data Flow)
4. Does the program allow you to change the car specifications (Type / Power)? (Functions)
5. Does the program defined class Body? (Problem Classes)
6. When the “Over Speed” statement is reached, is the value of "Speed" = 50? (State)

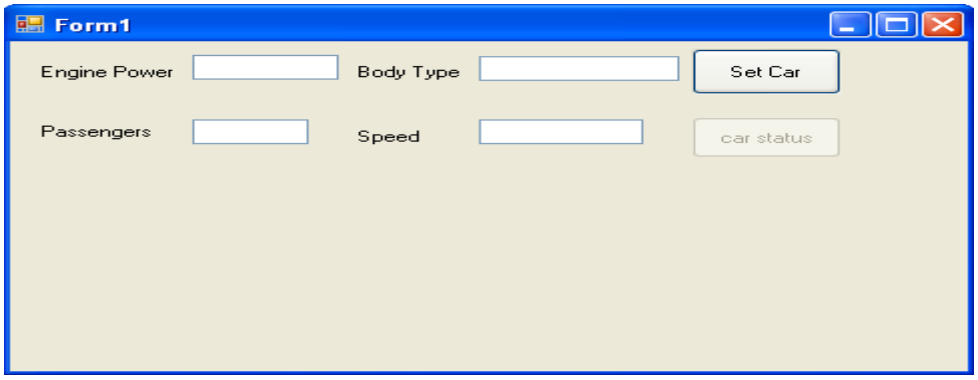
A2- Non-Class based Program Version

```
Public Class Car_Program
    Private Sub Set_Car_Click(… ) Handles Set_Car.Click
        Dim Power As Integer
        Dim Type As String
        ' Assigne the Power value of the engine
        Power = Val(TextPower.Text)
        ' Assigne the type value of the body
        Type = TextType.Text
        ' Discribe Car's specification
        MessageBox.Show("You have created car" & Type &" Its engine power="&Power)
        Car_status.Enabled = True

    End Sub
    Private Sub Car_status_Click(… ) Handles Car_status.Click
        Dim Passengers, Speed As Integer
        ' Assigne the No.of.Passengers valus
        Passengers = Val(TextPassengers.Text)
        If Passengers = 0 Then
            MessageBox.Show("Car is Stopping")
        Else
            ' Assigne the Speed value of the car
            Speed = Val(TextSpeed.Text)
            If Speed > 50 Then
                MessageBox.Show("Over Speed")
            Else
                MessageBox.Show("Within Normal Speed")
            End If
        End If

    End Sub

End Class
```



Comprehension Questions

1. Does the user assign a value to variable "Body"? (Elementary Operations)
2. In "Car_Status" method, does "Speed" value assigned in the case of "Passengers" =zero? (Control Flow)
3. Does the value of "Passengers" affect the value of "Speed"? (Data Flow)
4. Does the program allow you to change the car specifications (Type/Power)? (Functions)
5. Would class Body be used in designing the same program in Object oriented style? (Problem Classes)
6. When the “Over Speed” statement is reached, is the value of "Speed" = 50? (State)