

A Comprehensive Analysis of Object-Oriented Design: Towards A Measure of Assessing Design Ability

Tracy L. Lewis¹, Manuel A. Pérez-Quiñones² and Mary Beth Rosson³

Abstract - Throughout literature, there are varying schools of thought on what constitutes object-oriented design (OOD). Does OOD encompass the selection of the appropriate interfaces? Does reusing classes represent “good” OOD? And what about UML, if we have class interaction diagrams, is this OOD? There is a need to establish a set of requisite competencies to guide teaching of OOD. Within this paper, we present a comprehensive analysis of seven essential principles of object-oriented design.

In order to validate the effectiveness of the guiding principles OOD, we developed the Design Readiness Assessment Scale (DRAS). The DRAS provides a classification of one’s design ability in relation to the testing population. We further discuss a study of the DRAS using a subject pool consisting of freshmen computer science (novice designers) and graduate/professionals (expert designers). We present the preliminary results of this study and its impact on the future OOD education.

Index Terms – Assessment, Design, Design Readiness, Object-Oriented Design

INTRODUCTION

In many disciplines, particularly computer science and engineering, design activities utilize the regimented approach to invent a solution for some problem. Design is defined as the process and strategies used to manage software system complexity [27]. The products of design activities are models that enable us to reason about our structures, make trade-offs when requirements conflict, and provide blueprints for implementation [4].

Producing a design that captures the requirements and constraints necessary to be useful is a meandering task, only mastered through practical experience. Even experienced professionals may never become “expert” designers. Nevertheless, all professionals learn strategies and techniques for improving their design approaches simply through applying principles, extensive experience, feedback, and lessons learned from other designers.

Object-oriented design (OOD) is a highly regarded, special set of design skills used as a technique to create software models that simulate “real world” actions and conditions. Most people view the world as a series of objects interacting with each other, and conceptualize those interacting objects accordingly [2]. This makes OOD an ideal technique for modeling real-world problems. The basic design strategies that are embodied in the OOD paradigm – abstraction, separation of responsibilities, composition, encapsulation, etc. – evolved as techniques for dealing with complex natural and man-made systems [15].

OOD methods provide a notation, which allows one to store and communicate intrinsic design decisions. OOD can yield the following benefits: *maintainability* through which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment; *reusability* of the design artifacts, which saves time and costs; and productivity gains through direct mapping to features of Object-Oriented Programming Languages [3].

The very strategies that make OOD successful also make it difficult to learn. Many current courses that introduce the object-oriented methodology only focus on certain programming aspects of the paradigm (classes, objects, association, aggregation) and not on design concepts (abstraction, generalizations, composition). This approach gives the student an incomplete introduction that may inhibit a proper understanding of OOD, even if the missing “design” pieces are provided in later courses [2].

OOD methods have been used in industry since the late 1980s. AT&T Bell Labs used OOD and realized the benefits of reduced product development time and increased reuse of both code and analysis/design artifacts on a large project called the Call Attempt Data Collection System (CADCS) [27]. Additionally, OOD has been used worldwide in many commercial, Department of Defense (DoD), and government applications. There exists a wealth of documentation and training courses for each of the various OOD methods, along with commercially-available CASE tools with object-oriented extensions that support these OOD methods [27].

¹ Tracy L. Lewis, Radford University, Department of Information Technology, tracyL@radford.edu

² Manuel Perez-Quinones, Virginia Tech, Department of Computer Science, perez@cs.vt.edu

³ Mary Beth Rosson, Pennsylvania State University, School of Information Science and Technology, mrosson@psu.edu

There is an academic and industry need to establish a set of requisite competencies to guide the teaching of OOD. We believe that if instructors can focus on these guiding principles of OOD, students and industry professionals will develop the necessary skills to become excellent object-oriented designers.

For the remainder of this paper, we will present an historical overview of OOD, describe what OOD is not, and then present the fundamental principles of OOD. We will conclude by using these fundamental principles to characterize a skill we call *design readiness* and present the preliminary results of a novel assessment scale created to measure design readiness. This research is guided to answer the question: Can we identify a fundamental set of principles to guide the teaching and learning of Object-oriented design?

VARYING CONCEPTUALIZATIONS OF OBJECT-ORIENTED DESIGN

The process of software design is often misconceived as a straightforward activity of problem decomposition [25],[26]. In fact, the object-oriented approach is both more abstract and more comprehensive than its predecessors, and it seems to correspondingly require more maturity and depth to grasp [20]. Northrop states that mastering object-oriented language is not the difficult part; it is the analysis and design activities that pose a challenge [20].

We will take a walk through the history of OOD to examine where OOD began, how it has progressed, and where it needs to go. Early OO visionaries agreed that a focus on design was the impetus needed to successfully launch OO paradigm. They all realized the need for the *how* to utilize OO methodologies to treat systems as a cohesive collection of interacting objects [21]. The “how-to” in the OO methodologies represents the application of design techniques within design phase of system development. There were varying schools of thought on how one should conceptualize the design of a system. With systems becoming inherently more complex, there was a need to show the level and the type of interaction between classes, localized information representation, and the protection of information within classes. We will mention some of the more popular OOD modeling approaches utilized over the past twenty years.

Booch [4] saw design as an incremental, iterative process. His approach focused on identifying the relationships of classes in a static view of the system. We can summarize Booch’s OOD approach as a four step process:

- (1) Identify the classes.
- (2) Identify the semantics.
- (3) Identify the relationships.
- (4) Specify the interface and implementation.

One approach to identifying the classes involved to have the designer underline all the nouns and employ these as the

initial classes. Throughout Booch’s four step process, designers were expected iterate through these steps until a *satisfactory* system was created. It is the ill-definition of the word *satisfactory* that makes OOD such a difficult process. We really do not know what *satisfactory* is or how to measure it. This process worked well for simple to moderate design specifications, but this process became quite costly with more complex, ill-defined system requirements [21].

The most widely used OOD approach, Object Modeling Technique (OMT), was developed by James Rumbaugh [23]. His approach has been adopted into many OO Case Tools. The actual deliverables for this approach include an Object Model, Dynamic Model, and Functional Model. The OMT approach allows designers to conceptualize the overall system architecture. First the system is organized into subsystems which are then allocated to processes and tasks, taking into account concurrency and collaboration. Then persistent data storage is established along with a strategy to manage shared-global information. Next, boundary situations are examined to help guide trade-off priorities. Instead of following the strict waterfall software engineering model, OMT suggests a cyclical approach that is taken among the smaller steps, particularly within design.

In later years, Wirfs-Brock developed the OOD modeling approach called Responsibility-Driven Design [29]. Wirfs-Brock built upon the Class-Responsibility Collaborators (CRC) work of Beck and Cunningham. For each class, different responsibilities (roles and actions) are defined. To fulfill the responsibilities of the classes, they need to demonstrate collaboration with other classes. CRC cards provide a physical artifact for the designers to manipulate and designers can actually see how the classes collaborate with each other. They allow the participants to experience first hand how the system will work by physically picking up the cards and playing the roll of that class [29]. CRC cards are simple enough for novices to use, but often times CRC cards lacked the extensibility necessary to model complex systems.

There were many more prominent approaches used to capture and accurately express OOD. All these approaches have their advantages, but the problem was that they were disparate. Some approaches explicated required designers to *think* about the analysis of system as a separate disjoint process from design, while others suggested taking a spiral approach through analysis and design. Finally, Booch, Rumbaugh, and Jacobson combined OOD modeling theories and practices to create the Unified Modeling Language (UML). UML is now the industry standard in documenting object-oriented designs. UML provides the ability to capture the characteristics of a system by using a variety of simple and sophisticated notations for documenting systems. These notations are called the nine diagrams of UML [13]. The more popular diagrams used within UML are the class, sequence, and conceptual diagrams – that’s only one-third of the UML notations capabilities. BUT, much of the

functionality of UML is too complex for novice designers to comprehend. Many novice designers don't know where to begin when asked to design classes using UML notation.

These approaches illustrate what approach/tool to use to break a system in simple cohesive units, but the looming question is how do we know what to break into cohesive units, are there certain strategies we should use when decomposing the requirements? The issue we, as academics and industry professionals, have to address is how do we effectively combine and/or apply sound OOD approaches to create an industry of proficient OO designers.

A SURVEY OF OOD LITERATURE REVIEWS

Numerous reviews [1],[6],[9],[17] exist comparing the various object-oriented approaches. Fichman [9] presents an executive summary comparing various analysis and design approaches, focusing primarily on the structured paradigm, and concluding with a description of Wasserman OO Structured Design, Booch OO Design, and Wirfs-Brock Responsibility-Driven Design. This summary was designed to transition the structured programmer into object-oriented programming. A more exhaustive OO summary was performed by Cribbs et. al [6]. They present comparison of the notations, terminologies, and models used in methods by Booch, Coad and Yourdon, Edwards and Odell and Martin, Graham, Rumbaugh, Shlaer and Mellor, Wasserman and Pircher, and Wirfs-Brock. This work also served as a transitional piece from the structure to the object oriented approach. Monarchi and Puhr [17] compared twenty three object-oriented analysis and design methodologies to identify common themes, and strengths and weaknesses. This report provided a thorough review of the state of the art for object-oriented analysis and design. G. van den Goor et. al [11] This report compares the background ideas, steps, concepts, notations, communication mechanism, and the specification techniques of six accepted methods - Booch, Martin/Odell, Coad/Yourdon, Rumbaugh, Wirfs-Brock, Shlaer/Mellor. All of the aforementioned summaries present the various OOD approaches in terms of similarities and differences. Many of the summaries presented detailed tables showing the concepts and diagrams applicable in each approach. The Object Agency, Inc. [21] is the first known attempt to develop a set of validation measures for various OOD approaches. These validation measures include: Concepts, Notations, Process, Pragmatics, Support for Software Engineering, and Marketability. The measures introduced by The Object Agency, Inc. provided a valuable approach to evaluating design techniques from a business perspective.

In summary, the previously mentioned notation and documentation strategies provide the *how to* design, but the *what to* design remains uncertain. All the reviews provide excellent historical overviews of the OO paradigm, with segments focusing on design, but there was still a need to categorize what really constitutes OOD. Just as there were

many OOD modeling approaches; there have been many attempts to characterize design [12],[14],[15],[28],[29],[30]. It is time for an industry standard on the essential components of OOD. If we can establish those techniques that form a core competency for OOD we can then develop standardized design skills assessments materials as well as universal design evaluation tools. If students and professionals can apply the essential OOD techniques, application specific learning curves will progressively decrease, thus creating an industry and academic standard of quick-to-market, high quality, reliable software. Based on the lack of consensus of what constitutes OOD, we present a model that characterizes seven principles of OOD in relation to a skill that we coined as design readiness.

GUIDING PRINCIPLES OF OBJECT-ORIENTED DESIGN

After a thorough inspection of the OOD literature [4],[12],[13],[14],[22],[28], we found a fundamental set of principles that can be applied in any OOD conceptual modeling approach:

Divide and Conquer. The first step in designing a program is to divide the overall problem into a number of classes that will interact with each other to solve the problem. Identify smaller components of the problem and solve them separately. Employ a division of labor much as we do in organizing many of our real world tasks.

Encapsulation. Once the classes are identified, the next step involves deciding, for each class, what attributes it has and what action it will take. The goal is to place within each class, the combination of data and functionality into a single entity, with the implementation hidden from external entities. Each class is designed to be a self-contained module with a clear responsibility and the skills (attributes and actions) necessary to carry out its role. In addition to knowing how to perform its role, each class has to know exactly what information it needs to obtain from its collaborators (internal and external system classes).

Generality. As long as we are designing a class to solve a problem, we should design it as general as possible. But, to design a general class requires a great deal more thought and effort than designing a narrow, single purpose class. It may not be initially obvious how a specific problem might be a special case of a general solution. Nonetheless, we should design classes not for a particular task, but rather for a particular *kind* of task.

Information Hiding. The details of each class's performance should be hidden from other classes. This also will help classes work together cooperatively and efficiently. This technique is different from encapsulation because

encapsulation is simply bundling data with the methods that operate on the data. Information hiding involves hiding difficult design decisions or design decisions that are likely to change. We should hide information in a manner that isolates clients from requiring intimate knowledge of the design of a class, and from the effects of changing design decisions.

Inheritance and Polymorphism. In general terms, Inheritance and polymorphism manages a class’s ability to use properties and abilities of another class (parent or super class), while adding its own functionality. More specifically, polymorphism may be understood in terms of inheritance as a means to define the properties of a subclass, and by adding information delimiting a subset of the elements corresponding to the parent class.

Interface. In order for classes to work cooperatively and efficiently, we have to clarify exactly how they should interact, or interface, with one another. An interface is a contract in the form of a collection of method and constant declarations. When a class implements an interface, it promises to implement ALL of the methods declared in that interface. A class’s interface should be designed to protect its integrity and to constrain the way the components of the class can be used by other classes.

Abstraction. Taken individually, each of the preceding principles provides a manifestation of the more general principle of abstraction. An abstraction denotes the essential characteristics of a class that distinguishes it from all other kinds of classes and thus provides its essential behavior, and nothing more. Abstraction is the ability to group large quantities of information into a single chunk. Organizing a complex set of attributes and actions into a single class and then dealing with the module as a whole is a form of abstraction.

There is noticeable overlap in the definition of these principles as well as their applicability in OO design and coding. From a constructivist point of view, the overlap provides the transitional support needed to mentally *bridge* the principles to form the foundation for OOD. The ability to recognize and apply these fundamental OOD principles is the beginning phase and transitional process of what we call design readiness. We argue that effective designers have to be design ready in order to use sophisticated modeling approaches such as UML. Figure 1 presents a model of oriented design readiness, in relation to fundamental principles of object-oriented design. Each of these principles represents an attribute of design readiness. We are convinced that mastery of all these attributes leaves one in a state where the full power of OOD can be explored.

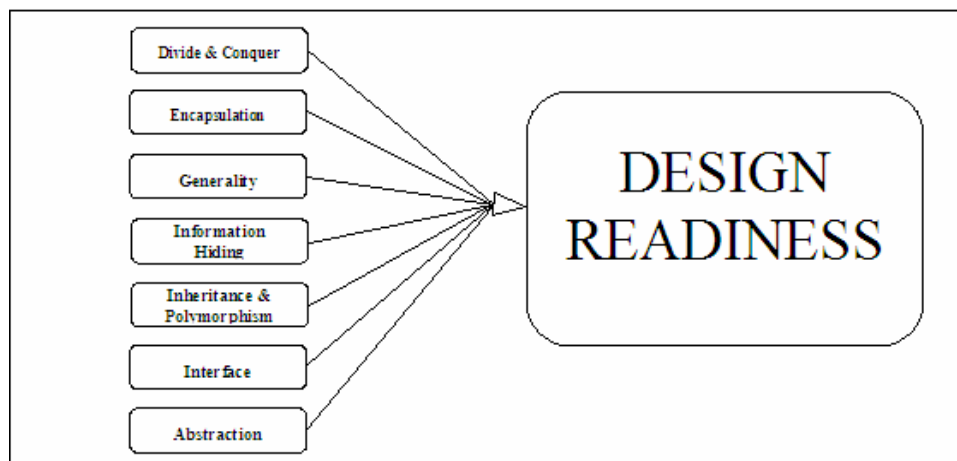


Figure 1 Model of Design Readiness

WHAT IS DESIGN READINESS

We have coined the term “design readiness” to refer to the point at which one is able to understand the concepts of OOD. We borrowed the concept of “readiness” from work on reading readiness. Simply stated, reading readiness is when a child is prepared to profit from beginning reading instruction, typically measured by his/her ability to decompose words into constituent sounds [18]. More specifically, reading readiness is “a transition extending over several months

during which time the child (student) gradually changes from a non-reader to a beginning reader. In this case, the readiness program couples the (student’s) past learning with new learning and brings the (student), gradually, through the transition.” [8] We created the concept of design readiness by analogy – an initial point and transitional process of coupling the student’s past learning with new learning, gradually increasing his/her level of design knowledge.

To effectively measure the validity of the fundamental principles of OOD, we developed the design readiness

assessment scale (DRAS). We believe that this scale will measure one's ability to apply these fundamental principles. Our goal is to validate the DRAS, so that we can then create instruction that coupled with students' past design experiences will bring them through the transition to become better OO designers.

THE DESIGN READINESS ASSESSMENT SCALE (DRAS)

We developed a twenty-eight question scale surveying the seven principles of OOD; allowing the DRAS to be divided in seven subscales of four questions each. The DRAS uses real-world problem scenarios to describe each of the principles. The use of real-world scenarios is a common testing format used for many of today's standardized tests, such as the ACT and SAT college entrance exam, as well as the GRE graduate school entrance exam. We found that the use of real-world scenarios to capture the essence of complex design problems was an enormous undertaking. We first

developed a language dictionary of common words and phrases that could map directly to the technical OOD terminology. The evaluation of the language dictionary was a separate study that will not be discussed in this paper. We did not create an exhaustive language dictionary; however we did produce an extensive list of common words and phrases for each of the OOD principles. Figure 2 gives a snap-shot of the language dictionary used to create the DRAS and how we mapped key words and phrases in an OOD real-world context scenario and questions. For example, to represent the divide and conquer principle, we used the phrase "things to do" to develop a scenario. The resulting question lists only one correct answer, at least two distracters, and one obvious outlier. Each question on the DRAS was created in the same fashion. Figure 2 demonstrates a straightforward example of divide and conquer. For a more detailed listing of the terminology dictionary and the DRAS questions, please send email to tracyL@radford.edu.

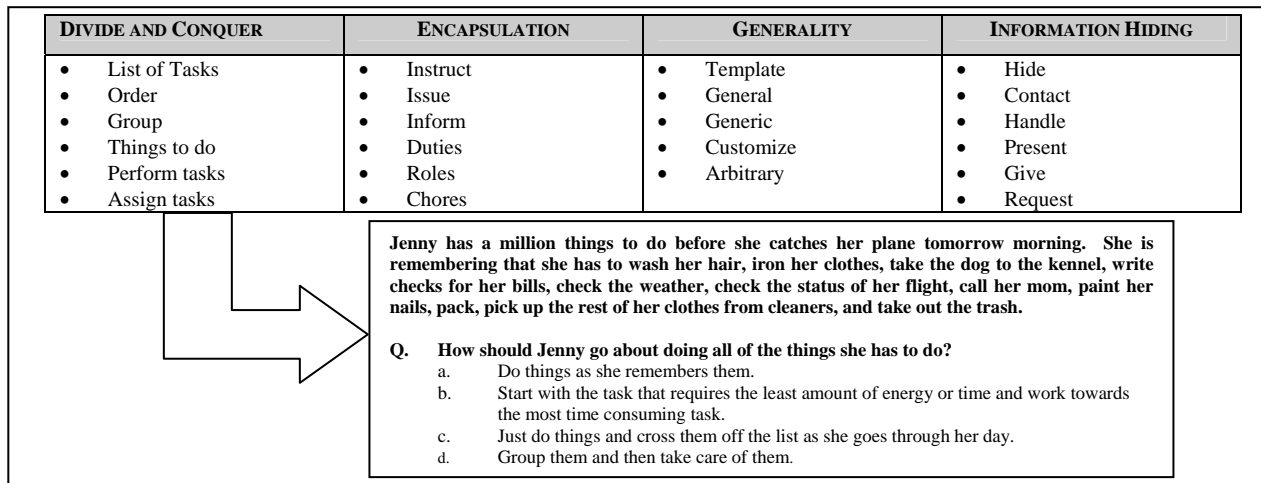


FIGURE 2
SAMPLE TERMINOLOGY DICTIONARY AND DESIGN READINESS ASSESSMENT SCALE QUESTION

DRAS ITEM RELIABILITY STUDY

We conducted a two day pilot study consisting of 28 computer science students and professionals from the Virginia Tech-Blacksburg, VA area. We had exactly half graduate professionals (experts) and half freshmen or sophomore computer science majors (novices). We hypothesized that the DRAS should reflect a consistency in high scores among experts and a varying range of lower scores for the novices. In addition, we wanted to eliminate unreliable questions for our set.

The instruction on the DRAS explicitly stated, and we verbally emphasized, that one should not read personal life experiences into the scenarios. Each participant was instructed to answer the questions based solely on the information provided in the accompanying scenario. They

were asked to complete an in-depth background survey, the DRAS, and a final self-efficacy survey. Participant completion times ranged from fifteen minutes to two hours.

RESULTS

We analyzed the data using Cronbach's Alpha inter-item reliability measure. After a stepwise elimination of redundant and unreliable questions, we were left with a two item DRAS scale for six of the OOD principles, with the reliability of .81 for novices and .67 for experts. During analysis of the data, we found that there was minimal inter-item reliability for the Interface OOD principle. We found significant overlap in terminology for Interface and several of the other OOD principles, which in turn meant that the questions developed for Interface would not fit within one subscale, but span across several subscales.

We provided an open dialogue question relevant to participant satisfaction of the overall assessment. Eighty-five percent of the participants expressed a concern about the length of the scale. In follow up studies, we found that two assessment items per OOD principles yielded the same or better results as did four items.

DISCUSSION

This research is the first step towards creating a standardized set of OOD principles. We are aware of the risks of trying to capture the understanding OOD principles in multiple choice questions. OOD is an art not a science, but even artists begin with fundamental strokes and techniques that will later turn into a masterpiece. We are simply trying to spur the academic and professional computer science industry to produce those simple strokes – core competency OOD principles. It is our ultimate goal to create a workforce of talented designers, equipped with the fundamental skills to effectively undertake any system design problem.

Since this study, we have further assessed the inter-item reliability of the DRAS and it is consistent at .80. We further developed the DRAS-RATIONALE, which gives the most popular answer to a design scenario and asks the question as to what reasoning best explains *why* the given answer is the most popular answer.

REFERENCES

- [1] Arnold, S. Bodoff, D. Coleman, H. Gilchrist, F. Hayes, *An Evaluation of Five Object-Oriented Development Methods*, Hewlett Packard Laboratories, Bristol, United Kingdom, Report No. HPL-91-52, June 1991.
- [2] Bagert, D. (1996). In Teaching the Object-Oriented Paradigm, Providing a Complete Picture is Essential. *Position paper for OOPSLA '96 workshop "Teaching and Learning Design in the First Academic Year"*, San Jose, CA.
- [3] Baudoin, Claude & Hollowell, Glenn. *Realizing the Object-Oriented Lifecycle*. Upper Saddle River, NJ: Prentice Hall, 1996.
- [4] Booch, G. (1994) *Object-Oriented Analysis and Design with Applications* Second Edition. Benjamin/Cummings, Redwood City, CA.
- [5] Buck, D. and Stucki, D. (1990) Design Early Considered Harmful: Graduated Exposure to Complexity and Structure Based on levels of Cognitive Development. *SIGCSE 2000*, Austin, TX, pp 75-79.
- [6] J. Cribbs, C. Roe, and S. Moon, *An Evaluation of Object-Oriented Analysis and Design Methodologies*, SIGS Books, New York, New York, 1992
- [7] Gamma, E., Helm, R., Johnson, R., and Vlissides, J.(1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, Reading, MA.
- [8] Harvard University. "What is Readiness?". An adaptation of Network readiness. <http://cyber.law.harvard.edu/readinessguide/readiness.html>
- [9] Fichman and C.F. Kemerer, *Object-Oriented and Conventional Analysis and Development Methodologies: Comparison and Critique*, 1991, Center for Information Systems Research, Sloan School of Management, M.I.T., CISR WP.
- [10] Fowler, M. (1997) *UML Distilled: Applying the Standard Object Modeling Language*. Upper Saddle, NJ: Addison-Wesley.
- [11] Goor, G. van den, Hong, S. and S. Brinkkemper, *A Comparison of Six Object-oriented Analysis and Design Methods*. Reportmethod Engineering Institute, University of Twente, the Netherlands, and Computer Information Systems Department, Georgia State University, Atlanta.
- [12] Horstmann, C. (2004). *Object-Oriented Design & Patterns*. San Jose, CA. Wiley & Sons.
- [13] Jacobson, I., Booch, G. and Rumbaugh, J. (1999). *The Unified Software Development Process*. MA: Addison-Wesley.
- [14] Jia, X. (2003). *Object-Oriented Software Development Using Java: Principles, Patterns, and Frameworks*. MA: Addison Wesley Longman, Inc.
- [15] Kafura, D. (1998). *Object-Oriented Software Design and Construction with C++*. Upper Saddle River, NJ: Prentice Hall.
- [16] Lee, R. and Tepfenhart W. (1997). *UML and C++: A Practical Guide to Object-Oriented Development*. Upper Saddle River, NJ: Prentice Hall.
- [17] D.E. Monarchi and G.I. Puhr, "A Research Typology for Object-Oriented Analysis and Design", *Communications of the ACM*, Vol. 35, No. 9, September 1992, pp. 35 - 47.
- [18] Matthews, D., Klaassens, A., Walter L. and Stewart T. (1999). LinguaLinks Library, Version 4.0, published on CD-ROM by SIL International, <http://www.sil.org/lingualinks/literacy/ReferenceMaterials/GlossaryOfLiteracyTerms/WhatIsReadingReadiness.htm#context>.
- [19] Morelli, R. (2002). *Object-Oriented Problem Solving*. Upper Saddle River, NJ: Prentice Hall.
- [20] Northrop, L.M. (1993). Finding an Educational Perspective for Object-Oriented Development. *Computer Science Education* 4(1), pg.5-12.
- [21] The Object Agency. A Comparison of Object-Oriented Development Methodologies. 1996. <http://www.toa.com/smmn?mcr.html#se>
- [22] Parnas D.L. (1972b), On the criteria to be used in decomposing systems into modules, *CACM* 15, pp. 1052-1058
- [23] Rumbaugh, James. (1996) *OMT Insights*. SIG Books.
- [24] Rappin, N. (1998). A framework for teaching learners to model by focusing complexity of modeling and simulation tools. Dissertation: Georgia Tech, College of Computing.
- [25] Rosson, M.B. and Alpert, S.R. (1990). The Cognitive Consequences of Object-Oriented Design. *Human Computer Interaction* 5. Lawrence Erlbaum Associates, pp.345-379. Shaw, M., Garlan, D. (1996). *Software Architecture Perspectives On An Emerging Discipline*. Upper Saddle River, NJ: Prentice Hall, Inc.
- [26] Rosson, M.B. and Carroll, J.M. (1997). Expertise and instruction in software development. In M. Helander & T.K. Landauer (Eds.) *Handbook of Human-Computer Interaction*, Second Edition. Amsterdam: North Holland, pp. 1105-1126. Sommerville, I. (2001). *Software Engineering*. Upper Saddle, NJ: Addison-Wesley.
- [27] Software Engineering Institute, Carnegie Mellon University, 2004, http://www.sei.cmu.edu/str/descriptions/oodesign_body.html.
- [28] Sommerville, I. (2001). *Software Engineering*. Upper Saddle, NJ: Addison-Wesley.
- [29] Wirfs-Brock, R., Wilkerson, B., and Wiener, L. (1990). *Designing Object-Oriented Software*. Englewood-cliffs, NJ: Prentice Hall.
- [30] Wirfs-Brock, R., McKean, A. (2003). *Object Design – Roles, Responsibilities, and Collaborators*. Boston, MA. Addison-Wesley.