Comparison study between Traditional and Object-Oriented Approaches to Develop all projects in Software Engineering

Nabil Mohammed Ali Munassar
PhD Scholar in Computer Science & Engineering
Jawaharlal Nehru Technological University Hyderabad
Kuktapally, Hyderabad- 500 085, Andhra Pradesh, India
E-mail: Nabil_monaser@hotmail.com

Dr. A. Govardhan
Professor of Computer Science - Engineering
& School of Information Technology
Jawaharlal Nehru Technological University Hyderabad
Kuktapally, Hyderabad- 500 085, Andhra Pradesh, India
E-mail: govardhan_cse@yahoo.co.in

Abstract— **Here in this paper we explore comparative study to analyze the performance differences between Traditional software development models and Object-Oriented approach. Traditional approaches like waterfall, spiral lack flexibility to deal with object oriented models. 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.**

Keywords-Software Engineering, Traditional Approach, Object-Oriented Approach, Analysis, Design, Deployment.

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 addresses 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].

The rest of the paper organized as fallow
Section II explored the procedural and object oriented software development models. Section III revealed the research methodology to conduct comparative study, in section IV the analysis of the comparative study carried out. Section V contains conclusion that fallowed by references.

II. SOFTWARE DEVELOPMENT MODELS

A. *Structural or Procedural software development models*

In structural software development models such as water fall, initially the project team analyses, then determining and prioritizing business requirements and needs. Next, in the design phase business requirements are translated into IT solutions, and a decision taken about which underlying technology i.e. COBOL, Java or Visual Basic, etc. etc. is to be used. Once processes are defined and online layouts built, code implementation takes place. The next stage of data conversion evolves into a fully tested solution for implementation and testing for evaluation by the end-user. The last and final stage involves evaluation and maintenance, with the latter ensuring everything runs smoothly.

The systems development life cycle (SDLC) can be defined as a process of understanding how an information system can support business needs or requirements of an organization, modeling the business processes, designing the systems components, and building the system. A systems development project thus goes through a sequence of four fundamental phases: planning, analysis, design, and implementation. Each of these phases also consists of a series of steps or activities that rely on some techniques to produce required deliverables. Even though all projects cycle through some common phases or activities, but how they are approached by the systems development group can be different – the project team might move through the phases and steps logically, consecutively, incrementally, or iteratively [5].

B. *Object Oriented software development models*

The object-oriented (OO) approach follows an iterative and incremental approach to systems development. The

978-1-4577-1531-0/11/$26.00 ©2011 IEEE

systems development life cycle is viewed as consisting of several increments or phases: inception, elaboration, construction, and transition [5]. In each increment or phase, the developers move through the activities of gathering requirements, analyzing the requirements, designing the system, implementing the design, and testing the system. See Figure 2. Thus the phases of the traditional systems development approach do not match with those of the OO life cycle; but in each increment, all phases of the traditional life cycle (requirements, analysis, design, implementation, testing) are visited iteratively until the developers are satisfied. However, there are times when one activity predominates over the other four – causing the systems development effort to move from the inception to elaboration, then elaboration to construction, and from construction to transition.

The object-oriented approach uses a set of diagramming techniques known as the Unified Modeling Language or UML [1]. It focuses on the three architectural views of a system: functional, static, and dynamic. The functional view describes the external behavior of the system from the perspective of the user. Use cases and use-case diagrams are used to depict the functional view. The static view is described in terms of attributes, methods, classes, relationships, and messages. Class-responsibility-collaboration (CRC) cards, class diagrams, and object diagrams are used to portray the static view. The dynamic view is represented by sequence diagrams, collaboration diagrams, and state charts. All diagrams are refined iteratively until the requirements of the information system are fully understood. Finally, the information system is developed through a combination of traditional relational database and object-oriented programming – rather than true object-oriented methodology for both programming and database.

III. Research methodology fallowed for Performance analysis of the traditional object oriented Software development process models

Software development is a highly complex field with countless variables impacting the system. All software systems are imperfect because they cannot be built with mathematical or physical certainty. Bridge building relies on physical and mathematical laws. Software development, however, has no laws or clear certainties on which to build. As a result, software is almost always flawed or sub-optimized. The author's study will provide the opinion of software developers on various aspects of developing a software development model such as: development time, Project complexity, Implementation challenges, Extensive and accurate documentation, Return on investment for minimal initial capital expense etc. Almost no software system is so simple that the development can be entirely scripted from beginning to end. The inherent uncertainty and complexity in all software projects requires an adaptive development plan to cope with uncertainty and a high number of unknown variables.

1 Overview

The basic objective of this section is concerned about: To study and analysis of different software development models which are being used by the software developers in software organizations. As there are a lot many models, which are being used like Code and fix model, Waterfall model, V-process model, Prototyping model, Spiral model and RAD model etc. To decide which model is more appropriate and suitable with respect to different software metrics, development time, complexity, implementation challenges, return-on-investment with minimal initial capital expenses, development cost etc. In order to find out the above mentioned software metrics research methodology was designed which is explained in the next section.

A software development process is a structure imposed on the development of a software product. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process. The most important aspect of a software development is its development, as it undergoes a number of development stages (software development life cycle) to reach to its final shape. The development steps, which need to be followed for developing software, project are- Project Planning, feasibility study, Requirement analysis, design etc. However the most important aspect of software development is the system design. There are several different approaches to software development, much like the various views of political parties toward governing a country. Some take a more structured, engineering-based approach to developing business solutions, whereas others may take a more incremental approach, where software evolves as it is developed piece-by-piece. Most methodologies share some combination of the following stages of software development: market research, analyzing the problem, implementation of the software, testing the software, deployment, maintenance and bug fixing [1].

2 Methodologies

In order to collect data from different organizations the author used two methods primary as well as secondary. The primary method will be collected through the close-end structured questionnaire. The secondary method will be collected by studying the documents from various organizations, established procedures, guidelines etc The First method selected to reach to goal will be questionnaires to measures the impact of lifecycles on the factors that influence the outcomes of software project. Questionnaire was designed on various aspects like software metrics, development time, complexity, implementation challenges, return-on-investment with minimal initial capital expenses, development cost etc.

As a complement of the comparative study analysis, interviews of project managers working in different companies were conducted to understand what type of methodologies are in use and what the most relevant aspects were found in developing software models.

3 Scalability of the information collection approach

Volunteers spent major part of their time in developing software (60%), and the next major part of time is spent as Technical Advisors in organizations.

Most of volunteers have more than ten years working experience, whereas while only small number of volunteers have experience below six months.

Most of the volunteer are from the commercial sector i.e. (65%) whereas volunteers in public sector, academic and research sector others are 20%, 10% and 5%. Most volunteers spent major part of their time in developing software; this suggests that the data collected through questioners can be accepted with confidence.

4 Findings

Table 1, shows the models and features opted to compare different models, which may influence the selection of lifecycle models. Each of the models has different response to these features as discussed in fallowing sections.

TABLE (1.A) MODELS OPTED FOR COMPARATIVE STUDY

Waterfall
Prototype
Spiral
Iterative
Object Oriented

TABLE (1.B) FEATURES OPTED FOR COMPARISON

Requirement Specifications
Understanding Requirements
Cost
Guarantee of Success
Resource Control
Cost Control
Simplicity
Risk Involvement
Expertise Required
Changes Incorporated
Risk Analysis
User Involvement
Overlapping Phases
Flexibility

5 Survey Questioner to verify the performance of models listed in table 1a.

a) Code and Fix

Identifying characteristics: Very limited or no specifications, Software developed in an ad-hoc manner

1. Q. How long have you been using this model?
2. Q. On average, how frequently do you use this model?
3. Q. Considering the projects or tasks for which you use this model, how long do they take to complete?

b) Waterfall, also known as: Classical model or variant such as 'V-model'

Identifying characteristics: Detailed documents produced during each 'phase', clearly defined phases are executed in turn with little or no overlap between phases, there is a single software release

1. Q. How long have you been using this model?
2. Q. On average, how frequently do you use this model?
3. Q. Considering the projects or tasks for which you use this model, how long do they take to complete?

c) Iterative, also known as 'Spiral'

Identifying characteristics: Same set of ordered 'phases' (an 'iteration') repeated multiple times, each iteration, addresses a set of identified risks, System released at the end of last iteration

1. Q. How long have you been using this model?
2. Q. On average, how frequently do you use this model?
3. Q. Considering the projects or tasks for which you use this model, how long do they take to complete?
   A week or less

d) Incremental, also known as 'Modular Development' or 'Staged Delivery'

Identifying characteristics: Software delivered in a set number of pre-planned releases, each release builds on the previous release, typically by adding a number of enhanced features

1. Q. How long have you been using this model?
2. Q. On average, how frequently do you use this model?
3. Q. Considering the projects or tasks for which you use this model, how long do they take to complete?

e) Prototyping, also known as 'Evolutionary Prototyping' or 'throw away Prototyping'

Identifying characteristics: 'Mock up' produced for evaluation purposes, 'Mock up' likely to be used to, determine requirements or to validate a specific design, 'Mock up' typically developed quickly

1. Q. How long have you been using this model?
2. Q. On average, how frequently do you use this model?
3. Q. Considering the projects or tasks for which you use this model, how long do they take to complete?

f) Transformational also known as 'Operational Specification' (for example 'Z') or '4th Generation Technique'

Identifying characteristics: Code is automatically created from a formal specification with no intermediate detailed design steps

1. Q. How long have you been using this model?
2. Q. On average, how frequently do you use this model?

3. Q. Considering the projects or tasks for which you use this model, how long do they take to complete?

6 Questionnaire to evaluate impact of software development process models on SDLC

The scenarios below are designed to understand why particular models are chosen in specific situations.

Each scenario has been designed to test how a specific single attribute influences lifecycle choice. (we acknowledge that real world projects are invariably a tradeoff between many different attributes).

The scenarios are applicable to any type of software task or project - ranging from say a simple bug fix through to a very large project. The examples have been chosen to be understandable rather than representative and should not be taken literally!

1. Q. A solution will take a considerable time to develop. For example, a multi-million lines-of-code solution or one that takes multi-person years to complete.
2. Q. A solution that is particularly complex, difficult or challenging to implement, for example, using a very complex or novel algorithm to solve a 'difficult' problem.
3. Q Where the solution has to be verified correct or exhibit a very low number of errors. For example, development of a solution in which human life would be in danger if the system should fail unexpectedly.
4. Q. A task in which the solution requirements are expected to change on a regular basis. For example, developing a solution for an 'emerging market' were the solution requirements are not initially known and may change as the market matures.
5. Q. A task for which the client requires extensive and accurate documentation. Typically, developing a solution for a client with a strict quality management system (such as ISO 9001)
6. Q. A task for which the only staff available are those with development experience - no other managerial or support staff are available. (The experience level of available developers is spread equally from novice to experienced professional). For example, this may be an internal development projects only
7. Q. A task for which many multi-skilled personnel are available, but all staff are considered novices in their area of expertise.
8. Q. The client requires an unusually fast return-on-investment for minimal initial capital expense. For example, a start-up company with limited capital investment.

7 Metrics used to evaluate Software development process models

- Shortfall is a measure of how far the operational system, at any time t, is from meeting the actual requirements at time t. This is the attribute most people are referring to when they ask 'does this system meet my needs?'
- Lateness is a measure of the time that elapses between the appearance of a new requirement and its satisfaction. Of course, recognizing that new requirements are not necessarily implemented in the order in which they appear, lateness actually measures the time delay associated with achievement of a level of functionality.
- Adaptability is the rate at which the software solution can adapt to new requirements, as measured as the slope of the solution curve.
- The longevity is the time a system solution is adaptable to change and remains viable, i.e. the time from system creation through the time it is replaced.
- Inappropriateness is the shared area between user needs and the solution curves. Thus captures the behavior of shortfall over time. The ultimately 'appropriate' model would exhibit a zero area meaning that new requirements are satisfied instantly.

IV. Comparative Study and Results Discussion

A. Summary of Lifecycle impact on Selected Cost Drivers

TABLE 2: Comparison report

1. Product Size

Waterfall	Documentation overhead suggests more suitable for larger projects.
incremental	Unknown although likely to be similar to Waterfall. Can degenerate into Code and Fix under some circumstances.
Prototyping	Less suitable for larger projects. May produce less code overall (compared to Waterfall), but design may be less coherent and harder to integrate.
Code & Fix	There is no formal communication between teams which suggests it will not scale well in large projects.

2. Software Problem Complexity

Waterfall	Complexity is tackled in an orderly and structured manner through separate well defined activities.
incremental	Unknown although likely to be similar to Waterfall. Increased emphasis on extensible architecture .
Prototyping	Approach allows prototype solutions can be evaluated and the specification altered. Increased chance of incoherent designs (compared to Waterfall) and undocumented 'compromises' being made.
Code & Fix	Difficult to assess, although there is a lack design and limited testing.

3. Required Quality

Waterfall	Highly structured approach, while phase documentation verification reduces chances of ‘downstream’ errors (where errors are much more

	costly to correct).
incremental	Similar to Waterfall, although can be hard to incrementally add new functionality without de-stabilizing existing functionality.
Prototyping	Undocumented compromises may affect quality however release QA activities should detect this.
Code & Fix	Problems discovered late due to lack of specification, design and limited testing leading to higher costs and chances of regression fault.

4. Requirements Volatility

Waterfall	Process discourages 'upstream' changes since they become exponentially more expensive (due to rework) as project progresses.
incremental	Process can start without complete requirements (not ideal), but requirements should remain relatively static within a stage (to avoid potential for ad-hoc changes).
Prototyping	Can be used to resolve ambiguous or unknown requirements or specifications. Improves quality and accuracy of specification reducing cost of downstream changes.
Code & Fix	Requirements can be changed as required since there are none! Ultimately, may not match needs as there is no specification

5. Amount of documentation

Waterfall	Detailed documentation is primary method for knowledge transfer between phases. All documentation subject to validation and verification activities.
incremental	Unspecified - depends on project, although likely to be similar to Waterfall.
Prototyping	Likely to produce less documentation (although documentation process can be added if required).
Code & Fix	No documentation required - left entirely up to developer.

6. Experience of Personnel

Waterfall	Can be used with weak or inexperienced staff since ordered structure helps to minimize wasted effort.
incremental	Similar to Waterfall, but requires skilled multi-disciplined staff to manage stage release inter-dependencies.
Prototyping	Objective and scope setting with prototype design, implementation and evolution through feedback suggest experienced personnel are required.
Code & Fix	Anyone can use it - no management experience required. 100% of effort focused on coding (and fixing).

7. Availability of Personnel for Project

Waterfall	Project can be planned initially allowing multi-skilled / experienced staff to be used only as required - documentation level helps knowledge transfer and training.
Incremental	Experienced management and developers required to schedule and resolve dependencies between increments.
Prototyping	May require skilled management to determine prototype objectives or scope. Developer must be able to interpret and respond to customer feedback appropriately.
Code & Fix	Only developer experience required - no management experience.

8. Project Duration

Waterfall	No usable software until the very end of the project - although documentation may have some tangible value.
Incremental	Incremental delivery, allowing staged payment and providing faster return on investment.
Prototyping	Users see progress in form of prototype very early. However, customer may object to paying for a redevelopment once they see a 'working' prototype.
Code & Fix	No means of assessing progress, identifying risk, or measuring quality. Software may be delivered or not.

9. Lifecycle Usage based on Task Duration

Process Model	less than a week	Over a week to Month	Over a month to 3 months	Over 3 months to Year	Over an Year
Code & Fix	48%	17%	21%	7%	7%
Waterfall	6%	6%	9%	46%	33%
Incremental	0%	11%	14%	50%	25%
Prototyping	0%	14%	34%	33%	19%

B. Comparison report and discussion on features impact:

Feature	Waterfall Model	Prototype Model	Spiral Model	Iterative Model	Object Oriented Model (Combination of incremental and iterative)
Requirement Specifications	Beginning	Frequently Changed	Beginning	Beginning	Frequently Changed
Understanding Requirements	Well Understood	Not Well understood	Well Understood	Not Well understood	Well understood
Cost	Low	High	Intermediate	Low	Peak
Guarantee of Success	Low	Good	High	High	Peak
Resource Control	Yes	No	Yes	Yes	No
Cost Control	Yes	No	Yes	No	Sure
Simplicity	Simple	Simple	Intermediate	Intermediate	Complex
Risk Involvement	High	High	Low	Intermediate	Dependent of requirement changes

Expertise Required	High	Medium	High	High	Obvious in all phases
Changes Incorporated	Difficult	Easy	Easy	Easy	depends on resource experts availability
Risk Analysis	Only at beginning	No Risk Analysis	Yes	No	Modular approach
User Involvement	Only at beginning	High	High	Intermediate	Dependent of requirement changes
Overlapping Phases	No	Yes	Yes	No	Dependant of requirement changes
Flexibility	Rigid	Highly Flexible	Flexible	Less Flexible	Peak

1. Requirement Specification

Requirement specifications are needed just at the beginning of the Wa11erfall model, Spiral model and Iterative model, however for Prototype model and Object oriented model process the requirement specifications are frequently changed during the development process [2].

2. Understanding Requirements

Waterfall model, spiral model and Object oriented model needs well understanding of the requirements, while prototype model and Iterative model do not need good understanding of the requirements [3].

3. Cost

Data was obtained for a cost driver value of 'very high quality' (expressed as' very low number of errors' or 'errors threatens human life'). The data shows that Waterfall and Iterative models are used for projects, which have low cost requirements for software development, where as for projects with intermediate cost Spiral model is suitable, while the Prototyping model is suitable for projects with more cost than waterfall model and Spiral model. While the Object oriented process model leads to very high cost [4].

4. Guarantee of Success

As per the research work if we use waterfall model for software projects the guarantee of success is very low, but on the other hand if we use Prototype model the guarantee of success is good, but again spiral model and Iterative model have intermediate guarantee of success between good and high. However out of all the five models under taken for this study, the model have very high guarantee of success.

5. Resources Control

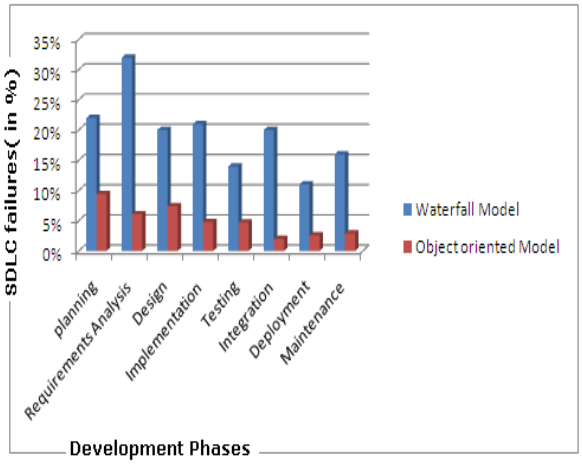
From the research work, it is concluded that prototype model and model do not have their control over resources, but on the contrary waterfall model, spiral model and iterative models have control over resources.

6. Cost Control

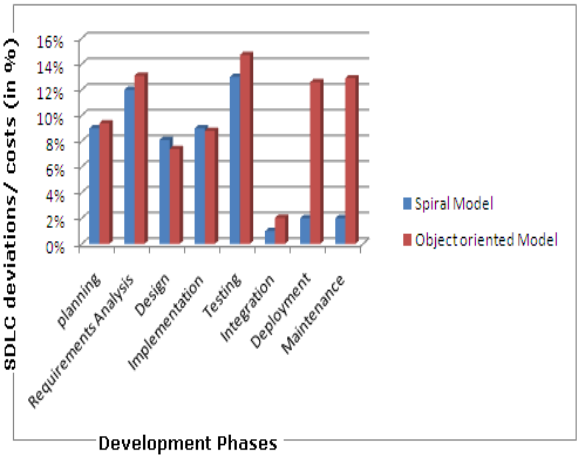
Data was obtained for a cost driver value of 'range of development experience'. The Prototyping model and Iterative model are only models, which don't have their cost control features, which make them inappropriate. The data values for the Waterfall, spiral and models are also supported by this study, because they have cost control feature, which make them best compared to others, as cost control factor is important for all software projects.

C. Comparison of percentage of failures occurred in different phases of the development models

We can observe in Graph 1 that traditional software development model such as waterfall model is not feasible on large projects with frequent requirement specification changes



Graph1: The comparison report: Failures of waterfall and Object Oriented Models on large project with frequent changes requirement specification



Graph 2: deviations / cost effecting development stages of structural model (spiral) and object oriented model on small size project with no changes in requirement specification during development process.

In graph 2: we can observe that the traditional models called spiral and object oriented model committing equal efficiency in majority of the development phases as in the farm of integration cost, deployment cost and maintenance cost the Object

oriented model is too high when compared to spiral model. Hence we can argue that the traditional software development models such as waterfall, spiral, incremental models are feasible to develop small size projects with stable requirement specifications.

V. Conclusion

After completing this paper, it is concluded that as with any technology or tool invented by human beings, all SE methodologies have limitations [9], The software engineering development has two ways to develop the projects that: structural or procedural approach and object-oriented approach, the structural or procedural models are feasible in terms of integration, deployment and maintenance but limited to small projects with stable requirement specification. These structural or procedural approaches lead software developers to focus on Decomposition of larger algorithms into smaller ones.

The object-oriented approach to software development has a decided advantage over the traditional approach in dealing with complexity, which is common preference of large projects with frequent changes in requirement specification and the fact that most contemporary languages and tools are object-oriented. In future it is recommended to conduct research to design cross models of the existing that includes the features of traditional approach and object-oriented approach and finding updates for some traditional approaches to improve their robustness and intensifying the object-oriented approach to support the development of small scale projects under object oriented model.

References

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

[2] Molokken-Ostvold et.al, "A comparison of software project overruns - flexible versus sequential development models", Volume 31, Issue 9, Page(s): 754 – 766, IEEE CNF, Sept. 2005.

[3] Boehm, B. W. "A spiral model of software development and enhancement", ISSN: 0018-9162, Volume: 21, Issue: 5, on page(s): 61-72, May 1988.

[4] Abrahamsson P. et.al, "Agile Software Development Methods: Review and Analysis", ESPOO, VTT Publications 478, VTT Technical Research Centre of Finland. http:/www. fi/pdf/publications/2002/P478.pdf, 2002.

[5] Dennis, A., Wixom, B. H. and Tegarden, D., "Systems Analysis and Design: An Object-Oriented Approach", John Wiley & sons, New York, 2002.

[6] Roger S. Pressman, "Software Engineering a practitioner's approach", McGraw-Hill, 5th edition, 2006.

[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