Ashwini Mujumdar, Gayatri Masiwal, P. M. Chawan / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 3, May-Jun 2012, pp.2015-2021

Analysis of various Software Process Models

Ashwini Mujumdar*, Gayatri Masiwal, P. M. Chawan*****
*(Department of Computer Science, VJTI, Mumbai)
** (Department of Computer Science, VJTI, Mumbai)
*** (Department of Computer Science, VJTI, Mumbai)

ABSTRACT
This paper conducts an analysis of various software development approaches, i.e. sequential, incremental, evolutionary, specialized and agile. An example of each approach is considered – Waterfall model (sequential approach), Incremental Model (incremental approach), Spiral Model (evolutionary approach), Formal Methods Model (specialized approach), Extreme Programming Model (agile approach) and RUP. This paper elaborates these models, as well as, it compares and contrasts between these different models.

Keywords - **About five key words in alphabetical order, separated by comma**

1. INTRODUCTION

Today, the computer has become a very crucial part of our life. It has become indispensible as it is used in various fields of life, such as, industry, medicine, education, commerce and even agriculture. Organizations have 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 working product that constructs programs of high quality.[1]

2. ACTIVITIES OF SOFTWARE DEVELOPMENT

Problem solving in software development consists of the following activities:

i. Understanding the problem
ii. Deciding a plan for the solution
iii. Coding the planned solution
iv. Testing the actual program [2]

These activities may be very complex for large systems. So, each of the activity has to be broken into smaller sub-activities or steps. These steps are then handled effectively to produce a software project or system.

The basic steps involved in software project development are:

i. Requirement analysis
ii. Design
iii. Coding
iv. Testing

In addition, there is a fifth step, "maintenance" that consists of maintaining the system after deployment, i.e. delivery to the customer. Unlike hardware, software does not wear out. But, it is very likely that some errors of the system, which were not found during the software testing phase, may be found by the customer. These errors or bugs need to be reported and resolved immediately. Also, over time, as newer technologies and platforms are developed, system starts becoming outdated. It is important to provide new features to the system after intervals and make it compatible with various latest platforms.

3. GENERAL APPROACHES

The various approaches to developing a software development process model are as follows:

3.1 Sequential Approach

Sequential approaches (e.g. waterfall model, V-model) refer to the completion of the work within one monolithic cycle. Projects are sequenced into a set of steps that are completed serially and typically span from determination of user needs to validation that the given solution satisfies the user. Progress is carried out in linear fashion enabling the passing of control and information to the next phase when pre-defined milestones are reached and accomplished. This approach is highly structured, provides an idealised format for the contract and allows maximum control over the process. On the other hand, it is also resistant to change and the need for corrections and re-work. Note that some variations, as well as Royce's original formulation of the model, allow for revision and re-tracing and may also incorporate prototyping or other requirements gathering sequences encompassed within the overall sequence frame[6].

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.
The steps followed in the waterfall model are:

i. **Communication:** establishes the expectations of the stakeholders and hence useful in requirements gathering.
ii. **Planning:** develops a well-defined plan of execution of the project.
iii. **Modeling:** develops a model of the project before developing the actual project.
iv. **Construction:** builds the actual project following the plan of execution defined in the planning stage and testing.
v. **Deployment:** the delivery of end-product to the customer and its maintenance.

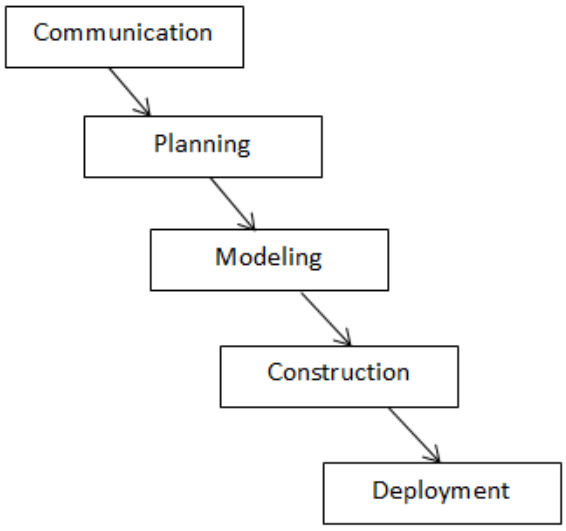

Fig. 1 Waterfall Model

The advantages of waterfall model are:

- Easy to understand and implement
- Reinforces good habits: define-before-design and design-before-code.
- Identifies deliverables and milestones
- Works well on mature products and weak teams[1]

The disadvantages of the waterfall model are:

- Real projects rarely follow the sequential approach
- There is uncertainty at the beginning of the project regarding requirements and goals. This model does not accommodate these uncertainties very well.
- It does not yield a working version of the system until late in the process.[7]

3.1 Incremental Approaches:

Incremental approaches emphasize phased development by offering a series of linked mini-projects (referred to as increments, releases or versions) working from a pre-defined requirements specification up front. Work on different parts and phases, is allowed to overlap throughout the use of multiple mini-cycles running in parallel. Each mini-cycle adds additional functionality and capability. The approach is underpinned by the assumption that it is possible to isolate meaningful subsets that can be developed, tested and implemented independently. Delivery of increments is staggered as calendar time progresses. The first increment often acts as the core product providing the functionality to address the basic requirements. The staggered release philosophy allows for learning and feedback which can modify some of the customer requirements in subsequent versions. Incremental approaches are particularly useful when the full complement of personnel required to complete the project is not available and when there is an inability to fully specify the required product or to fully formulate the set of expectations[6].

The Incremental Model:

There are many situations in which initial software requirements are reasonably well-defined, but the overall scope of the development effort precludes a purely linear process. In addition, there may be a compelling need to provide a limited set of software functionalities to a user quickly and then refine and expand on that functionality in later software releases. In such cases, a process model that is designed to produce the software in increments is chosen.[7]

The incremental model combines elements of the waterfall model in an iterative fashion. Each linear sequence produces deliverable "increments" of the software. The first increment is the *core product*. That is, basic requirements are addressed, but many supplementary features (some known, others unknown) remain undelivered.[7]

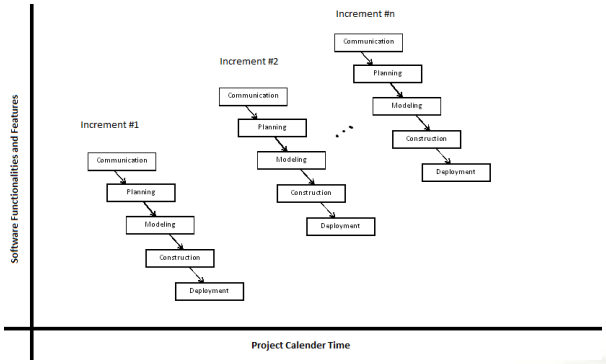

Fig. 2 Incremental Model

The advantages of the incremental model are:

- Divides project into smaller parts
- Creates working model early and provides valuable feedback
- Feedback from one phase provides design information for the next phase
- Very useful when more staffing is unavailable

The disadvantages of the incremental model are:

- User community needs to be actively involved in the project. This demands on time of the staff and add project delay
- Communication and coordination skills take a center stage
- Informal requests for improvement for each phase may lead to confusion
- It may lead to "scope creep"

3.2 Evolutionary Approaches:

Evolutionary approaches recognize the great degree of uncertainty embedded in certain projects and allow developers and managers to execute partial versions of the project while learning and acquiring additional information and gradually evolving the conceptual design. Evolutionary projects are defined in a limited sense allowing a limited amount of work to take place before making subsequent major decisions. Projects can start with a macro estimate and general directions allowing for the fine details to be filled-in in evolutionary fashion. The initial implementation benefits from exposure to user comments leading to a series of iterations. Finite goals are thus allowed to evolve based on the discovery of user needs and changes in expectations along the development route. Projects in this category are likely to be characterized by a high degree of technological risk and lack of understanding of full implications by both stakeholders and developers. Evolutionary approaches are particularly effective in change-intensive environments or where resistance to change is likely to be strong.

The Spiral Model:

The spiral development model is a risk driven process model generator that is used to guide multi-stakeholder concurrent engineering of software intensive systems. It has two main distinctive features. One is *cyclic* approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk. The other is a set of anchor-point milestones for ensuring the stakeholder commitment to feasible and mutually satisfactory system solutions.[7]

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). In 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.[1]

In the spiral model, the angular component represents progress, and the radius of the spiral represents cost.[1]

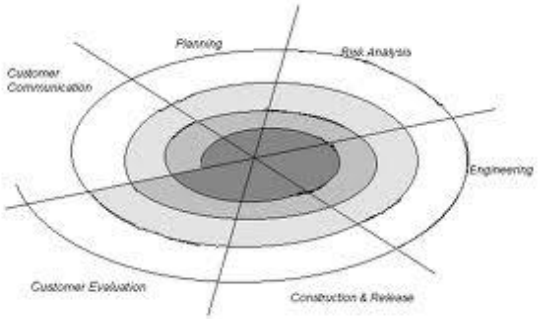

Fig. 3 Spiral Model

A spiral model is divided into various activities which include Analysis, Design, Implementation, Testing and Deployment. The spiral is implemented in a clockwise fashion, beginning at the center and working its way outwards, during which it passes through each of the above regions.

A spiral model is divided into a number of framework activities, also called *task regions.* Typically, there are between three and six task regions. Figure 2.8 depicts a spiral model that contains six task regions:

• **Customer communication**—tasks required to establish effective communication between developer and customer.

• **Planning**—tasks required to define resources, timelines, and other project related information.

• **Risk analysis**—tasks required to assess both technical and management risks.
• **Engineering**—tasks required to build one or more representations of the application.
• **Construction and release**—tasks required to construct, test, install, and provide user support (e.g., documentation and training).[7]

Unlike other process models that end when software is delivered, the software model and be adapted to apply throughout the life of the computer software. The first circuit around the spiral might represent a "concept development project" which starts at the core of the spiral and may continue for several iterations till the concept development is complete. If the concept is to be developed into an actual project, the process proceeds outwards on the spiral and a "new product development phase" commences. The new product will evolve through a number of iterations around the spiral, following the path that bounds the region that has somewhat lighter shading than the core. In essence, the spiral, when characterized in this way, remains operative until the software is retired. There are times when the process is dormant, but whenever a change is initiated, the process starts at the appropriate entry point (e.g., product enhancement).[7]

The advantages of spiral model are:

- Was designed to include the best features form Waterfall and Prototyping Model
- Good for large and mission-critical projects
- Introduces a new component – risk assessment
- Similar to prototyping model, an initial version of system is developed and modified based on input from customer

The disadvantages of the spiral model are:

- Can be a costly model to use
- Risk analysis requires highly specific expertise
- Project's success id highly dependent on risk analysis phase
- Doesn't work well for smaller projects

Specialized process models can take the characteristics of any or many of the conventional models presented in the above sections. However specialized models tend to be applied when a narrowly defined software engineering approach is chosen[7].

In some cases, these specialized models might better be characterized as a collection of techniques or a methodology for accomplishing a specific software development goal.[7] However, they do imply a process model which is highly project specific.

The Formal Methods Model:

The *formal methods* model encompasses a set of activities that leads to formal mathematical specification of the project or the computer software to be developed. Formal methods enable a software engineer to specify, develop and verify a computer-based system by applying a rigorous mathematical notation. A variation on this approach, called clean-room software engineering, is currently applied by some software development organizations.

When formal methods are used during development, they provide a mechanism for eliminating many of the problems that are difficult to overcome using other software engineering paradigms. Ambiguity, incompleteness and inconsistency can be discovered and corrected more easily – not through ad-hoc review, but through the application of mathematical analysis. When formal methods are used during design, they serve as a basis for program verification and therefore enable the software engineer to discover and correct errors that might otherwise go undetected.

Although not a mainstream approach, the formal methods model offers the promise of defect-free software. Yet, concern about its application in business environment has been voiced:

- The development of formal models is currently quite time-consuming and expensive.
- Because few software developers have the necessary background to apply formal methods, extensive training is required.
- It is difficult to use the models as a communication mechanism for technically unsophisticated customers.

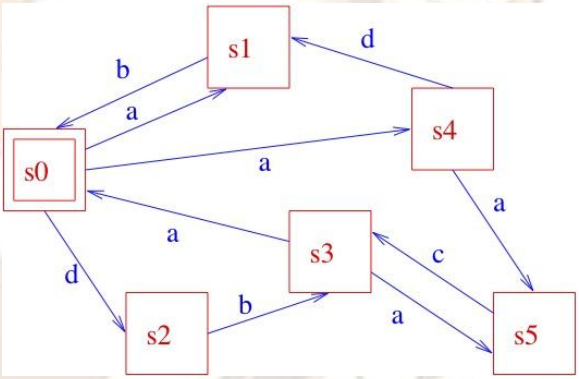

Fig. 4 Formal Methods Model Analysis

These concerns notwithstanding, the formal methods approach has gained adherents among software developers who must build safety-critical software, such as aircraft avionics and medical devices, and among developers who would suffer serious economic hardship, should software errors occur.

3.3 Agile Approaches:

Agile development is claimed to be a creative and responsive effort to address users' needs focused on the requirement to deliver relevant working business applications quicker and cheaper. The application is typically delivered in incremental (or evolutionary or iterative) fashion. The agile development approaches are typically concerned with maintaining user involvement through the application of design teams and special workshops. The delivered increments tend to be small and limited to short delivery periods to ensure rapid completion. The management strategy utilized relies on the imposition of *timeboxing*, the strict

delivery to target which dictates the scoping, the selection of functionality to be delivered and the adjustments to meet the deadlines. Agile development is particularly useful in environments that change steadily and impose demands of early (partial) solutions. Agile approaches support the notion of concurrent development and delivery within an overall planned context.

Extreme Programming:

It is 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.[1]

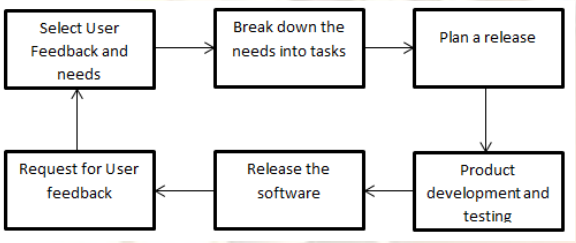

Fig. 5 Extreme Programming

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. Emphasizes 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.[1]

Each of the approaches described above appears to have clear benefits, at least from a theoretical perspective. However the variety of different approaches leads to a dilemma when it comes to selecting the most suitable one for a project. At the beginning of every project the manager is expected to commit to a development approach. This is often driven by past experience or other projects that are, or have been, undertaken by the organization. Project managers are expected to select the most suitable approach that will maximize the chances of successfully delivering a product that will address the client's needs and prove to be both useful and usable.

The choice should clearly relate to the relative merits of each approach.

3.4 Rational Unified Process

The Rational Unified Process (RUP) is an iterative software development process framework created by the Rational Software Corporation. RUP is not a single concrete prescriptive model, but rather an adaptable process

framework, intended to be tailored by the development organizations and software project teams that will select the elements of the process that are appropriate for their needs. RUP is a specific implementation of the Unified Process.

RUP is based on a set of building blocks, or content elements, describing what is to be produced, the necessary skills required and the step-by-step explanation describing how specific development goals are to be achieved.

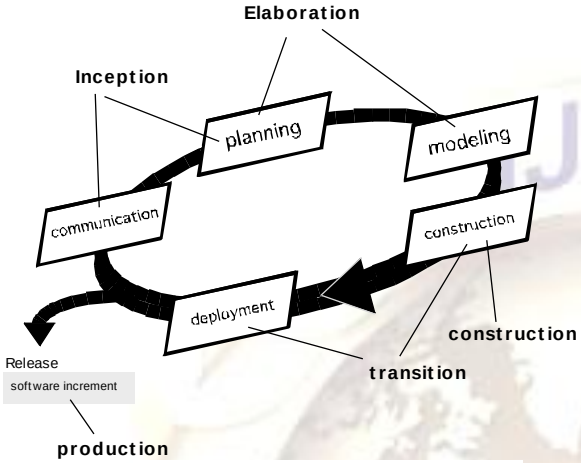

Fig. 6 Rational Unified Process [7]

1) Four Project Life cycle Phases:

The RUP has determined a project life cycle consisting of four phases. These phases allow the process to be presented at a high level in a similar way to how a 'waterfall'-styled project might be presented, although in essence the key to the process lies in the iterations of development that lie within all of the phases. Also, each phase has one key objective and milestone at the end that denotes the objective being accomplished. The visualization of RUP phases and disciplines over time is referred to as the RUP hump chart. [7]

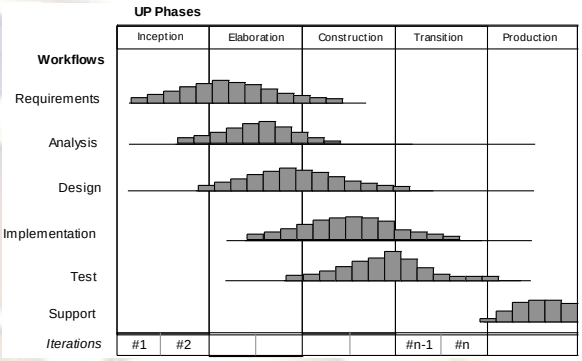

Fig. 7 Four phases of Rational Unified Process [7]

The various process models discussed above can be summarized as follows:

TABLE 1. Comparison between different software development models

Model/Features	Waterfall	Incremental	Spiral	Agile	RUP
Requirement Specifications	Beginning	Beginning	Beginning	Frequently changed	Beginning
Cost	Low	Low	Expensive	Very High	Expensive
Resource Control	Yes	Yes	Yes	No	Yes
Simplicity	Simple	Intermediate	Intermediate	Complex	Simple and clear
Risk Analysis	Only at beginning	No risk analysis	Yes	Yes	Yes
User Involvement	Only at beginning	Intermediate	High	High	Only at beginning of last phase
Flexibility	Rigid	Less Flexible	Flexible	Highly Flexible	Considerable
Reusability	Limited	Yes	Yes	*Use Case* reuse	Supports reusability of existing classes

4. CONCLUSION

After completing this analysis, we have concluded that there are many existing models for developing systems and project requirements. Of these, waterfall model and spiral model are more commonly used than the others. Each model has advantages and disadvantages. Each model tries to eliminate the disadvantages of the previous model.

References

[1] Nabil Mohammed Ali Munassar1 and A. Govardhan, A Comparison Between Five Models Of Software Engineering, *IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 5, September 2010*

[2] Sanjana Taya and Shaveta Gupta, Comparative Analysis of Software Development Life Cycle Models, *IJCST Vol. 2, Iss ue 4, Oct . - Dec. 2011*

[3] Alan M. Davis, Edward H. Bersoff and Edward R. Comer, A Strategy for comparing Alternative Software Development Life Cycle Models, *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 14, NO. 10, OCTOBER 1988*

[4] Nicholas H. Malcolm, Software Development Life Cycles: History and Future

[5] Jim Hurst, Comparing Software Development Life Cycles

[6] Oddur Benediktsson, Darren Dalcher and Helgi Thorbergsson, Comparison of Software Development Life Cycles: A Multiproject Experiment

[7] Roger Pressman, *Software Engineering: A Practitioner's Approach*, Sixth Edition, McGraw-Hill Publication