IJERMT

International journal of Engineering Research & Management Technology

ISSN:2348-4039
Volume-1,Issue-1
January 2014

An Analytical Study of Software Process Models : WHITE PAPER

Dr. Shweta Paradkar
Assistant Professor,
kalindi College,
University of Delhi

Dr. Diksha Grover
Assistant Professor
Rajdhani College,
University of Delhi

Abstract

This research deals with one of the crucial and imperative issues in computer world today. It focuses mainly on software management process which concerns with the software development through various software process models applied on software development life cycle. The models covered are Prescriptive model, Incremental process model, Evolutionary process model and Specialized process models. Each of these models suggests a somewhat different process flow but all have a series of generic phases in common. These models have their own advantages and disadvantages, hence the goal of the research is to present each of these models of software development and highlight their features and applicability for all various types of softwares.

1. ***Introduction:***

"Necessity is mother of invention" holds true for every field in our every day lives. The invent of one kind of technology has a profound and unexpected effect on totally unrelated technologies, industries, people and cultures across all the boundries.

Today, computers have successfully become an integral part across a broad spectrum of industry applications, namely medicine, education , commerce. It plays a very vital role in overall industrial and technical developments in both developed and developing nations. Software engineering has evolved from inconspicuous idea practiced by a relatively small number of practitioners to a full fledged engineering discipline. Software has become a catalyst that enables the creation of new technologies, enhancement of the existing ones and and the death of older technologies. Software has become one of the most critical factors to the advancement of human race.

The overall aim of software engineering is to provide models and processes that culminate in the production of a well-documented software that is easy to predict and maintain. The development organization follows a formal process while developing the software. The key component of the whole development process is the life cycle model on which the development is based. The important choice of the process model could significantly affect the development cost, effort and schedule. Life cycle of the software starts from concept development and ends at the retirement stage.

2. Software Process Model

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.

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

The various software process models covered are:

Prescriptive Models

Incremental Process Models

Evolutionary Process Models

2.1 Prescriptive Models

2.1.1 Traditional Software Life Cycle Models (Water Fall Model)

Traditional models of software evolution have been part of software engineering ever since its infancy. The classic software life cycle (or "waterfall chart") and stepwise refinement models are widely discussed in all programming practices. The incremental release model is closely related to industrial practices where it most often occurs. The software evolution can be described in a progressive manner, starting with requirements specification, preliminary design, and implementation; these usually have little or no further characterization other than a list of attributes that the product of such a stage should possess. Further, these models are independent of any organizational development setting, choice of programming language, software application domain, etc..

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 nonoverlapping 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.

The following list details the steps for using the waterfall

System requirements: Establishes the components for building the system, including the hardware requirements, software tools, and other necessary components. Software requirements: Establishes the expectations for software functionality and identifies which system requirements the software affects.
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.
Detailed Designs: Examines the software components defined in the architectural design stage and produces a specification for how each component is implemented.
Coding: Implements the detailed design specification.
Testing: Determines whether the software meets the specified requirements and finds any errors present in the code.
Maintenance: Addresses problems and enhancement requests after the software releases.

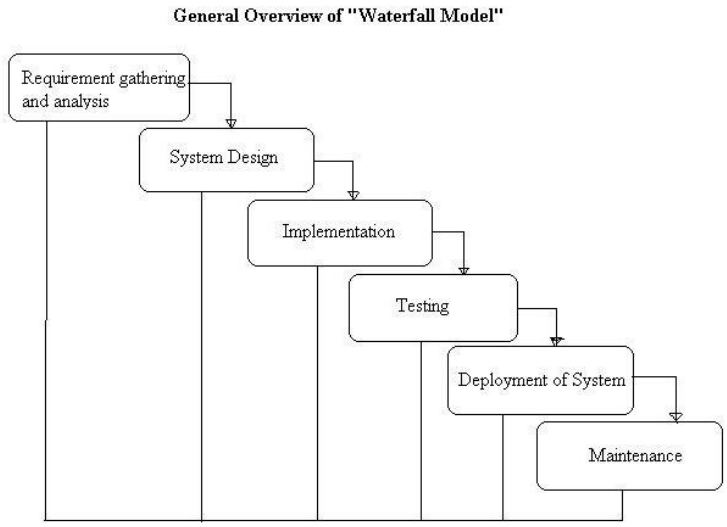

Fig 1. WaterFall Model.

Strenghts :
Easy to understand and implement.
Widely used and known (in theory!).
Reinforces good habits: define-before- design, design-before-code.
Identifies deliverables and milestones.

Document driven

Works well on mature products and weak teams.

Weakness :
Idealized, doesn't match reality well.
Doesn't reflect iterative nature of exploratory development.
Unrealistic to expect accurate requirements so early in project.
Software is delivered late in project, delays discovery of serious errors.
Difficult to integrate risk management.

Difficult and expensive to make changes to documents, "swimming upstream".

Significant administrative overhead, costly for small teams and projects .

2.2 Incremental Process Model
Developers often come across a situation in which the initial software requirements are reasonably well-defined, yet the overall scope of the software still eludes the developers. In addition, sometimes a working version of software based on initial requirements is to be given to customers which provide a limited set of functionality. Later on the product can be refined to provide an enhanced functionality in the next releases. This approach combines the sequential nature of the water fall model applied in iterative manner. This model applies linear sequences in a staggered fashion as the time progresses, producing a deliverable increment of the software. The first increment is known as "core product" where the basic requirements are met. This is then use and/or evaluated by the customer providing the necessary feedback to the developer. Later on with every increment the core product is modified until the complete product is delivered.

Strength:
Particularly useful when not sufficient manpower is available for a complete implementation by the deadline.
Early increments can be implemented with fewer people.
Few features can be postponed until later stage due to unavailability of some resources.

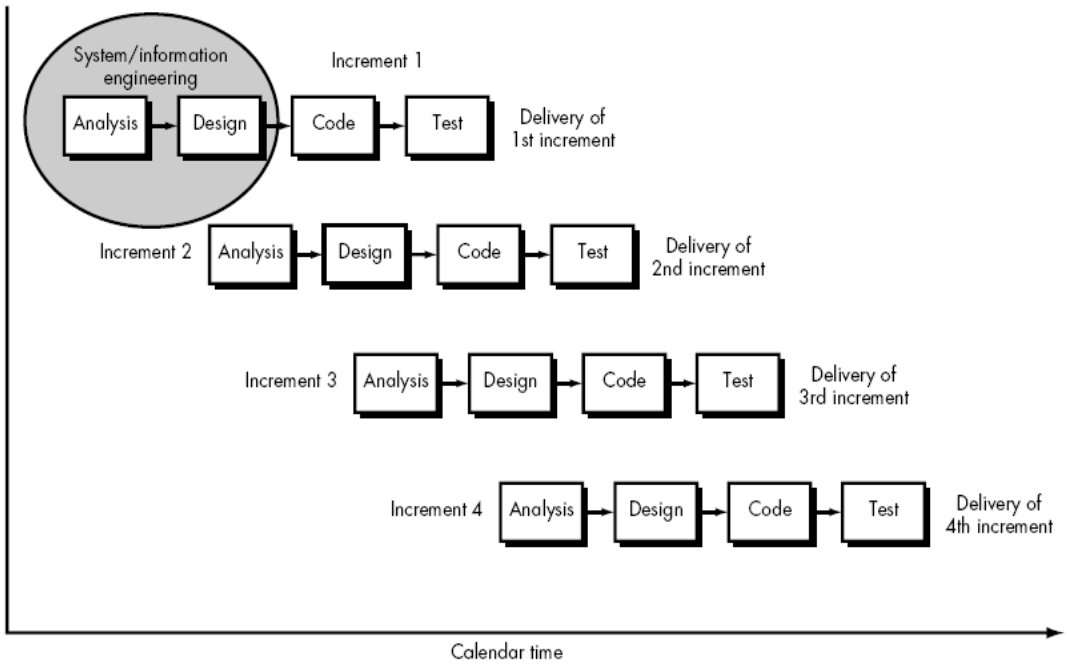

Fig.2 incremental model

2.2.1 ***Rapid Application Model***

This model works on the principle of short development cycle(e.g., 60 to 90 days). In approach it can viewed as a high speed version of waterfall model. This model requires the product scope to be constrained and requirement well understood. The rapid development is achieved by component based construction. Like all the other models it also encompasses the generic framework activities:

Communication: understanding the business problem.

Planning: managing the multiple teams working in parallel.

Modeling: includes business, data, and process modeling.

Construction: use of existing software components and automatic code generation.

Deployment: establishes the basis for subsequent iterations.

Strength:
Short development cycle
If scalable scope of project then a very efficient approach.

Weakness:

Requires a sufficient amount of human resource to build multiple teams.

If problem not modularized then can be ineffective approach.

Not very effective if technical risks are high.

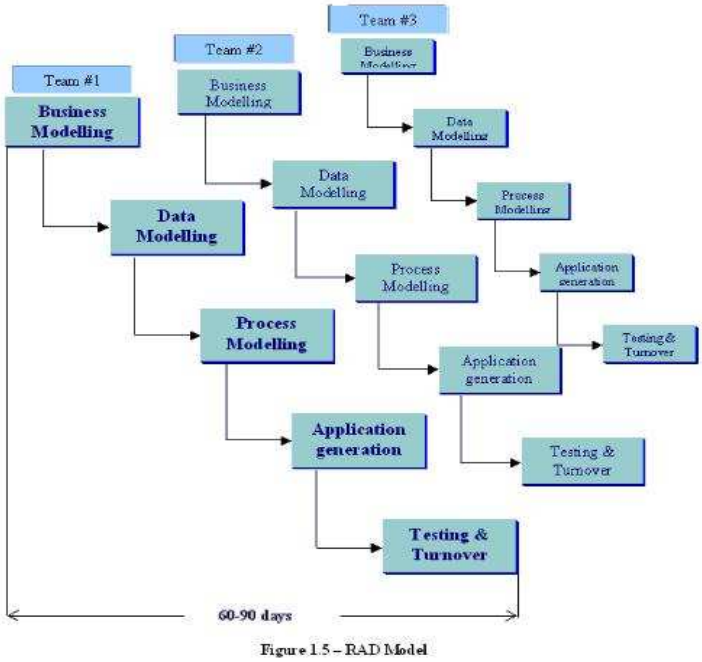

Figure 1.5 – RAD Model

Fig.3 RAD model

2.3 Evolutionary Process Models:
2.3.1 Prototyping
This model works in the scenario when the customer has legitimate needs but is clueless about the details, like input, processing or output requirements. On the other hand if the developer is unsure about a certain algorithm, man machine interface, platform in general, this approach proves to be very handy. It begins with communication between customer and developer where the goal of the system in general is discussed and those areas are identified which need further refinement. A quick design is planned that focuses on visible aspects of software to the customer. This leads to construction of a prototype which is handed over to customer to test

drive. Customer then gives feedback which is then used to refine the requirements further. This is continued till the time customer is satisfied and the requirements are met.

Strength:

The prototype serves as a mechanism for identifying software requirement.

The prototype can serve as the first system for understanding of requirement.

Weakness:

The prototype model is not the actual thing but a quick fix.

It is built using some inferior technology or inappropriate operating system or programming language.

The customer is not convinced about the term prototype as it can be mistaken for the actual thing.

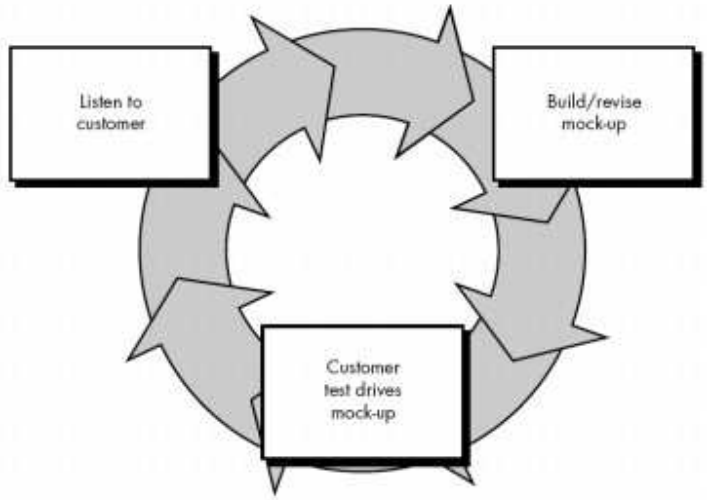

Fig.4 Prototype model

2.3.2Spiral Model

The spiral model is similar to the incremental model, withmore 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.

Strength:

High amount of risk analysis.Good for large and mission-critical projects.

Software is produced early in the software life cycle.

Weakness:

Can be a costly model to use.
Risk analysis requires highly specific expertise.
Project's success is highly dependent on the risk analysis phase.
Doesn't work well for smaller projects.
Spiral model sectors
Objective setting :Specific objectives for the phase are identified.
Risk assessment and reduction: Risks are assessed and activities are put in place to reduce the key risks.
Development and validation: A development model for the system is chosen which can be any of the general models.
Planning: The project is reviewed and the next phase of the spiral is planned.

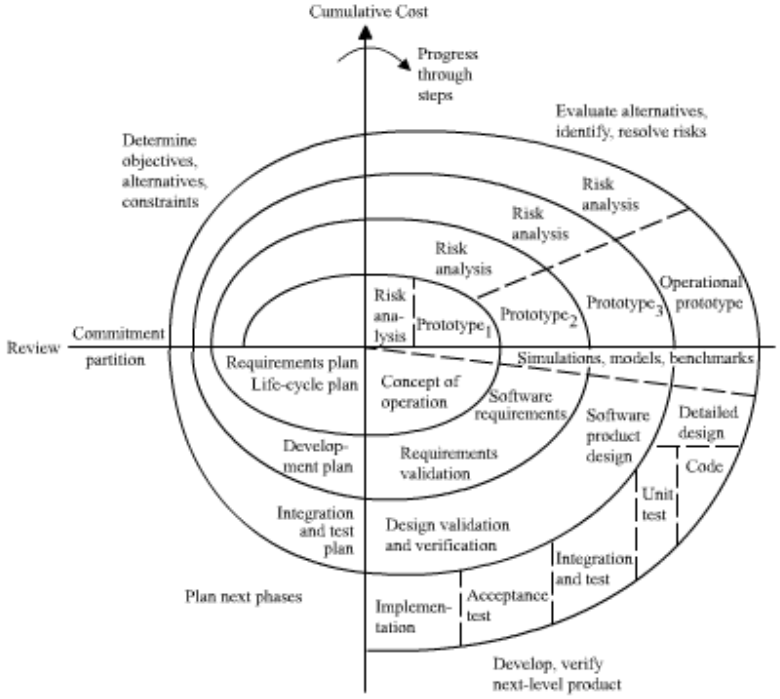

Fig. 4 Spiral Model

3. Specialized process models

3.1 Component-based development

These software components are developed by the vendors who pass them as products. These components provide the required functionality and well defined interfaces making them ready to use and can be incorporated into the software. The component-based development model exhibit few features of the spiral model. It is evolutionary in nature, having an iterative approach to the development of the software. However, the model composes application from prepackaged software components. Firstly the components are identified and then only the modeling and construction activities can begin with.the components designed can fall into conventional software modules or object-oriented classes categories. The component-based development model comprises of the following steps, regardless of the technology being used:

All the components are well researched in terms of the domain they will operate in and also evaluated.
Issues regarding integration are kept in mind.
Software architecture is designed to incorporate these components.
Components fit well into the architecture.
All the components are tested for the desired functionality.
The clear advantage of using component-based development model is reuse, leading to reduced effort, time and cost over substantial amounts.

3.2 The formal methods models

The formal methods are based on a pure mathematical approach to specify, develop and verify a computer based system by applying rigorous mathematical notations. This model carries out a set of activities that leads to formal mathematical specification of computer software. This approach helps to identify a set of problems which go undetected in case of the other models. Problem areas such as ambiguity, incompleteness and inconsistency can be discovered fairly easily with a mathematical model than the other models. When applied during the design phase, it performs the task of program verification and therefore enables the software engineers to discover and correct errors that might otherwise go undetected. Due to a complex approach, this model is not as popular as the other models, yet it gives better results when defect free software is required.

Strengths:

A very effective approach for building safety-critical software (aircraft avionics and medical devices). Any slipups in performance would result severe economics hardships for the developers.

Weakness:

Formal methods can be quite time consuming as well as costly for the developers.

Not every organization is equipped with personnel who are comfortable with approach; hence the team needs to be well trained for this.

It is very difficult to convince the unsophisticated customers about the technical details of this model.

4. Conclusion and Future Work

In the end, the study showed the various existing model for software development. These models have been in existence since last few decades. Based on the strengths and weaknesses of each of the models , their applicability ranges from all different kinds of application areas. Waterfall model is suited for

smaller projects which have clearly defined requirements. Incremental modal is good choice when customer is not able to spell out all the requirements at the onset, whereas Prototype model serves well for situations where customer is unable to clearly define the functionality of the system. The RAD model is applicable where there is short development cycle and Spiral model is more suited for large and complex systems.

Having discussed each model for their respective features, the upcoming trends in the software industry is for the Unified Process.

It is an attempt to draw best features and characteristics of conventional models, but characterize them in a way that implements many of the best principles of agile software development. It recognizes importance of customer communication , also focuses greatly on software architecture. It suggests a process flow that is iterative and incremental, providing the evolutionary feel that is essential in modern software development.

REFERENCES

Roger Pressman , " Software Engineering A Practitioner's Approach" sixth edition.

Pankaj Jalote, "An Integrated Approach to Software Engineering" third Edition.

Ian Sommerville "Software Engineering"Wesley, 7th edition, 2004.

Model driven Software Engineering Practices by Marco Brambilla