The Comparative Research on Various Software Development Process Model

Mr. Preyash Dholakia[1], Mr. Dishek Mankad[2]

[1] M.C.A. K.S.K.V. Kachchh University MCA College
[2] M.C.A., B.R.Patel Institute of Computer Application [MCA Program]

Abstract- In the present scenario all software systems are because they cannot be built with mathematical or physical certainty, Hence in this research paper the comparison of various software development models has been carried out. According SDLC e ach and e very model have the advantage and disadvantage so in this research we have to calculate the performance of each model on behalf of some important features. The concept of system lifecycle models came into existence that emphasized on the need to follow some structured approach towards building new or improved system.

Index Terms- Software Engineering, Model, SDLC, Software Product, software Development Process

I. INTRODUCTION

A software development process, also known as a software development life cycle (SDLC),is a structure imposed on the development of a software product. It is often considered as a subset of system development life cycle. There are several models for such processes, each describing app roaches to a variety o f activities that take place during the process. Software Engineering (SE) is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering g to software because e it integrates significant mathematics, computer science and practices whose origins are in Engineering. Various processes and methodologies have been developed over the last few decades to imp rove software quality , with varying degrees of success .However, it is widely agreed that no single approach that will prevent project overruns and failures in all cases. Software projects that are large, complicated , poorly-specified, and involve unfamiliar aspects , are still particularly vulnerable to large, unanticipated problems .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 . It aims to be the standard that defines all the tasks required for developing and maintaining software. Software Engineering processes are composed of many activities, notably the following :

- ✓ Requirement Analysis
- ✓ Specification
- ✓ Software architecture
- ✓ Implementation
- ✓ Testing, Documentation
- ✓ Training and Support
- ✓ Maintenance

Software development teams, taking into account its goals and the scale of a particular project, and have a number of well-established software development models to choose from. Therefore, even though there are number of models each software Development Company adopts the best-suited model, which facilitates the software development process and boosts the productivity of its team members .

II. RESEARCH ELABORATIONS

A Programming process model is an abstract representation to describe the process from a particular perspective. There are numbers of general models for software processes, like: Waterfall model, Evolutionary development, Formal systems development and Reuse-based development, etc. This research will view the following five models:

1. Waterfall model.
2. Iteration model.
3. V-shaped model.
4. Spiral model.
5. Extreme model.

These models are chosen because their features correspond to most software development programs.

- ➢ **The Waterfall Model**

The waterfall model is the classical model of software engineering. This model is one of the oldest models and is widely used in government projects and in many major companies. As this model emphasizes planning in early stages, it ensures design flaws before they develop. In addition, its intensive document and planning make it work well for projects in which quality control is a major model:

1 System requirements: Establishes the components for building the system, including the hardware requirements, software tools, and other necessary components. Examples include decisions on hardware, such as plug-in boards (number of channels, acquisition speed, and so on), and decisions external pieces of software, such as databases or libraries.

2 Software requirements: Establishes the expectations for software functionality and identifies which system requirements the software affects. Requirements analysis includes determining interaction needed with other applications and

databases, performance requirements, user interface requirements, and so on.

3 Architectural design: Determines the software framework of a system to meet the specific requirements. This design defines the major components and the interaction of those components, but it does not define the structure of each component. The external interfaces and tools used in the project can be determined by the designer.

4 Detailed design: Examines the software components defined in the architectural design stage and produces a specification for how each component is implemented.

5 Coding: Implements the detailed design specification.

6 Testing: Determines whether the software meets the specified requirements and finds any errors present in the code.

7 Maintenance: Addresses problems and enhancement requests after the software releases.

In some organizations, a change control board maintains the quality of the product by reviewing each change made in the maintenance stage. Consider applying the full waterfall development cycle model when correcting problems or implementing these enhancement requests.

In each stage, documents that explain the objectives and describe the requirements for that phase are created. At the end of each stage, a review to determine whether the project can proceed to the next stage is held. Your prototyping can also be incorporated into any stage from the architectural design and after.

Many people believe that this model cannot be applied to all situations. For example, with the pure waterfall model, the requirements must be stated before beginning the design, and the complete design must be stated before starting coding. There is no overlap between stages. In real-world development, however, one can discover issues during the design or coding stages that point out errors or gaps in the requirements.

The waterfall method does not prohibit returning to an earlier phase, for example, returning from the design phase to the requirements phase. However, this involves costly extensive documentation development. Thus, oversights made in the requirements phase are expensive to correct later.

Because the actual development comes late in the process one does not see results for a long time. This delay can be disconcerting to management and customers. Many people also think that the amount of documentation is excessive and inflexible.

Advantages :

- Easy to understand and implement.
- Widely used and known (in theory!).
- 3. Reinforces good habits: define-before- design, design-before-code.
- Identifies deliverables and milestones.
- Document driven, URD, SRD, … etc. Published documentation standards, e.g. PSS-05.
- Works well on mature products and weak teams.

Disadvantages :

- Idealized, doesn't match reality well.
- Doesn't reflect iterative nature of exploratory early in project.
- Software is delivered late in project, delays discovery
- 4. Difficult and expensive to make changes to documents, "swimming upstream".

Pure Waterfall

This is the classical system development model. It consists of discontinuous phases:

- ✓ Concept.
- ✓ Requirements.
- ✓ Architectural design.
- ✓ Detailed design.
- ✓ Coding and development.
- ✓ Testing and implementation.

Table 1: Strengths & Weaknesses of Pure Waterfall

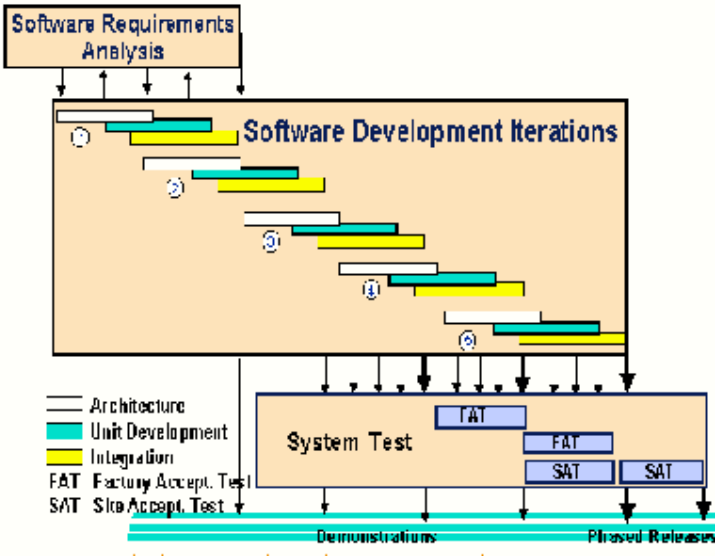
Strengths	Weaknesses
Minimizes planning overhead since it can be done up front.	Only the final phase produces a non-documentation deliverable.
Structure minimizes wasted effort, so it works well for technically weak or inexperienced staff.	Backing up to address mistakes is difficult.

- **Pure Waterfall Summary**

The pure waterfall model performs well for products with clearly understood requirements or when working with well understood technical tools, architectures and infrastructures. Its weaknesses frequently make it inadvisable when rapid development is needed. In those cases, modified models may be more effective.

Iterative Development

The problems with the Waterfall Model created a demand for a new method of developing systems which could provide faster results, require less up-front information, and offer greater flexibility. With Iterative Development, the project is divided into small parts. This allows the development team to demonstrate results earlier on in the process and obtain valuable feedback from system users. Often, each iteration is actually a mini-Waterfall process with the feedback from one phase providing vital information for the design of the next phase. In a variation of this model, the software products, which are produced at the end of each step (or series of steps), can go into production immediately as incremental releases.

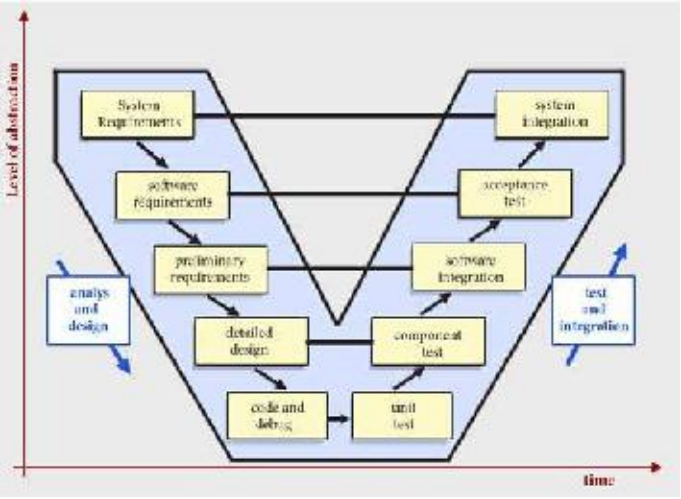


➢ **V-Shaped Model**

Just like the waterfall model, the V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins. Testing is emphasized in this model more than the waterfall model. The testing procedures are developed early in the life cycle before any coding is done, during each of the phases preceding implementation. Requirements begin the life cycle model just like the waterfall model. Before development is started, a system test plan is created. The test plan focuses on meeting the functionality specified in requirements gathering.

The high-level design phase focuses on system architecture and design. An integration test plan is created in this phase in order to test the pieces of the software systems ability to work together. However, the low-level design phase lies where the actual software components are designed, and unit tests are created in this phase as well.

The implementation phase is, again, where all coding takes place. Once coding is complete, the path of execution continues up the right side of the V where the test plans developed earlier are now put to use.



Advantages

- Simple and easy to use.
- Each phase has specific deliverables.
- Higher chance of success over the waterfall model due to the early development of test plans during the life cycle.

Disadvantages

- Very rigid like the waterfall model.
- Little flexibility and adjusting scope is difficult and expensive.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.
- This Model does not provide a clear path for problems found during testing phases [7].

➢ **Spiral Model**

The spiral model is similar to the incremental model, with more emphases placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spiral, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spiral builds on the baseline spiral. Requirements are gathered during the planning phase. In the risk analysis phase, a process is undertaken to identify risk and alternate solutions. A prototype is produced at the end of the risk analysis phase. Software is

produced in the engineering phase, along with testing at the end of the phase. The evaluation phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

Advantages

1. High amount of risk analysis.
2. Good for large and mission-critical projects.
3. Software is produced early in the software life cycle.

Disadvantages

1. Can be a costly model to use.
2. Risk analysis requires highly specific expertise.
3. Project's success is highly dependent on the risk
4. Doesn't work well for smaller projects .

Spiral model sectors

1. Objective setting :Specific objectives for the phase are identified.
2. Risk assessment and reduction: Risks are assessed and
3. Development and validation: A development model for the system is chosen which can be any of the
4. Planning: The project is reviewed and the next phase of the spiral is planned

WinWin Spiral Model

The original spiral model [Boehm 88] began each cycle of the spiral by performing the next level of elaboration of the prospective system's objectives, constraints and alternatives. A primary difficulty in

applying the spiral model has been the lack of explicit process guidance in determining these objectives, constraints, and alternatives. The Win-Win Spiral Model [Boehm 94] uses the theory W (win-win) approach [Boehm 89b] to converge on a system's next-level objectives, constraints, and alternatives. This Theory W approach involves identifying the system's stakeholders and their win conditions, and using negotiation processes to determine a mutually satisfactory set of objectives, constraints, and alternatives for the stakeholders. In particular, as illustrated in the figure, the nine-step Theory W process translates into the following spiral model extensions:

1. Determine Objectives: Identify the system life-cycle stakeholders and their win conditions and establish initial system boundaries and external interfaces.
2. Determine Constraints: Determine the conditions under which the system would produce win-lose or lose-lose outcomes for some stakeholders.

3. Identify and Evaluate Alternatives: Solicit suggestions from stakeholders, evaluate them with respect to stakeholders' win conditions, synthesize and negotiate candidate win-win alternatives, analyze, assess, resolve
win-lose or lose-lose risks, record commitments and areasto be left flexible in the project's design record and life cycle plans.
4. Cycle through the Spiral: Elaborate the win conditions evaluate and screen alternatives, resolve risks, accumulate appropriate commitments, and develop and execute downstream plans

- **Extreme Programming**

An approach to development, based on the development and delivery of very small increments f functionality. It relies on constant code improvement, user involvement in the development team and pair wise programming . It can be difficult to keep the interest of customers who are involved in the process. Team members may be unsuited to the intense involvement that characterizes agile methods. Prioritizing changes can be difficult where there are multiple stakeholders. Maintaining simplicity requires extra work. Contracts may be a problem as with other approaches to iterative development.

- **Extreme Programming Practices**

Increment Planning: Requirements are recorded on Story Cards and the Stories to be included in a release are Incremental planning: Requirements are recorded on determined by the time available and their relative priority. The developers break these stories into development "Tasks".

Small Releases: The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.

Simple Design: Enough design is carried out to meet the current requirements and no more.

Test first development: An automated unit test framework is used to write tests for a new piece of functionality before functionality itself is implemented.

Refactoring: All developers are expected to re-factor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable. \

Pair Programming: Developers work in pairs, checking each other's work and providing support to do a good job.

Collective Ownership: The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers own all the code. Anyone can change anything.

Continuous Integration: As soon as work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.

Sustainable pace: Large amounts of over-time are not considered acceptable as the net effect is often to reduce code quality and medium term productivity.

On-site Customer: A representative of the end-user of the system (the Customer) should be available full time for the use of the XP team. In an extreme programming process the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

- **XP and agile principles**
  - ✓ Incremental development is supported through small, frequent system releases.
  - ✓ Customer involvement means full-time customer engagement with the team.
  - ✓ People not process through pair programming, collective ownership and a process that avoids long working hours.
  - ✓ Change supported through regular system releases.
  - ✓ Maintaining simplicity through constant refactoring of code.

Advantages

- Lightweight methods suit small-medium size projects.
- Produces good team cohesion.
- Emphasizes final product.
- Iterative.
- Test based approach to requirements and quality assurance.

Disadvantages

- Difficult to scale up to large projects where documentation is essential.
- Needs experience and skill if not to degenerate into
- code-and-fix.
- Programming pairs is costly.

II RESEARCH RESULT

Modified Waterfall

The modified waterfall uses the same phases as the pure waterfall, but is not based on a discontinuous basis. This enables the phases to overlap when needed. The pure waterfall can also split into subprojects at an appropriate phase (such as after the architectural design or detailed design).

Table 2: Strengths & Weaknesses of Modified Waterfall

Strengths	Weaknesses
More flexible than the pure waterfall model.	Milestones are more ambiguous than the pure waterfall.
If there is personnel continuity between the phases, documentation can be substantially reduced	Activities performed in parallel are subject to miscommunication and mistaken assumptions.

Modified Waterfall Summary

Risk reduction spirals can be added to the top of the waterfall to reduce risks prior to the waterfall phases. The waterfall can be further modified using options such as prototyping, JADs or CRC sessions or other methods of requirements gathering done in overlapping phases .

ACKNOWLEDGMENT

The success of our research is never limited to the individual undertaking the paper. It is the cooperative effort of the people around an individual that spell success. For all efforts, behind this successful research, we are highly intended to all those personalities without whom this research would ever be completed. We find no words to express our gratitude towards those who were constantly involved with us throughout our work.

REFERENCES

[1] Inmon, W.H., *Building the Data Warehouse*. John Wiley, 1992.
[2] http://www.olapcouncil.org
[3] Codd, E.F., S.B. Codd, C.T. Salley, "Providing OLAP (On-Line Analytical Processing) to User Analyst: An IT Mandate." Available from Arbor Software's web site http://www.arborsoft.com/OLAP.html.
[4] http://pwp.starnetinc.com/larryg/articles.html
[5] Kimball, R. *The Data Warehouse Toolkit*. John Wiley, 1996.
[6] Barclay, T., R. Barnes, J. Gray, P. Sundaresan, "Loading Databases using Dataflow Parallelism." *SIGMOD Record*, Vol. 23, No. 4, Dec.1994.
[7] Blakeley, J.A., N. Coburn, P. Larson. "Updating Derived Relations: Detecting Irrelevant and Autonomously Computable Updates." *ACM TODS*, Vol.4, No. 3, 1989.
[8]Gupta, A., I.S. Mumick, "Maintenance of Materialized Views: Problems, Techniques, and Applications." *Data Eng. Bulletin*, Vol. 18, No. 2, June 1995. 9 Zhuge, Y., H. Garcia-Molina, J. Hammer, J. Widom, "View Maintenance in a Warehousing Environment, *Proc. Of SIGMOD Conf.*, 1995.
[10] Roussopoulos, N., et al., "The Maryland ADMS Project: Views R Us*." Data Eng. Bulletin*, Vol. 18, No.2, June 1995.
[11] O'Neil P., Quass D. "Improved Query Performance with Variant Indices", To appear in *Proc. of SIGMOD Conf.*, 1997.
[12] O'Neil P., Graefe G. "Multi-Table Joins through Bitmapped Join Indices" *SIGMOD Record*, Sep 1995.
[13] Harinarayan V., Rajaraman A., Ullman J.D. " Implementing Data Cubes Efficiently" *Proc. of SIGMOD Conf.*, 1996.
[14] Chaudhuri S., Krishnamurthy R., Potamianos S., Shim K. "Optimizing Queries with Materialized Views" *Intl.Conference on Data Engineering*, 1995.
[15] Levy A., Mendelzon A., Sagiv Y. "Answering Queries Using Views" *Proc. of PODS*, 1995. 16 Yang H.Z., Larson P.A. "Query Transformations for PSJ Queries", *Proc. of VLDB*, 1987.
[17] Kim W. "On Optimizing a SQL-like Nested Query" *ACM TODS*, Sep 1982.
[18] Ganski,R., Wong H.K.T., "Optimization of Nested SQL Queries Revisited " *Proc. of SIGMOD Conf.*, 1987.
[19] Dayal, U., "Of Nests and Trees: A Unified Approach to Processing Queries that Contain Nested Subqueries, Aggregates and Quantifiers" *Proc. VLDB Conf.*, 1987. 20 Murlaikrishna, "Improved Unnesting Algorithms for Join Aggregate SQL Queries" *Proc. VLDB Conf.*, 1992.
[21] Seshadri P., Pirahesh H., Leung T. "Complex Query Decorrelation" *Intl. Conference on Data Engineering*, 1996.
[22] Mumick I.S., Pirahesh H. "Implementation of Magic Sets in Starburst" *Proc.of SIGMOD Conf.*, 1994.Authors

First Author – Mr.Preyash Dholakia, M.C.A.,K.S.K.V Kachchh University,Bhuj and preyash_dholakia@yahoo.com.

Second Author – Mr. Dishek Mankad, M.C.A., B.R.Patel Institute of Computer Application[MCA Progrm], Dahemi-Anand , dishek.mankad@gmail.com.

Correspondence Author – Mr.Preyash Dholakia, M.C.A.,K.S.K.V Kachchh University,Bhuj and preyash_dholakia@yahoo.com.