Object-Oriented Design Process Model

***Jamilah Din*[1] , *Sufian Idris*[2]**

[1]Universiti Putra Malaysia, 43300 UPM Serdang, Selangor, Malaysia.
[2]Universiti Kebangsaan Malaysia, 43600 UKM Bangi Selangor, Malaysia.

Summary
Design is a first step in the development phase for any engineered product or system. It is defined as the process and strategies used to manage complexity. Software design process is an iterative process whereby the requirements are transformed into a "blueprint" for constructing the software. A design model is developed based on the combination of intuition and judgment, a set of principles and heuristics, and a process of iteration that leads to final design specifications. Without a proper design, a software system may fail to deliver its intended service and often will lead to some consuming maintenance activities. Therefore it is necessary for software developers to do the design process thoroughly before they start implementing the system. Object-oriented design is not an easy task. It is even difficult for a novice designer or for an experienced designer who wants to shift to object-oriented approach. Throughout literature, there are varying schools of thought on what constitutes object-oriented design. What is the process involved in this phase and what are components or structures? This paper presents four popular object-oriented design methods, and then a process model of object-oriented design for novice designer is proposed. The model consists of a process and four components. The process model is part of the model of a guidance system to assist novice designers in designing object-oriented systems.

Key words:
Object-oriented design, process model

1. Introduction

There are three important phases in developing software: analysis, design, and implementation. The analysis phase is where the developer determines what the proposed software is supposed to do; in other words, try to get comprehensive requirements of the system. The design phase will provide the detailed specifications of how the software should be developed. The latter is more technical than analysis specification, and at this point of time, the developer needs to think about how the software can realize the requirements into a working system. And lastly, the implementation phase is where the programmer develops the system based on the blueprint from the design phase using appropriate tools and programming languages.

Object-Oriented Design (OOD) is a process where the designer synthesizes all the requirements gathered during analysis phase and maps them to objects and relationships between the objects. It is a difficult process because the designer needs to synthesize from a variety of requirements and constraints to create a new structure [1]. It is also not easy for a novice designer or for a designer who wants to shift from structured approach to object-oriented [2]. Nevertheless, all designers need to learn strategies and techniques to improve their design simply through applying concepts, principles and heuristics.

OOD is benefited in terms of maintainability through which software can be modified to correct faults or to adapt to changed environments; reusability of the design artifacts and productivity gains through direct mapping to Object-oriented Programming Language (OOPL) [3].

In this paper, we define a model as a design process and a set of components that support the process. The term 'design process' refers to the activities involved in the design phase, and 'components' are the elements in the process. Designing a model is a complex task with several issues that have to be overcome and the model requires a good documentation to be actually reused. This paper discusses part of our on-going research work in developing tools for assisting novice designers build OO designs. In this paper we present our proposed process model, OOD-PM, which is targeted for novice designers. It will be used as the process model for a guidance system that the authors intend to develop to guide novice designers in designing object-oriented systems.

The rest of this paper is organized as follows. Section 2 reviews a few OOD process models that are still widely used. Section 3 discusses who the object-oriented novice designer is. Section 4 discusses OOD-PM, the proposed process model for novice designers. Finally Section 5 will conclude this paper.

2. Related work

This section discusses related works that form an important background of this research. Design process is a process whereby the design activities are performed to produce a design model that satisfies the given requirement specifications. Several design processes [4, 5, 6, and 7] have been proposed during the last three decades. Those processes claimed and have proven, in a certain case, to be good processes to follow. Each process involves the

Manuscript received October 5, 2009
Manuscript revised October 20, 2009

identification of classes, responsibilities and hierarchies. Those are the basic components in OOD. This section focuses on the process models in OOD. These four process models are chosen because most academicians use these models in introducing OOD to students who are novice designers which the proposed process model is targeted for. These four process models become a background to the proposed process model (OOD-PM)

2.1 The Responsibility-Driven Design

The first and foremost process model is defined by Wirfs-Brock [4]. The Responsibility-Driven Design (RDD), as shown in Fig.1, looks at responsibility instead of the data, which is commonly used by designers. RDD divides OOD into two distinct phases.

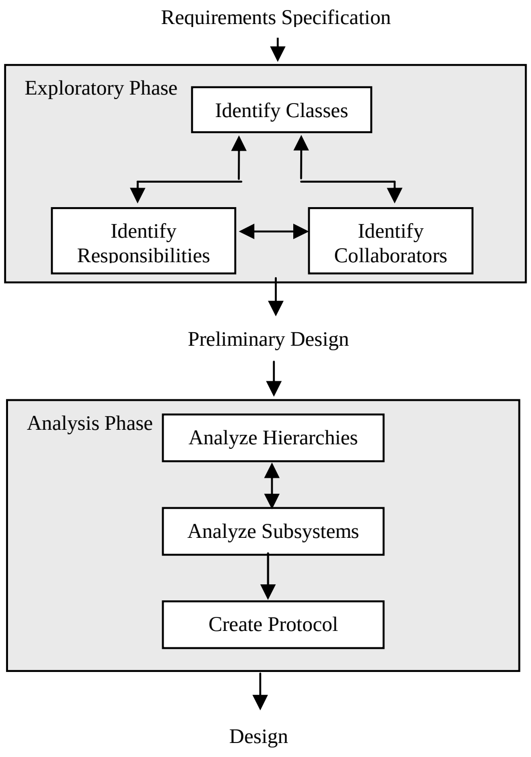

Fig. 1 Responsibility-Driven Design

The model is developed from requirements specification by extracting nouns and verbs. Responsibilities are determined before collaborations. The output of this phase is called a preliminary design. In the second phase, responsibilities are factored into hierarchies, collaborations are detailed, and the protocols determined. The most important aspect in this phase is to give full attention to structuring abstract and concrete classes before improving the collaborations. This process stops at the hierarchies, which is sufficient for design activity. This process model is easy to follow but it lacks the detailed design, which includes the implementation details. The proposed model (OOD-PM) used this model as a guideline.

2.2 Object-Oriented Analysis and Design

The idea in RDD is also present in Booch's process model [5] but Booch suggested an iterative approach, instead of top-down. In Booch's OOAD, design is an incremental and iterative process. It is composed of four steps; identify classes, identify semantics of classes, identify relationships among the classes, and lastly specify interfaces and implementations. Even though this process is widely used, designers still need to find many important details of how these complex, major activities are to be done. Moreover it works well for simple to moderate design specifications, and it will become costly for more complex systems.

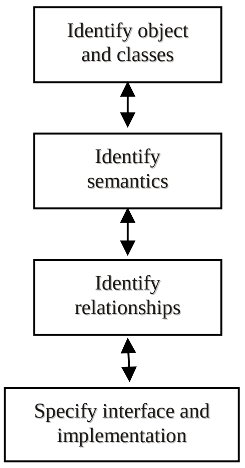
Identify object and classes
Identify semantics
Identify relationships
Specify interface and implementation

Fig.2 Object-Oriented Analysis Design

2.3 Object-Oriented Software Engineering (OOSE)

OOSE [6] is another popular process model. This process considers adaptations to make idealized analysis model fit the real world environment, and then create blocks (design abstraction that allows for the presentation of an aggregate object) as the primary design object. The next step is to create interaction diagrams that show how stimuli are passed between blocks before organizing the blocks into subsystems. The design work is then reviewed. This model does not include the implementation part of the design. It is easy to follow but does not have implementation details. Fig 3 shows the process of this model.

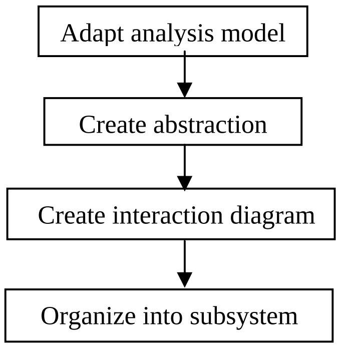

Fig. 3 OOSE

2.4 Object Modeling Technique (OMT)

Another object-oriented design approach, which is also widely used, is Object Modeling Technique (OMT), developed by Rumbaugh [7]. OMT, as shown in Fig 4, has a simple notation and supports the structural, behavioral, and functional aspects of a system. Those aspects are represented respectively by the object model, the dynamic model, the functional model. OMT divides the design phase into two stages: system design and object design. During system design, the designer designs the overall architecture of the system, followed by the adding of details about the implementation. Many design models and CASE tools have been developed based on this approach [ref?]. It is also a commonly taught object-oriented methodology in academic settings.

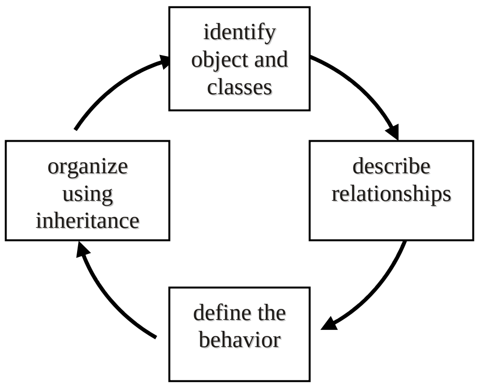

Fig. 4 Object Modeling Technique

Table 1 summarizes the design processes of the four process models discussed above. These four process models become a background to the suggested process model of our work.

Table 1: Design Processes

Object-Oriented Process Model	Design Process
Designing Object-Oriented Software Method [4]	• construct protocols (formal description of the messages to which a class will respond) for each class • create a design specification for each class • create a design specification for each subsystem
Object-Oriented Analysis and Design With Applications [5]	• architectural planning • tactical design • release planning
Object-Oriented Software Engineering [6]	• consider adaptations to make idealized analysis model • create blocks (design abstraction) as the primary design object • create an interaction diagram • organize blocks into subsystems • review the design work
Object Modeling Technique (OMT) [7]	• perform system design • conduct object design • implement control mechanisms defined in system design • adjust class structure to strengthen inheritance • design messaging • package classes and associations

3. Novice Designers

There are a few studies which describe novice designers as undergraduate students or new object-oriented designers who have no industrial experience [8, 9]. However, in the field of object-oriented design, it is not yet clear how experts and novices differ in terms of cognition, despite the fact that expert and novice behavior can be observed in terms of actual outcomes. In order to understand how to support the designer, in particular novice designer, a study into how novice designers approach their design task was reviewed by Suppapitnarm & Ahmed [10]. Their study showed that novice designers take considerable effort and time to pick up tacit knowledge in the design development process. From the literature study discussed above, this study refers novice designers as new designers or experienced non-object-oriented designers who are shifting to object-oriented approach. This is because novices need to be guided in order to produce better designs and the best practices from the experts can be used in guiding them. The next section provides a discussion on the common problems in designing object-oriented systems among novice designers.

3.1 Problems in Designing Object-Oriented Systems among Novice Designers

This section surveys the current study on the problems of designing object-oriented software among novice designers. The focus of this study is on novice designers. The definition of the novice designer and the literatures that support it has been discussed in the previous subsection. Several empirical studies have looked at the understanding and misconception in object-oriented approach among novices. However, many of the works are based on student programming [11, 12, 13, 14], and only a few study have been done on students designing simple object-oriented programs [15. 16].

Ryan [9] has done an empirical study on designer behavior. The study wanted to show the difference in cognitive activities and design strategies between both novice and expert object-oriented designer. The object-oriented expert designers were designers with more than seven years of experience in developing object-oriented systems. The novices were designers with no object-oriented experience, though they might have experience in procedural approach. Even though the study showed that there is no difference between these two groups in terms of generalization, object-oriented design is still considered difficult especially to novices [2, 17, 18, 19].

Blaha et al [20] have done a “scaffolding” experiment in a multi-national, multi-institutional study that looked at several aspects of software design. The study involved 21 post-secondary institutions in the US, UK, Sweden, and New Zealand. The subjects in this study were divided into three groups: competent students, graduating seniors and educators. They were needed to perform two tasks: software decomposition and design criteria prioritization. For the design task, they were given a brief description of a “super alarm clock” system. The study reported that there was a significant trend in the group in increasing use of standard graphical representation and increased in recognition of ambiguity. This study raised important questions for educators about how they can help the student in recognizing ambiguities in specification.

The study by Or-Bach & Lavy [15] reveals that students have difficulties in the concepts of abstraction, inheritance and polymorphism. This report shares the same finding as other studies [21]. Sim & Wright [21] reported the difficulties of students in learning object-oriented concepts of object-oriented analysis and design. The study has shown that students have difficulties in understanding and modeling the behavioral aspect of object-oriented analysis and design, especially concerning messages and operations.

The recent study by Eckerdal et al [13] has proven that, object-oriented design is still a difficult task to novice designers. The study has been done on near-graduating senior students by examining the design artifacts produced by these students. The result showed that about 60% of Computer Science graduates fail to design simple software systems.

Gibbon [22] in his thesis has listed common design mistakes made by the novice designers, which are:

- Large number of attributes
- Large number of methods
- Poor encapsulation
- Behavior rich or God Class – centralized design architecture (collaborate with numerous system classes to fulfill their responsibility)
- Behavior poor (inert class) – defines only get and set methods.

Based on the literature study elaborated above, problems in designing object-oriented system can be categorized into two groups:

1. basic concepts of object-oriented design :
   - problem in designing class – with large number of attributes and methods
   - God Class – classes with many responsibility
2. advanced concepts of object-oriented design
   - abstraction
   - encapsulation
   - inheritance

This study focuses on these two categories in order to guide the novices in designing object-oriented system.

4. OOD Process Model

The Object-Oriented Design process model considers the common design faults among novice designers as has been discussed previously. Considering those common faults and what other guidance system that support the design phase have done and also combining the other methodologies available from literature, the OOD-PM process model is proposed. Before discussing OOD-PM in detail, the following section presents its elements which are based on the concepts and definitions defined in the model.

4.1 Elements in the OOD-PM

The process model has many elements that can be viewed as static elements. This section defines those elements which will explain the terminology used in the discussion about OOD-PM in the next section. The elements are discussed according to their categories.

Class: A class is an abstraction of similar objects. A class is a set of objects that share a common structure, common behavior, and common semantics [23]. During the analysis stage, individual object instances are identified and then grouped into general classes. A class is defined by defining the attributes and methods of its objects.

Attributes refer to data belonging to a class. Collectively, the attributes of an object represent its state. There are three types of attributes: single-valued, multiplicity or multivalued attribute. There are also attributes that refer to other objects or instance connections.

Instances of a class exhibit behavior when they execute their methods. A method contains logic operations of the class.

Class Collaboration: Class collaboration refers to the cooperation between a class and other classes in implementing its responsibilities. Responsibilities are implemented in terms of methods. For each method, a class can perform the method on its own, or it needs to collaborate and communicate with other classes.

Responsibility is a function that a class needs to fulfill. A class should not have too many responsibilities. This is the principle of cohesion [24]. Cohesion reflects the single purpose of a class. Highly cohesive components can reflect lower coupling between classes because only minimal information needs to be passed between them. The concept of class cohesion means that all the class's methods and attributes to be used by internal methods and derived classes' methods.

Interaction means the relation of a class and other class. Classes communicate through messaging. Message is a request service that an object sends to another object and communication is the collaboration between classes in order to perform a responsibility.

Class Relationship: Abstraction is a process of identifying the similarities between classes. Abstraction is defined as "the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer" [25]. During the abstraction process, the similarities of the classes are identified and this helps in the process of dividing the classes into hierarchies. The allowable behavior, which is the outside view of the object, is determined for the abstraction.

There are three relationships that are common among classes [26]. The paragraphs below discuss those relationships.

- Association is a general dependency relationship between classes. For example, a class depends on another class if it manipulates objects of the other class. The UML notation for this relationship is just a plain line between the classes, as shown in Fig 5.

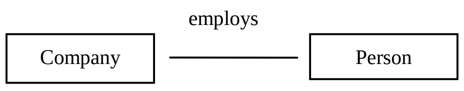

Fig 5 Association Relationship

- Aggregation is the association of a class with other component classes. It is also sometimes known as whole-part relationships or composition. In the UML notation, the line representing the relationship has a filled diamond at the "whole" end. An example of aggregation relationship is depicted in Fig. 6.

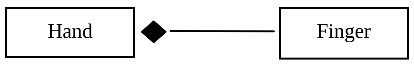

Fig 6 Aggregation Relationship

- Inheritance is the relationship that exists when a class inherits attributes and behavior from another class. The most general class will become the super class and other class is the sub-class. The sub-class inherits all the data and behavior of its parent, besides having its own data and methods. In the UML notation, the super-sub class relationship is represented as a line with a hollow triangle at the "super class" end. Fig 7 illustrates an example of inheritance.

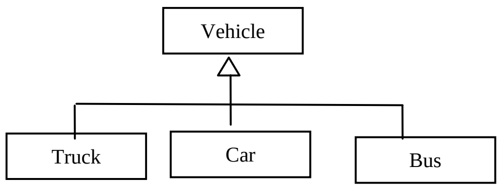

Fig 7 Inheritance Relationship

Another concept related to class relationship is visibility. "Visibility is the ability of one object to "see" or have a reference to another object" [27]. The protocol is the interface of the class operation and its visibility. One class need to be visible to another class in order to collaborate. The protocol can be either the member is public, private or protected. A class can be visible (has public protocol) to other classes or it can be hidden (has private protocol) from other classes. The visibility is depends on the type of relationship.

Class Encapsulation: Encapsulation is one of the most important principles in object-orientation. It means that the object includes both data and operations on the data (methods). It is also known as information hiding. This is important not only in programming, but also in analysis and design. Most traditional methodologies separate the data and functional components of the system. Encapsulation is intended to reduce coupling among modules. The less the other object knows about the implementation of the module, the looser the coupling between the module and its client can be. Even though coupling is a concept in structured approach, it is a good design concept that is embedded in the object-oriented approach. Other object can only ask an object for services, but cannot directly access the object's state data.

Polymorphism means the ability to take more than one form. Through polymorphism, it is possible to hide implementation behind the same interface. Thus, an object can send a message to another object without necessarily knowing the precise class to which the object belongs. For example, the message "move" might be sent to a vehicle object, without knowing whether the object is a car, a motorcycle, or an airplane. Each of these different types of vehicles knows how to interpret and carry out the "move" message. It is a capability of a single variable to refer to different objects that fulfill certain message protocol responsibilities (roles). By using polymorphism, the designer can build reusable classes that contribute to reusable object-oriented design.

4.2 The Model - OOD-PM

The proposed model, OOD-PM,is mainly derived and concluded from Wirfs-Brock [4], Booch [5], Jacobson [6] and Rumbaugh [7]. This model shows the process and components involved in the process.

Fig. 8 shows the OOD-PM model for novice designers. The object-oriented design process starts with the identification of classes, which are abstractions of similar objects. Objects are determined during the analysis phase. Those common objects are grouped into classes during the design phase. A class has attributes (data) and methods (behavior) and also identity. Even though the attributes and methods have been discovered during the analysis phase, they should be refined in the design phase to get more comprehensive attributes and methods for the classes.

After defining the classes, the designer can then start identifying class collaborations. It involves identifying class relationships and responsibilities. Discovering a responsibility means identifying a class (and only one class) that owns that responsibility. The purpose is to produce a highly cohesive design model. For each method, either the class can perform on its own, or it needs to collaborate and communicate with other classes. In order for a class to communicate, its instances need to send/receive messages to instances of other classes. Messaging and communication are closely-related concepts. When there is a message passing between class instances, communication exists between the corresponding classes.

The next step is to determine relationships between classes. A class relationship is how the class interacts with other classes. The first activity is to do abstraction, before categorizing the classes into hierarchies; whether dependency, aggregation or inheritance. The abstraction identifies similarities between classes and this helps in the process of dividing the classes into hierarchies. The allowable behavior, which is the outside view of the object, is determined for the abstraction. At this point, the designer might decide to introduce new classes, so the process goes back to the first activity. As with most object-oriented design models, the proposed model is also an iterative process. From the abstraction, three common relationships can be identified. Dependency is the relationship when a class manipulates objects of the other class. Aggregation is the relationship when a class is part of the other class, and inheritance is when a class shares the structure or behavior defined in the other class. To complete this activity, the protocols of the classes need to be determined. Protocols show the specific signature of each responsibility.

The abstraction of the class should precede the decision about its implementation. The implementation is treated as a secret of the abstraction. It is a detailed design. Abstraction and encapsulation are closely-related concepts [25]. Abstraction focuses on the outside view of a class and encapsulation helps manage the complexity by hiding the inside view of the abstraction. Liskov [28] suggests, "for abstraction to work, implementations must be encapsulated". Information hiding can be used to implement encapsulation. It means the object includes both data and operations on the data. The concept of encapsulation is intended to reduce coupling among modules. The less the object knows about the implementation of the module, the looser the coupling between the module and its client to be. Coupling is a good design concept that is embedded in the object-oriented approach. Another activity is to apply polymorphism in the design.. Polymorphism enables a message to refer to different methods depending on the actual type of the receiver object. Applying polymorphism can improve the flexibility and extendibility of a design.

The output of OOD-PM is a complete design for an object-oriented system. OOD-PM is the process model which will be incorporated in our OOD guidance system for novice designers. The model has been discussed with experts in order to ascertain the feasibility and reliability of the model to novice designers. A number of experts and academicians who have taught the object-oriented approach have given feedback on the model. In addition, the model has also been presented in a software engineering conference. Based on discussions and comments, the model has been further refined and enhanced.

A comparison of OOD-PM with the four process models discussed in section 2 is shown in Table 2.

Table 2: comparison of elements in the OOD process model

Elements	RDD	Booch's	OOSE	OMT	OOD-PM
Class	x	x	x	x	x
Attribute	x	x	x	x	x
Method	x	x	x	x	x
Collaboration	x		x	x	x
Abstraction	x		x	x	x
Relationship	x	x	x	x	x
Visibility	x			x	x
interface		x			
subsystem			x		
Information hiding		x			x
Polymorphism		x			x

5. Conclusion

In software engineering, design plays an important role. The design phase has contributed to highest rate of software failure. A proper process model in designing a system is required to assist novices in producing better designs. In this context, we proposed a process model called object-oriented design process model (OOD-PM). This process model is suitable for novice designers because it is systematic and covers most object-oriented design components. This model is incorporated in the model of a guidance system to assist novice designers in designing object-oriented systems.

References

[1] Biddle, R., “A Lightweight CASE Tool for Learning OO Design”, Proceedings of Oopsla 2000 Educators Symposium, pp. 78-83, 2000.

[2] Gamma, E., Helm, R., Johnson, R. and Vlissides, J., Design Patterns: Element of Reusable Object-oriented Software, Addison-Wesley, 1994.

[3] Lewis, T.L., Perez-Quinones, M.A. and Rosson, M.B., “A Comprehensive Analysis of Object-Oriented Design: Towards A Measure of Assessing Design Ability”, Proceedings of 34th ASEE/IEEE Frontiers in Education Conference, 2004.

[4] Wirfs-Brock, R.J., Surveying Current Research In Object-Oriented Design. Communication of the ACM, 33(9), pp. 104-124, 1990.

[5] Booch, G., Object-Oriented Analysis and Design with Applications, The Benjamin/Cummings Publishing Company, Inc, 1994.

[6] Jacobson, I., Object-oriented Software Engineering, ACM Press, NY, 1991.

[7] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W, Object- Oriented Modeling and Design, Prentice Hall, New Jersey, 1991.

[8] Muraki, T. and Saeki, M., “Metrics for Applying GOF Design Patterns in Refactoring Processes”, Proceedings of 4th International Workshop on Principles of Software Evolution (IWPSE 2001), pp. 27-36, 2001.

[9] Ryan C., A Methodology for the Empirical Study of Object-Oriented Designers. PhD Dissertation. RMIT University, Melbourne, Australia, 2002.

[10] Suppapitnarm, A. & Ahmed, S. e-Learning from Knowledge and Experience Capture in Design. Proceeding of the The First National Conference on Electronic Business (NCEB2002). 2002.

[11] Garner, S., Haden, P. & Robins, A. My Program is Correct But it Doesn’t Run: A Preliminary Investigation of Novice Programmers’ Problems. Proceeding of 7th Australasian Conference on Computing Education, pp 173 – 180, 2005.

[12] Robins, A., Haden, P. & Garner, S. Problem Distributions in a CS1 Course. Proceeding of the Eighth Australasian Computing Education Conference (ACE2006), pp 165-173, 2006.

[13] Eckerdal, A., McCartney, R., Mostr¨om, J. E., Ratcliffe, M. & Zander, C. Can Graduating Students Design Software Systems? Proceeding of the Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education, pp 403 - 407. 2006.

[14] Hanks, B. Problems Encountered by Novice Pair Programmers. Journal on Educational Resources in Computing (JERIC) 7(4), 2008.

[15] Or-Bach, R. & Lavy, I., Cognitive activities of abstraction in object orientation: an empirical study. ACM SIGCSE Bulletin 2: 82 – 86, 2004.

[16] Thomasson, B., Ratcliffe, M. & Thomas, L. Identifying Novice Difficulties in Object Oriented Design. Proceeding of the Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education, pp 28 – 32, 2006.

[17] Opdyke, W. F. & Johnson, R. E. Creating Abstract Superclass by Refactoring. Proceeding of the ACM Confference on Computer Science, pp 66-73, 1993.

[18] Marinescu, R. Measurement and Quality in Object-Oriented Design. Thesis Doctor of Philosophy in Computer Science. ”Politehnica” University of Timi¸soara. 2002.

[19] Moynihan, G. P., Suki, A. & Fonseca, D. J. An expert system for the selection of software design patterns. Expert Systems 23(1): 39-52, 2006.

[20] Blaha, K., Monge, A., Sanders, D., Simon, B. & VanDeGrift, T. Do Students Recognize Ambiguity in Software Design? A Multi-national, Multi-institutional Report. Proceeding of the 27th International Conference on

Software Engineering, 2005 (ICSE 2005), pp 615-616, 2005.
[21] Sim, E. R. & Wright, G. The Difficulties of Learning Object-Oriented Analysis and Design: An Exploratory. Journal of Computer Information Systems Winter 2001-2002): 95-100, 2002.
[22] Gibbon, C. A. Heuristics for Object-Oriented Design. Thesis Doctor of Philosophy. University of Nottingham, 1997.
[23] Booch, G., Maksimchuk, R. A., Engle, M. W., Young, B. J., Conallen, J. & Houston, K. A. Object-Oriented Analysis and Design with Application. Addison-Wesley. 2007.
[24] Braude, E. J. Software Design: From Programming to Architecture. Wiley. 2003.
[25] Booch, G. Rules of Thumb. Report on Object Analysis and Design (ROAD) 2(4): 2-3. 1995.
[26] Bahrami, A. Object-Oriented System Development. Irwin/McGraw-Hill. 1999.
[27] Larman, C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Prentice Hall. 2005.
[28] Liskov, B. Data Abstraction and Hierarchy. Proceeding of the Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA '87), pp 17-34. 1988.

Jamilah Din received the B.S. in Computing Science from University of Evansville, Indiana in 1987 and MSc in Computer Science from Putra University of Malaysia (UPM) in 2002. She is now a lecturer at Faculty of Computer Science and Information Technology, UPM and also pursuing her PhD in Software Engineering at National University of Malaysia (UKM).

Sufian Idris is a faculty member in the Computer Science Department of the Faculty of Information Science and Technology at National University of Malaysia (NUM). He holds an MSc in Computer Science from the University of Manchester, UK and a PhD in Computer Science from the University of Manchester Institute of Science and Technology, UK. He has been a faculty member in NUM since 1988. His teaching interests are in the areas of Programming, and Object-Oriented Analysis and Design. His research interests and work are in the areas of Teaching of Programming, Programming Tools, Object-Oriented Development and Mobile Computing.

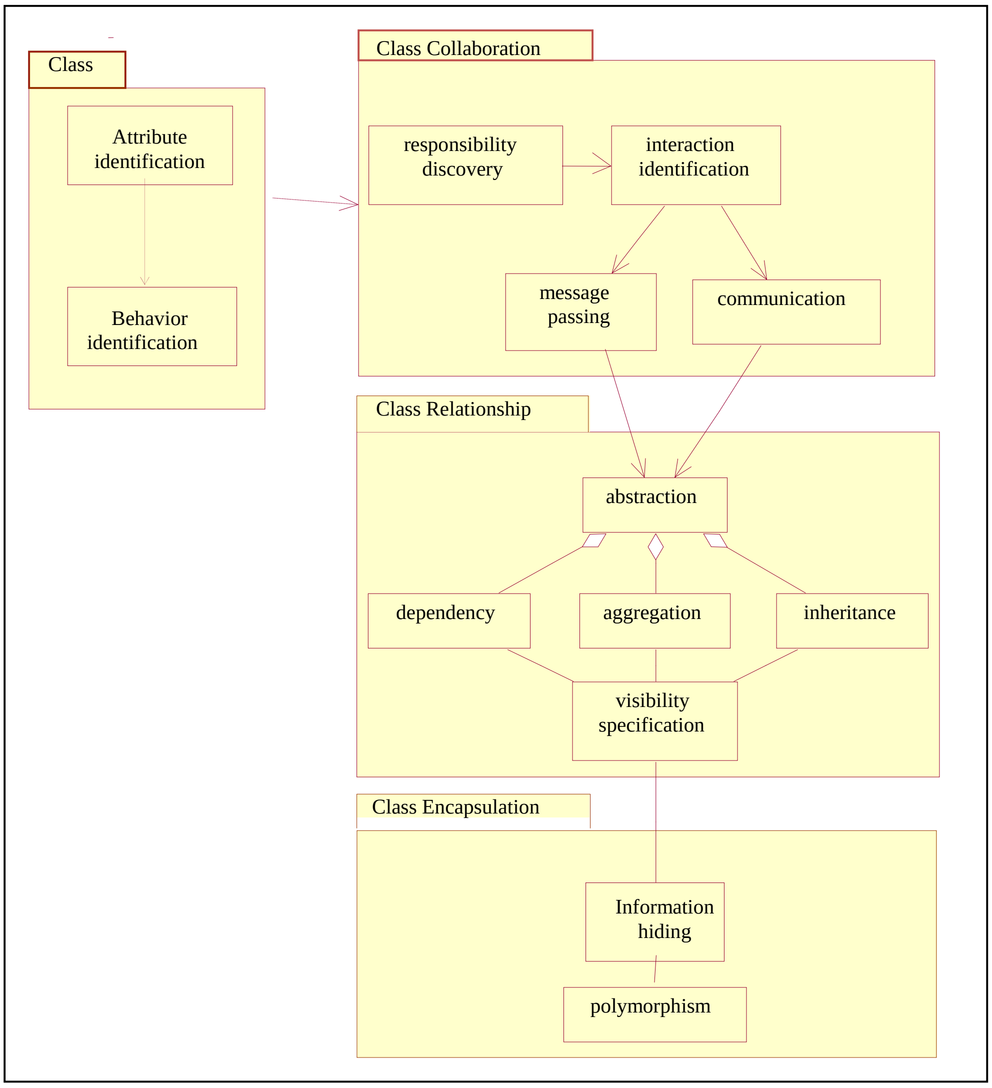
Class
Attribute identification
Behavior identification
Class Collaboration
responsibility discovery
interaction identification
message passing
communication
Class Relationship
abstraction
dependency
aggregation
inheritance
visibility specification
Class Encapsulation
Information hiding
polymorphism

Fig 8 OOD-PM