Comparison between Traditional Approach and Object-Oriented Approach in Software Engineering Development

Nabil Mohammed Ali Munassar 1
PhD Student 3rd year of Computer Science & Engineering
Jawaharlal Nehru Technological University
Kuktapally, Hyderabad- 500 085, Andhra Pradesh, India

Dr. A. Govardhan 2
Professor of Computer Science & Engineering
Principal JNTUH of Engineering College, Jagityal,
Karimnagar (Dt), A.P., India

***Abstract*— This paper discusses the comparison between Traditional approaches and Object-Oriented approach.**

Traditional approach has a lot of models that deal with different types of projects such as waterfall, spiral, iterative and v-shaped, but all of them and other lack flexibility to deal with other kinds of projects like Object-Oriented.

Object–oriented Software Engineering (OOSE) is an object modeling language and methodology. The approach of using object – oriented techniques for designing a system is referred to as object–oriented design. Object–oriented development approaches are best suited to projects that will imply systems using emerging object technologies to construct, manage, and assemble those objects into useful computer applications. Object oriented design is the continuation of object-oriented analysis, continuing to center the development focus on object modeling techniques.

Keywordss- Software Engineering; Traditional Approach; Object-Oriented Approach; Analysis; Design; Deployment; Test; methodology; Comparison between Traditional Approach and Object-Oriented Approach.

I. INTRODUCTION

All software, especially large pieces of software produced by many people, should be produced using some kind of methodology. Even small pieces of software developed by one person can be improved by keeping a methodology in mind. A methodology is a systematic way of doing things. It is a repeatable process that we can follow from the earliest stages of software development through to the maintenance of an installed system. As well as the process, a methodology should specify what we're expected to produce as we follow the process. A methodology will also include recommendation or techniques for resource management, planning, scheduling and other management tasks. Good, widely available methodologies are essential for a mature software industry.

A good methodology will address at least the following issues: Planning, Scheduling, Resourcing, Workflows, Activities, Roles, Artifacts, Education. There are a number of phases common to every development, regardless of methodology, starting with requirements capture and ending with maintenance. During the last few decades a number of software development models have been proposed and discussed within the Software Engineering community. With the traditional approach, you're expected to move forward gracefully from one phase to the other. With the modern approach, on the other hand, you're allowed to perform each phase more than once and in any order [1, 10].

II. TRADITIONAL APPROACH

There are a number of phases common to every development, regardless of methodology, starting with requirements capture and ending with maintenance. With the traditional approach, will be expected to move forward gracefully from one phase to the other. The list below describes the common phases in software development [1, 6].

A. Requirements

Requirements capture is about discovering what is going to achieve with new piece of software and has two aspects. Business modeling involves understanding the context in which software will operate. A system requirement modeling (or functional specification) means deciding what capabilities the new software will have and writing down those capabilities [1].

B. Analysis

Analysis means understanding what are dealing with. Before designing a solution, it needs to be clear about the relevant entities, their properties and their inter-relationships. Also needs to be able to verify understanding. This can involve customers and end users, since they're likely to be subject-matter experts [1].

C. Design

In the design phase, will work out, how to solve the problem. In other words, make decisions based on experience, estimation and intuition, about what software which will write and how will deploy it. System design breaks the system down into logical subsystems (processes) and physical subsystems (computers and networks), decides how machines will communicate, and chooses the right technologies for the job, and so on [1].

D. Specification

Specification is an often-ignored, or at least often-neglected, phase. The term specification is used in different ways by different developers. For example, the output of the requirements phase is a specification of what the system must be able to do; the output of analysis is a specification of what are dealing with; and so on [3].

E. Implementation

In this phase is writing pieces of code that work together to form subsystems, which in turn collaborate to form the whole system. The sort of the task which is carried out during the implementation phase is 'Write the method bodies for the Inventory class, in such a way that they conform to their specification' [5].

F. Testing

When the software is complete, it must be tested against the system requirements to see if it fits the original goals. It is a good idea for programmers to perform small tests as they go along, to improve the quality of the code that they deliver [5].

G. Deployment

In the deployment phase, are concerned with getting the hardware and software to the end users, along with manuals and training materials. This may be a complex process, involving a gradual, planned transition from the old way of working to the new one [1].

H. Maintenance

When the system is deployed, it has only just been born. A long life stretches before it, during which it has to stand up to everyday use – this is where the real testing happens. The sort of the problem which is discovered discover during the maintenance phase is 'When the log-on window opens, it still contains the last password entered.' As the software developers, we normally interested in maintenance because of the faults (bugs) that are found in software. Must find the faults and remove them as quickly as possible, rolling out fixed versions of the software to keep the end users happy. As well as faults, users may discover deficiencies (things that the system should do but doesn't) and extra requirements (things that would improve the system) [3, 6].

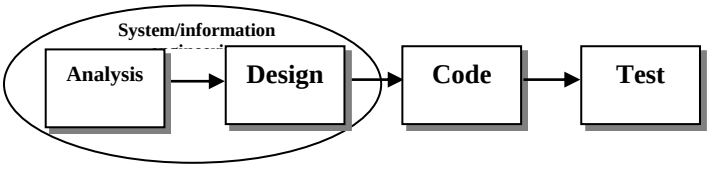

Figure 1: The linear Sequential Model [6].

III. Object-Oriented Approach

In object-oriented approach, a system is viewed as a set of objects. All object-orientation experts agree that a good methodology is essential for software development, especially when working in teams. Thus, quite a few methodologies have been invented over the last decade. Broadly speaking, all object-oriented methodologies are alike – they have similar phases and similar artifacts – but there are many small differences. Object-oriented methodologies tend not to be too prescriptive: the developers are given some choice about whether they use a particular type of diagram, for example. Therefore, the development team must select a methodology and agree which artifacts are to be produced, before they do any detailed planning or scheduling. In general, each methodology addresses:

- The philosophy behind each of the phases.
- The workflows and the individual activities within each phase.
- The artifacts that should be produced (diagrams, textual descriptions and code).
- Dependencies between the artifacts.
- Notations for the different kinds of artifacts.
- The need to model static structure and dynamic behavior.

Static modeling involves deciding what the logical or physical parts of the system should be and how they should be connected together. Dynamic modeling is about deciding how the static parts should collaborate. Roughly speaking, static modeling describes how we construct and initialize the system, while dynamic modeling describes how the system should behave when it's running. Typically, we produce at least one static model and one dynamic model during each phase of the development.

Some methodologies, especially the more comprehensive ones, have alternative development paths, geared to different types and sizes of development.[1,4]

The benefits of Object-Oriented Development are reduced time to market, greater product flexibility, and schedule predictability and the risks of them are performance and start-up costs [5].

A. Analysis

The aim of the analysis process is to analyze, specify, and define the system which is to be built. In this phase, we build models that will make it easier for us to understand the system. The models that are developed during analysis are oriented fully to the application and not the implementation environment; they are "essential" models that are independent of such things as operating system, programming language, DBMS, processor distribution, or hardware configuration.

Two different models are developed in analysis; the Requirements Model and the Analysis Model. These are based on requirement specifications and discussions with the prospective users. The first model, the Requirements Model, should make it possible to delimit the system and to define what functionality should take place within it. For this purpose we develop a conceptual picture of the system using problem domain objects and also specific interface descriptions of the system if it is meaningful for this system. We also describe the system as a number of use cases that are performed by a number of actors. The Analysis Model is an architectural model used for analysis of robustness. It gives a conceptual configuration of the system, consisting of various object classes: active controllers, domain entities, and interface objects. The purpose of this model is to find a robust and extensible structure for the system as a base for construction. Each of the object types has its own special purpose for this robustness, and together they will offer the total functionality that was specified in the Requirements Model. To manage the development, the Analysis Model may combine objects into Subsystems [2].

B. Construction

We build our system through construction based on the Analysis Model and the Requirements Model created by the analysis process. The construction process lasts until the coding is completed and the included units have been tested. There are three main reasons for a construction process:

1) *The Analysis Model is not sufficiently formal.*
2) *Adaptation must be made to the actual implementation environment.*
3) *We want to do internal validation of the analysis results.*

The construction activity produces two models, the Design Model and the Implementation Model. Construction is thus divided into two phases; design and implementation, each of which develops a model. The Design Model is a further refinement and formalization of the Analysis Model where consequences of the implementation environment have been taken into account. The Implementation model is the actual implementation (code) of the system. [2].

C. Testing

Testing is an activity to verify that a correct system is being built. Testing is traditionally an expensive activity, primarily because many faults are not detected until late in the development. To do effective testing we must have as a goal that every test should detect a fault.

Unit testing is performed to test a specific unit, where a unit can be of varying size from a class up to an entire subsystem. The unit is initially tested structurally, that is, "white box testing." This means that we use our knowledge of the inside of the unit to test it. We have various coverage criteria for the test, the minimum being to cover all statements. However, coverage criteria can be hard to define, due to polymorphism; many branches are made implicit in an object-oriented system. However, polymorphism also enhances the independence of each object, making them easier to test as standalone units. The use of inheritance also complicates testing, since we may need to retest operations at different levels in the inheritance hierarchy. On the other hand, since we typically have less code, there is less to test. Specification testing of a unit is done primarily from the object protocol (so-called "black box testing). Here we use equivalence partitioning to find appropriate test cases. Test planning must be done early, along with the identification and specification of tests [2].

D. UML

By the mid-1990s, the best-known methodologies were those invented by Ivar Jacobson, James Rumbaugh and Grady Booch. Each had his own consulting company using his own methodology and his own notation. By 1996, Jacobson and Rumbaugh had joined Rational Corporation, and they had developed a set of notations which became known as the Unified Modeling Language (UML). The ‘three amigos’, as they have become known, donated UML to the Object Management Group (OMG) for safekeeping and improvement. OMG is a not-for-profit industry consortium, founded in 1989 to promote open standards for enterprise-level object technology; their other well-known work is CORBA [1].

1) Use Case Diagram

A use case is a static description of some way in which a system or a business is used, by its customers, its users or by other systems. A use case diagram shows how system use cases are related to each other and how the users can get at them. Each bubble on a use case diagram represents a use case and each stick person represents a user. Figure 2 depicts a car rental store accessible over the Internet. From this picture, we can extract a lot of information quite easily. For example, an Assistant can make a reservation; a Customer can look for car models; Members can log on; users must be logged on before they can make reservations; and so on [1, 3].

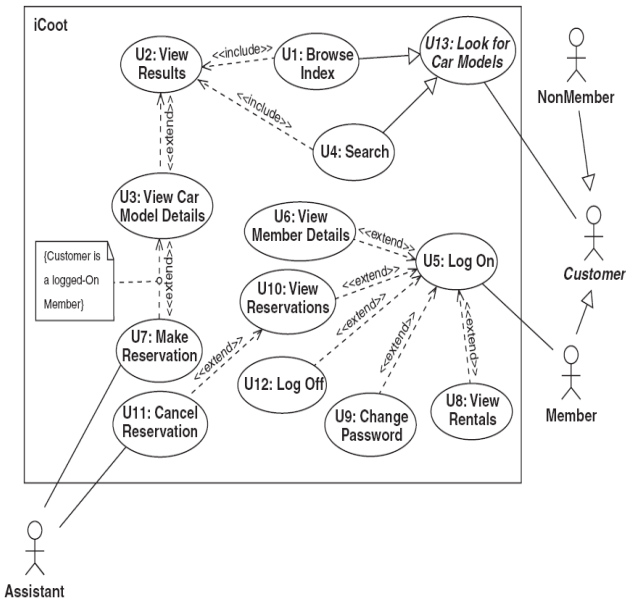

Figure 2: A use Case Diagram

2) Class Diagram (Analysis Level)

A class diagram shows which classes exist in the business (during analysis) or in the system itself (during subsystem design). Figure 3 shows an example of an analysis-level class diagram, with each class represented as a labeled box. As well as the classes themselves, a class diagram shows how objects of these classes can be connected together. For example, Figure 3 shows that a CarModel has inside it a CarModelDetails, referred to as its details.U3: View Car Model Details. (Extends U2, extended by U7.) Preconditions: None.

a) Customer selects one of the matching Car Models.
b) Customer requests details of the selected Car Model.
c) iCoot displays details of the selected Car Model (make, engine size, price, description, advert and poster).
d) If Customer is a logged-on Member, extend with U7.

Postconditions: iCoot has displayed details of selected Car Models.

Nonfunctional Requirements: r1. Adverts should be displayed using a streaming protocol rather than requiring a download [1, 5].

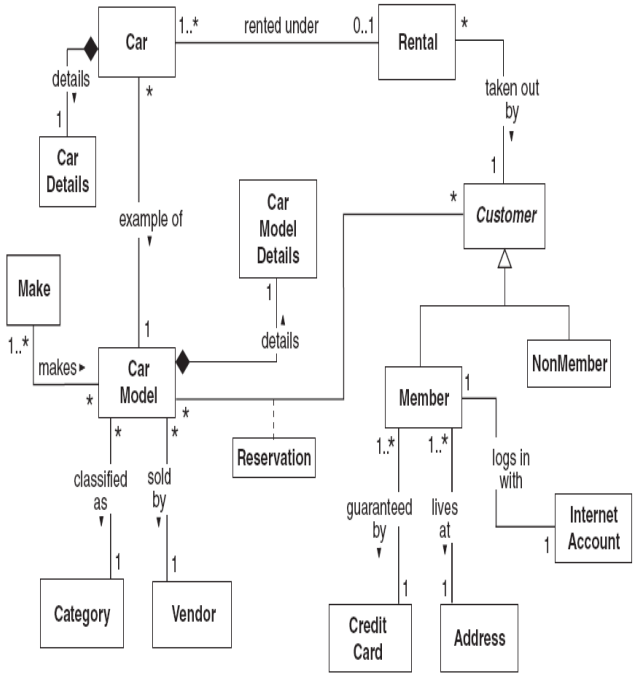

Figure 3: A class Diagram at the Analysis Level.

3) *Communication Diagram*

A communication diagram, as its name suggests, shows collaborations between objects. The one shown in Figure 4 describes the process of reserving a car model over the Internet: A Member tells the MemberUI to reserve a CarModel; the MemberUI tells the ReservationHome to create a Reservation for the given CarModel and the current Member; the MemberUI then asks the new Reservation for its number and returns this to the Member [1].

1) *Deployment Diagram*

A deployment diagram shows how the completed system will be deployed on one or more machines. A deployment diagram can include all sorts of features such as machines, processes, files and dependencies. Figure 5 shows that any number of HTMLClient nodes (each hosting a Web Browser) and GUIClient nodes communicate with two server machines, each hosting a WebServer and a CootBusinessServer; each Web Server communicates with a CootBusinessServer; and each CootBusinessServer communicates with a DBMS running on one of two DBServer nodes [1].

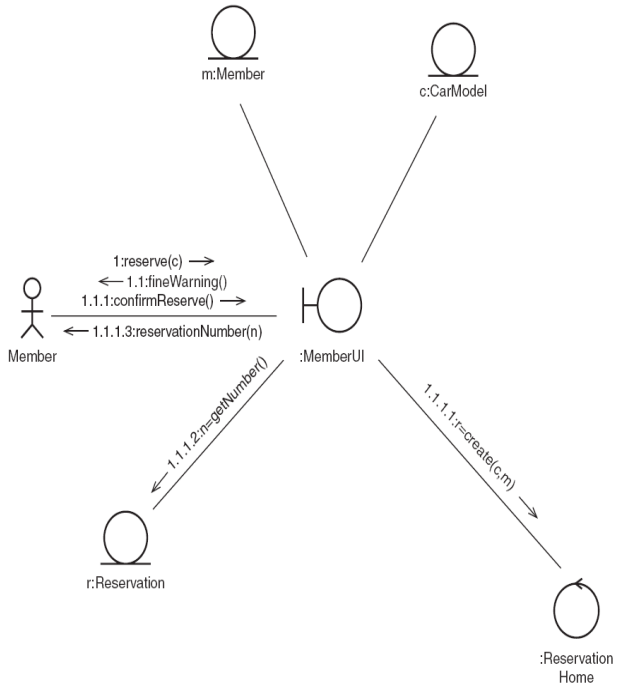

Figure 4: A communication Diagram

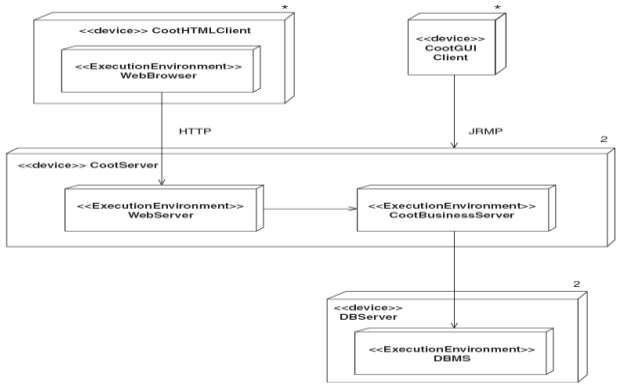

Figure 5: A deployment Diagram.

2) *Class Diagram (Design Level)*

The class diagram shown in Figure 6 uses the same notation as the one introduced in Figure 3. The only difference is that design-level class diagrams tend to use more of the available notation, because they are more detailed. This one expands on part of the analysis class diagram to show methods, constructors and navigability [1, 3].

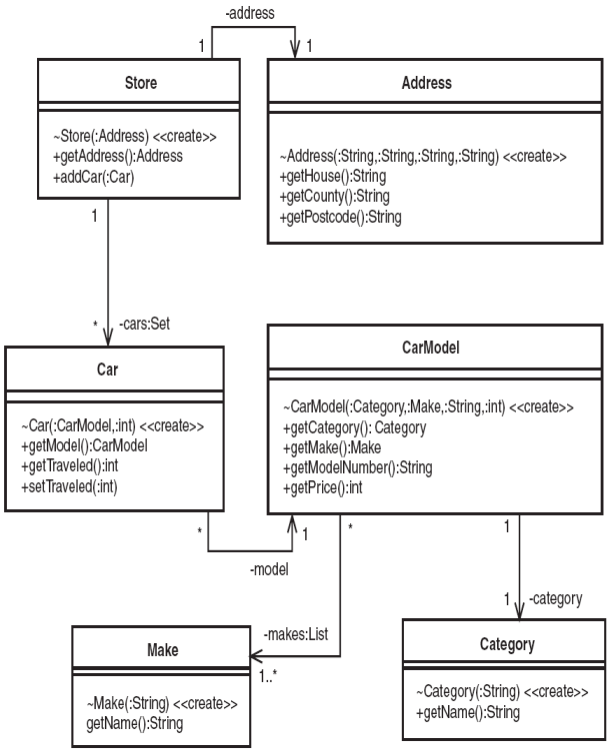

Figure 6: A design-level Class Diagram

3) Sequence Diagram

A sequence diagram shows interactions between objects. Communication diagrams also show interactions between objects, but in a way that emphasizes links rather than sequence. Sequence diagrams are used during subsystem design, but they are equally applicable to dynamic modeling during analysis, system design and even requirements capture. The diagram in Figure 7 specifies how a Member can log off from the system. Messages are shown as arrows flowing between vertical bars that represent objects (each object is named at the top of its bar). Time flows down the page on a sequence diagram. So, Figure 7 specifies, in brief: a Member asks the AuthenticationServlet to logoff; the AuthenticationServlet passes the request on to the AuthenticationServer, reading the id from the browser session; the AuthenticationServer finds the corresponding Member object and tells it to set its session id to 0; the Member passes this request on to its InternetAccount; finally, the Member is presented with the home page [1, 5].

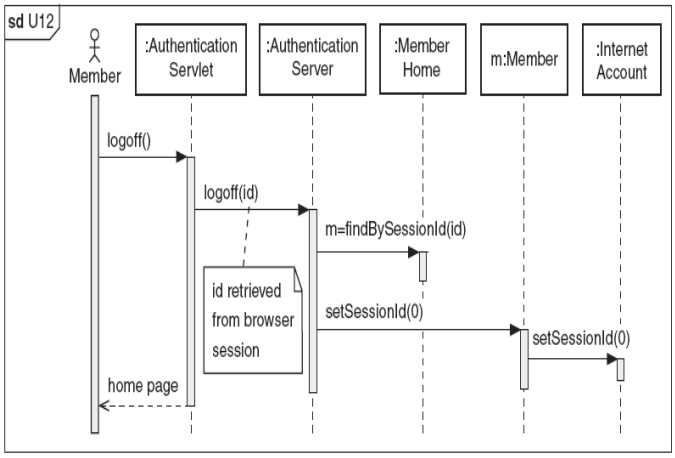

Figure 7: A sequence Diagram from the Design Phase

IV. Comparison between Traditonal Approach and Object-Oriented Approach to Development in Software Engineering

Summarize the comparison between Traditional Approach and Object-Oriented Approach shows through the table1.

TABLE 1. COMPARISON BETWEEN TRADITIONAL APPROACH AND OBJECT-ORIENTED APPROACH

TABLE I.

Traditional Approach	Object-Oriented Approach
Used to develop the Traditional Projects that uses procedural programming.	Used to develop Object-oriented Projects that depends on Object-Oriented programming.
Uses common processes likes: analysis, design, implementation, and testing.	Uses UML notations likes: use case, class diagram, communication diagram, development diagram and sequence diagram.
Depends on the size of projects and type of projects.	Depends on the experience of the team and complexity of projects through the numbers of objects.
Needs to large duration sometimes to development the large projects.	Need to more time than Traditional approach and leads that to more cost.
The problem of Traditional approach using classical life cycle [7, 8].	The object-oriented software life cycle identifies the three traditional activities of analysis, design, and implementation.[8].

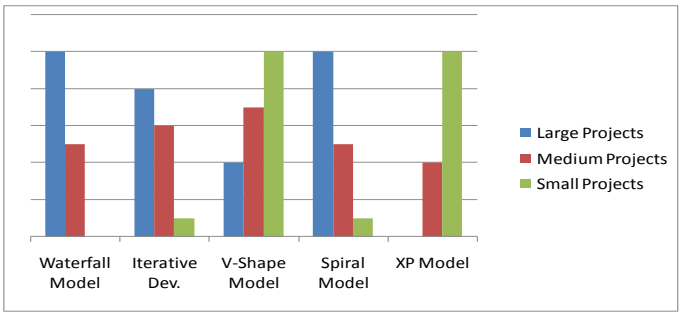

Figure 8: Illustrate The Different Models of Traditional Approach with Different Projects. [1, 6, 11]

From the previous figure 8 which illustrates the five models from traditional approach that deals with three types of projects, where we notice the waterfall model deals properly with large and medium projects like spiral model and iterative model that needs more time more cost and experience for team, however the V-shape model and XP model use properly with medium and small projects, because they need little time and some experience of team to perform projects.

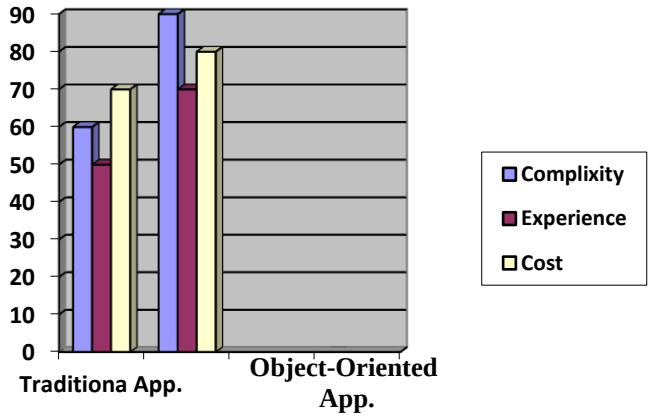

Figure 9: Illustrate The Different Criteria (Complexity, Experience and Cost) for Traditional Approach and Object-oriented Approach. [3, 5, 10]

From the previous chart illustrates the some criteria such as (Complexity, Experience, and Cost). In Traditional Approach this criterion depends on the type of model and size of project, but in general as shows from figure 9 is little above from the middle, however the Object-Oriented Approach depends on the complexity of project that leads to increase the cost than other approach.

V. Conclusion and Future Work

After completing this paper, it is concluded that:

1. As with any technology or tool invented by human beings, all SE methodologies have limitations [9].
2. The software engineering development has two ways to develop the projects that: traditional approach and object-oriented approach.
3. The traditional approach uses traditional projects that used in development of their procedural programming like C, this approach leads software developers to focus on Decomposition of larger algorithms into smaller ones.
4. The object-oriented approach uses to development the object-oriented projects that use the object-oriented programming like: C++ and Java.
5. The object-oriented approach to software development has a decided advantage over the traditional approach in dealing with complexity and the fact that most contemporary languages and tools are object-oriented.

Finally, some topics can be suggested for future works:

1. Design the model that includes the features of traditional approach and object-oriented approach to develop and deals with different projects in software engineering.
2. Updating some traditional approach to be able to use different types of projects.
3. Simplifying the object-oriented approach through its steps to use the smallest projects that deal with simple programming.

References

[1] Mike O'Docherty, "Object-Oriented Analysis and Design Understanding System Development with UML 2.0", John Wiley & Sons Ltd, England, 2005.

[2] Magnus Christerson and Larry L. Constantine, "Object-Oriented Software Engineering- A Use Case Driven Approach ", Objective Systems, Sweden, 2009.

[3] Ian sommerville, "Software Engineering", Addison Wesley, 7th edition, 2004.

[4] Pankaj Jalote, "An Integrated Approach to Software Engineering", Springer Science Business Media, Inc, Third Edition, 2005.

[5] Grady Booch, "Object-Oriented Analysis and Design with applications", Addison Wesley Longman, Inc, second Edition, 1998.

[6] Roger S. Pressman, "Software Engineering a practitioner's approach", McGraw-Hill, 5th edition, 2001.

[7] M M Lehman,"Process Models, Process Programs, Programming Support", ACM, 1987.

[8] Tim Korson and John D. McGregor," Understanding Object-Oriented: A Unifying Paradigm", ACM, Vol. 33, No. 9, 1990.

[9] Li Jiang and Armin Eberlein," Towards A Framework for Understanding the Relationships between Classical Software Engineering and Agile Methodologies", ACM, 2008.

[10] Luciano Rodrigues Guimarães and Dr. Plínio Roberto Souza Vilela," Comparing Software Development Models Using CDM", ACM, 2005.

[11] Alan M. Davis and Pradip Sitaram, "A Concurrent Process Model of Software Development", ACM, Software Engineering Notes Vol. 19 No. 2, 1994.

Authors Profile

Nabil Mohammed Ali Munassar

Was born in Jeddah, Saudi Arabia in 1978. He studied Computer Science at University of Science and Technology, Yemen from 1997 to 2001. In 2001 he received the Bachelor degree. He studied Master of Information Technology at Arab Academic, Yemen, from 2004 to 2007. Now he Ph.D. Student 3rd year of CSE at Jawaharlal Nehru Technological University (JNTU), Hyderabad, A. P., India. He is working as Associate Professor in Computer Science & Engineering College in University Of Science and Technology, Yemen. His areas of interest include Software Engineering, System Analysis and Design, Databases and Object Oriented Technologies.

Dr.A.Govardhan

Received Ph.D. degree in Computer Science and Engineering from Jawaharlal Nehru Technological University in 2003, M.Tech. from Jawaharlal Nehru University in 1994 and B.E. from Osmania University in 1992. He is working as a Principal of Jawaharlal Nehru Technological University, Jagitial. He has published around 108 papers in various national and international Journals/conferences. His research of interest includes Databases, Data Warehousing & Mining, Information Retrieval, Computer Networks, Image Processing, Software Engineering, Search Engines and Object Oriented Technologies.