

A Comparison Between Five Models Of Software Engineering

Nabil Mohammed Ali Munassar[1] and A. Govardhan[2]

[1]Ph.D Student of Computer Science & Engineering
Jawahrlal Nehru Technological University
Kuktapally, Hyderabad- 500 085, Andhra Pradesh, India

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

Abstract

This research deals with a vital and important issue in computer world. It is concerned with the software management processes that examine the area of software development through the development models, which are known as software development life cycle. It represents five of the development models namely, waterfall, Iteration, V-shaped, spiral and Extreme programming. These models have advantages and disadvantages as well. Therefore, the main objective of this research is to represent different models of software development and make a comparison between them to show the features and defects of each model.

Keywords: *Software Management Processes, Software Development, Development Models, Software Development Life Cycle, Comparison between five models of Software Engineering.*

1. Introduction

No one can deny the importance of computer in our life, especially during the present time. In fact, computer has become indispensible in today's life as it is used in many fields of life such as industry, medicine, commerce, education and even agriculture. It has become an important element in the industry and technology of advanced as well as developing countries. Now a days, organizations become more dependent on computer in their works as a result of computer technology. Computer is considered a time- saving device and its progress helps in executing complex, long, repeated processes in a very short time with a high speed. In addition to using computer for work, people use it for fun and entertainment. Noticeably, the number of companies that produce software programs for the purpose of facilitating works of offices, administrations, banks, etc, has increased recently which results in the difficulty of enumerating such companies. During the previous four decades, software has been developed from a tool used for analyzing information or solving a problem to a product in itself. However, the early programming stages have created a number of problems turning software an obstacle to software development particularly those relying on computers. Software consists of documents and programs that contain a collection that has been established to be a part of software engineering procedures. Moreover, the aim of software engineering is to create a suitable work that construct programs of high quality.

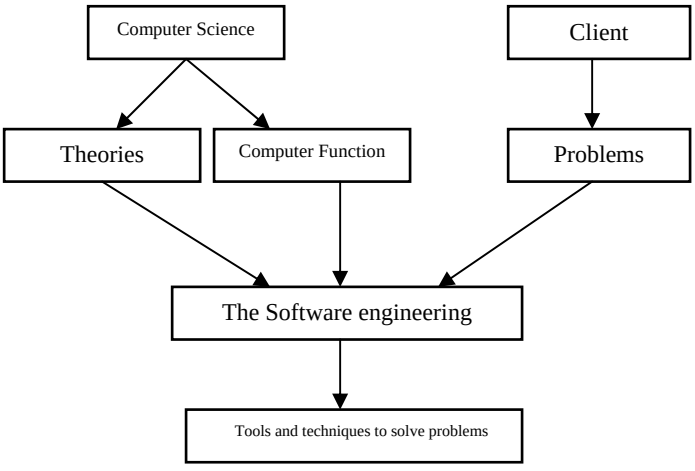

Fig. 1 Explanation of software engineering conception.

2. Software Process Models

A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective as:

1. Specification.
2. Design.
3. Validation.
4. Evolution.

General Software Process Models are

1. Waterfall model: Separate and distinct phases of specification and development.
2. Prototype model.
3. Rapid application development model (RAD).
4. Evolutionary development: Specification, development and validation are interleaved.
5. Incremental model.
6. Iterative model.
7. Spiral model.
8. Component-based software engineering : The system is assembled from existing components.

There are many variants of these models e.g. formal development where a waterfall-like process is used, but the specification is formal that is refined through several stages to an implementable design[1].

3. Five Models

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.

3.1 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 concern.

The pure waterfall lifecycle consists of several non-overlapping stages, as shown in the following figure. The model begins with establishing system requirements and software requirements and continues with architectural design, detailed design, coding, testing, and maintenance. The waterfall model serves as a baseline for many other lifecycle models.

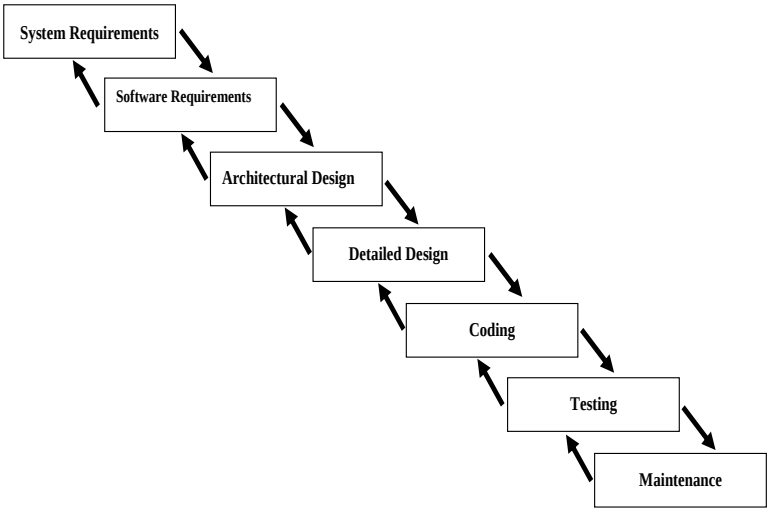

Fig. 2 Waterfall Model[4].

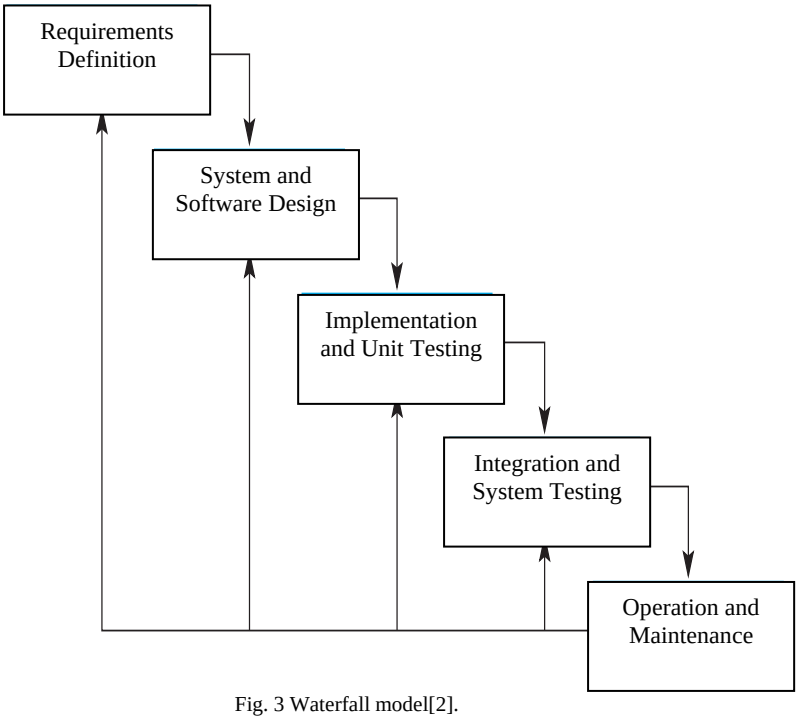

Fig. 3 Waterfall model[2].

The following list details the steps for using the waterfall

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 on 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 rework. Each completed phase requires formal review and 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.

Although the waterfall model has its weaknesses, it is instructive because it emphasizes important stages of project development. Even if one does not apply this model, he must consider each of these stages and its relationship to his own project [4].

- **Advantages :**

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

- **Disadvantages :**

1. Idealized, doesn't match reality well.
2. Doesn't reflect iterative nature of exploratory development.
   3. Unrealistic to expect accurate requirements so early in project.
4. Software is delivered late in project, delays discovery of serious errors.
5. Difficult to integrate risk management.
6. Difficult and expensive to make changes to documents, "swimming upstream".
7. Significant administrative overhead, costly for small teams and projects [6].

- **Pure Waterfall**

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

1. Concept.
2. Requirements.
3. Architectural design.

4. Detailed design.
5. Coding and development.
6. Testing and implementation.

Table 1: Strengths & Weaknesses of Pure Waterfall

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

- **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. • If there is personnel continuity between the phases, documentation can be substantially reduced. • Implementation of easy areas does not need to wait for the hard ones.	• Milestones are more ambiguous than the pure waterfall. • Activities performed in parallel are subject to miscommunication and mistaken assumptions. • Unforeseen interdependencies can create problems.

- **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 [5].

3.2 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.

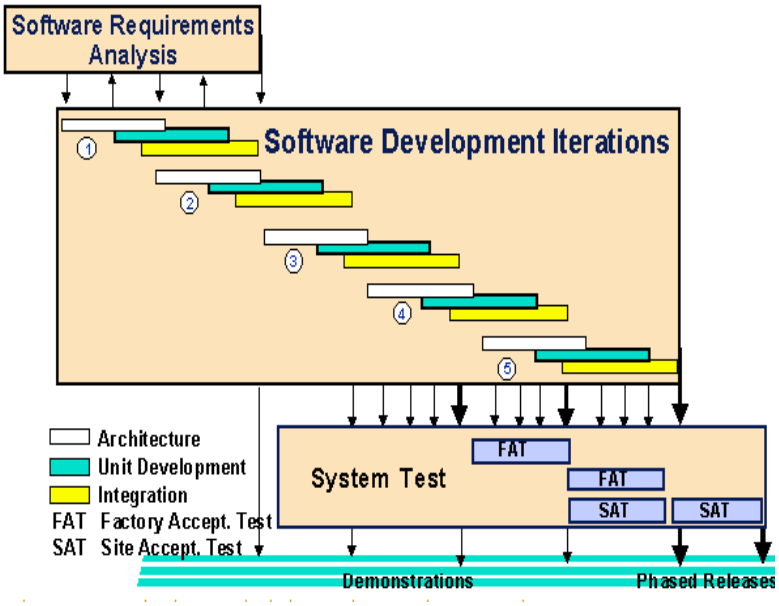

Fig. 4 Iterative Development.

3.3 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**

1. Simple and easy to use.
2. Each phase has specific deliverables.
3. Higher chance of success over the waterfall model due to the early development of test plans during the life cycle.
4. Works well for small projects where requirements are easily understood.

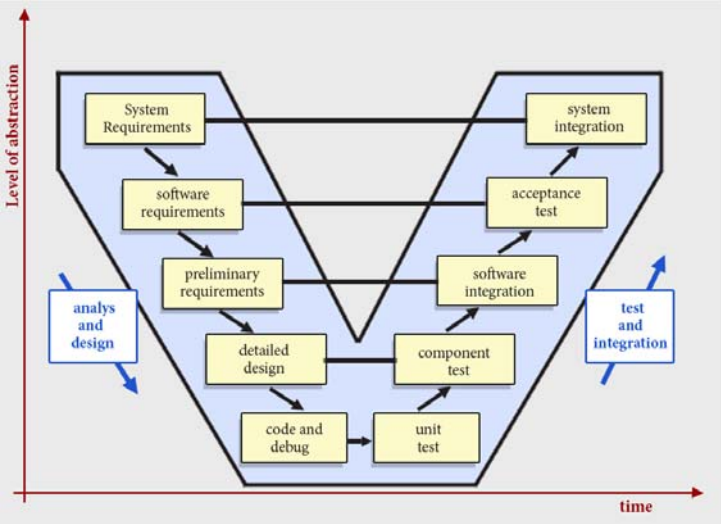

Fig. 5 V-Model [3]

- **Disadvantages**

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

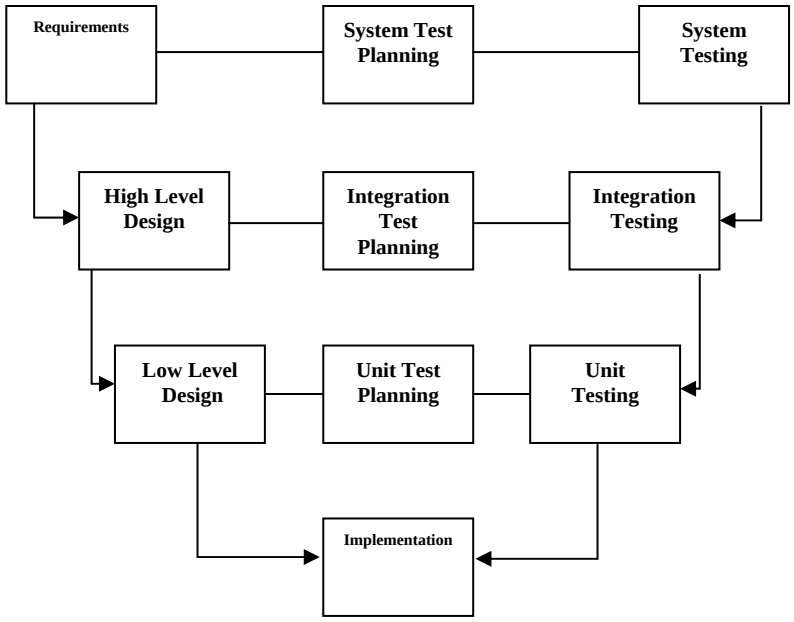

Fig. 6 V-Shaped Life Cycle Model[7].

3.4 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.

In the spiral model, the angular component represents progress, and the radius of the spiral represents cost.

- **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 analysis phase.
4. Doesn't work well for smaller projects [7].

- **Spiral model sectors**

1. Objective setting :Specific objectives for the phase are identified.
2. Risk assessment and reduction: Risks are assessed and activities are put in place to reduce the key risks.
3. Development and validation: A development model for the system is chosen which can be any of the general models.
4. Planning: The project is reviewed and the next phase of the spiral is planned [1].

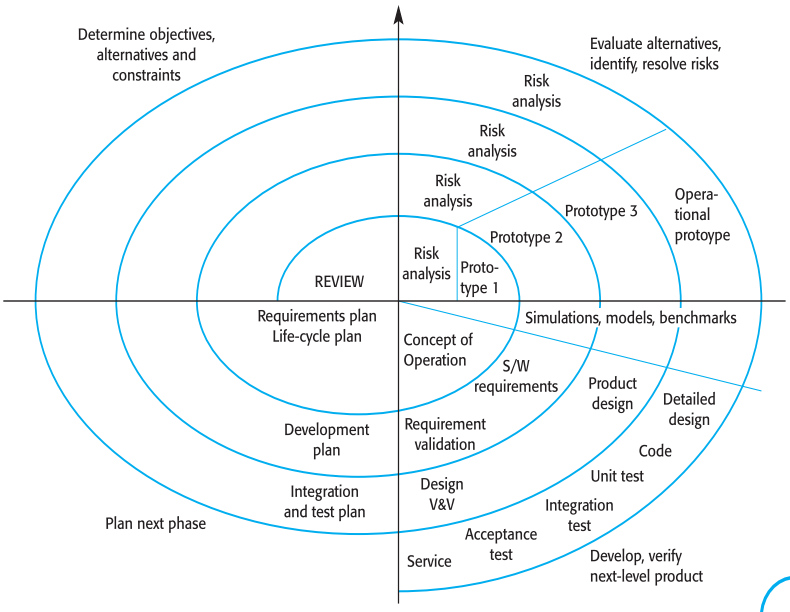

Fig. 7 Spiral Model of the Software Process[1].

- **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 areas to 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 [8].

3.5 Extreme Programming

An approach to development, based on the development and delivery of very small increments of 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.

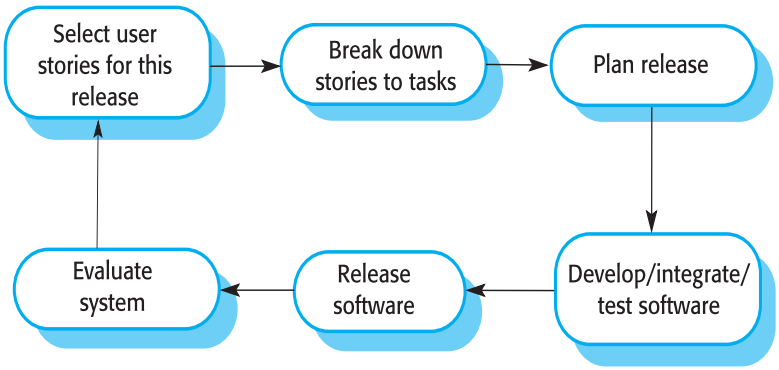

Fig. 8 The XP Release Cycle

- **Extreme Programming Practices**

Incremental planning: Requirements are recorded on Story Cards and the Stories to be included in a release are 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**

1. Incremental development is supported through small, frequent system releases.
2. Customer involvement means full-time customer engagement with the team.
3. People not process through pair programming, collective ownership and a process that avoids long working hours.
4. Change supported through regular system releases.
5. Maintaining simplicity through constant refactoring of code [1].

- **Advantages**

1. Lightweight methods suit small-medium size projects.
2. Produces good team cohesion.
3. Emphasises final product.
4. Iterative.
5. Test based approach to requirements and quality assurance.

- **Disadvantages**

1. Difficult to scale up to large projects where documentation is essential.
2. Needs experience and skill if not to degenerate into code-and-fix.
3. Programming pairs is costly.
4. Test case construction is a difficult and specialized skill [6].

4. Conclusion and Future Work

After completing this research , it is concluded that :

1. There are many existing models for developing systems for different sizes of projects and requirements.
2. These models were established between 1970 and 1999.
3. Waterfall model and spiral model are used commonly in developing systems.
4. Each model has advantages and disadvantages for the development of systems , so each model tries to eliminate the disadvantages of the previous model

Finally, some topics can be suggested for future works:

1. Suggesting a model to simulate advantages that are found in different models to software process management.
2. Making a comparison between the suggested model and the previous software processes management models.
3. Applying the suggested model to many projects to ensure of its suitability and documentation to explain its mechanical work.

REFERENCES

[1] Ian Sommerville, "Software Engineering", Addison Wesley, 7th edition, 2004.
[2] CTG. MFA – 003, "A Survey of System Development Process Models", Models for Action Project: Developing Practical Approaches to Electronic Records Management and Preservation, Center for Technology in Government University at Albany / Suny,1998 .
[3] Steve Easterbrook, "Software Lifecycles", University of Toronto Department of Computer Science, 2001.
[4] National Instruments Corporation, "Lifecycle Models", 2006 , http://zone.ni.com.
[5] JJ Kuhl, "Project Lifecycle Models: How They Differ and When to Use Them",2002 www.business-esolutions.com.
[6] Karlm, "Software Lifecycle Models', KTH,2006 .
[7] Rlewallen, "Software Development Life Cycle Models", 2005 ,http://codebeter.com.
[8] Barry Boehm, "Spiral Development: Experience, Principles, and Refinements", edited by Wilfred J. Hansen, 2000 .

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 area 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.