E-notes Software Engineering

By Sagar N. Gharate

Syllabus

Chapter 1 : Introduction To Software Engineering and Process Models

References :

1. Software Engineering : A Practitioner's Approach - Roger S. Pressman, McGraw hill(Eighth Edition) ISBN-13: 978-0-07-802212-8, ISBN-10: 0-07-802212-6
2. A Concise Introduction to Software Engineering - PankajJalote, Springer ISBN: 978-1-84800-301-9
3. The Unified Modeling Language Reference Manual - James Rambaugh, Ivar Jacobson, Grady Booch ISBN 0-201-30998-X

Chapter 1

Software Engineering

Software Engineering

The term software engineering is composed of two words, software and engineering.

Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be a collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **software product.**

Engineering on the other hand, is all about developing products, using welldefined, scientific principles and methods.

Software Engineering:

So, we can define **software engineering** as an engineering branch associated with the development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

IEEE defines software engineering as: The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.

We can also define software engineering as an engineering branch associated with the development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

We can alternatively view it as a systematic collection of past experience. The experience is arranged in the form of methodologies and guidelines. A small program can be written without using software engineering principles. But if one wants to develop a large software product, then software engineering principles are absolutely necessary to achieve a good quality software cost effectively.

The Nature of Software Engineering

Today, software takes on a dual role. It is a product and, at the same time, the vehicle for delivering a product. As a product, it delivers the computing potential embodied by computer hardware or, more broadly, a network of computers that are accessible by local hardware. Whether it resides within a cellular phone or operates inside a mainframe computer, software is an information transformer producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as a single bit or as complex as a multimedia presentation. As the vehicle used to deliver the product, software acts as the basis for the control of the computer (operating systems), the communication of information (networks), and the creation and control of other programs (software tools and environments).

Software delivers the most important product of our time—information. The role of computer software has undergone significant change over a time span of little more than 50 years. Dramatic improvements in hardware performance, profound changes in computing architectures, vast increases in memory and storage capacity, and a wide variety of exotic input and output options have all precipitated more sophisticated and complex computer-based systems.

The lone programmer of an earlier era has been replaced by a team of software specialists, each focusing on one part of the technology required to deliver a complex application. And yet, the same questions asked of the lone programmer are being asked when modern computer-based systems are built:

- Why does it take so long to get software finished?
- Why are development costs so high?
- Why can't we find all the errors before we give the software to customers?
- Why do we continue to have difficulty in measuring progress as software is being developed?

These, and many other questions are a manifestation of the concern about software and the manner in which it is developed a concern that has lead to the adoption of software engineering practice.

Software Evolution

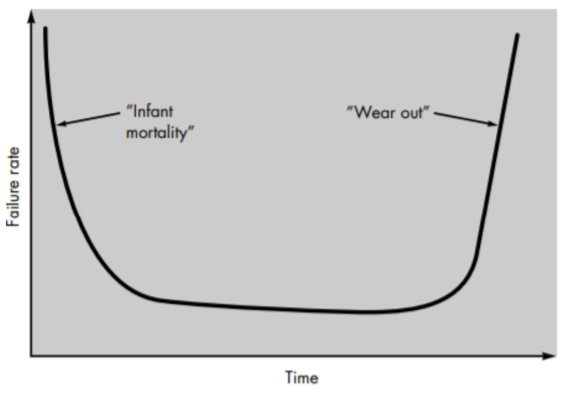
The process of developing a software product using software engineering principles and methods is referred to as Software Evolution. This includes the initial development of software and its maintenance and updates, till desired software product is developed, which satisfies the expected requirements.

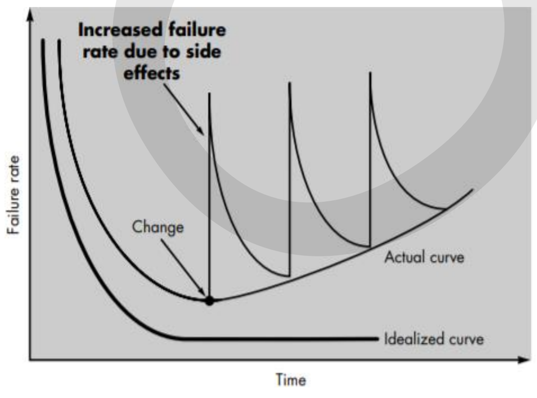
We can alternatively view it as a systematic collection of past experience. The experience is arranged in the form of methodologies and guidelines. A small program can be written without using software engineering principles. But if one wants to develop a large software product, then software engineering principles are absolutely necessary to achieve a good quality software cost effectively.

Software Characteristics:

Software is a logical rather than a physical system element. Therefore, software has characteristics that are considerably different than those of hardware:

1. Software is developed or engineered, it is not manufactured in the classical sense.
   - Although some similarities exist between software development and hardware manufacture, the two activities are fundamentally different.
   - In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent (or easily corrected) for software.
   - Both activities require the construction of a "product" but the approaches are different.
   - Software costs are concentrated in engineering. This means that software projects cannot be managed as if they were manufacturing projects.





2. Software doesn't "wear out."

- Figure 1.1 depicts failure rate as a function of time for hardware.It indicates that hardware exhibits relatively high failure rates early in its life (these failures are often attributable to design or manufacturing defects);
- Defects are corrected and the failure rate drops to a steady-state level (ideally, quite low) for some period of time.
- As time passes, however, the failure rate rises again as hardware components suffer from the cumulative effects of dust, vibration, abuse, temperature extremes, and many other environmental maladies.
- Stated simply, the hardware begins to wear out. Software is not susceptible to the environmental maladies that cause hardware to wear out.

3. Although the industry is moving toward component-based assembly, most software continues to be custom built.

- A software component should be designed and implemented so that it can be reused in many different programs.
- Modern reusable components encapsulate both data and the processing applied to the data, enabling the software engineer to create new applications from reusable parts.
- For example, today's graphical user interfaces are built using reusable components that enable the creation of graphics windows, pull-down menus, and a wide variety of interaction mechanisms.
- The data structure and processing detail required to build the interface are contained with a library of reusable components for interface construction.

Software Applications Domain:

Software may be applied in any situation for which a pre-specified set of procedural steps (i.e., an algorithm) has been defined. Information content and determinacy are important factors in determining the nature of a software application. Content refers to the meaning and form of incoming and outgoing information. For example, many business applications use highly structured input data (a database) and produce formatted “reports.”

The following software areas indicate the breadth of potential applications:

System software

System software is a collection of programs written to service other programs. Some system software (e.g., compilers, editors, and file management utilities) process complex, but determinate, information structures. Other systems applications (e.g., operating system components, drivers, telecommunications processors) process largely indeterminate data.

Real-time software

Software that monitors/analyzes/controls real-world events as they occur is called real time. Elements of real-time software include a data gathering component that collects and formats information from an external environment, an analysis component that transforms information as required by the application, a control/output component that responds to the external environment, and a monitoring component that coordinates all other components so that real-time response (typically ranging from 1 millisecond to 1 second) can be maintained.

Business software

Business information processing is the largest single software application area. Discrete "systems" (e.g., payroll, accounts receivable/payable, inventory) have evolved into management information system (MIS) software that accesses one or more large databases containing business information. Applications in this area restructure existing data in a way that facilitates business operations or management decision making.

Engineering and scientific software

Engineering and scientific software have been characterized by "number crunching" algorithms. Applications range from astronomy to volcanology, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing. However, modern applications within the engineering/scientific area are moving away from conventional numerical algorithms. Computer-aided design, system simulation, and other interactive applications have begun to take on real-time and even system software characteristics.

Embedded software

Intelligent products have become commonplace in nearly every consumer and industrial market. Embedded software resides in read-only memory and is used to control products and systems for the consumer and industrial markets. Embedded software can perform very limited and esoteric functions (e.g., keypad control for a microwave oven) or provide significant function and control capability (e.g., digital functions in an automobile such as fuel control, dashboard displays, and braking systems).

Personal computer software

The personal computer software market has burgeoned over the past two decades. Word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, personal and business financial applications, external network, and database access are only a few of hundreds of applications.

Web-based software

The Web pages retrieved by a browser are software that incorporates executable instructions (e.g., CGI, HTML, Perl, or Java), and data (e.g., hypertext and a variety of visual and audio formats). In essence, the network becomes a massive computer providing an almost unlimited software resource that can be accessed by anyone with a modem.

Artificial intelligence software

Artificial intelligence (AI) software makes use of non-numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis. Expert systems, also called knowledgebase systems, pattern recognition (image and voice), artificial neural networks, theorem proving, and game playing are representative of applications within this category

- **Software Engineering Layers: A Layered Approach**

Software engineering is a layered technology. Referring to Figure 1.2, any engineering approach (including software engineering) must rest on an organizational commitment to quality. Total quality management and similar philosophies foster a continuous process improvement culture, and this culture ultimately leads to the development of increasingly more mature approaches to software engineering.

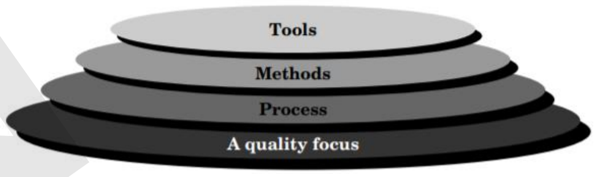


Fig. 1.2 Software engg layers

Quality Focus

- Every organization is rest on its commitment to quality.
- Total quality management, Six Sigma, or similar continuous improvement culture.
- The bedrock that supports software engineering is a quality focus.
- Ultimately leads to development of increasingly more effective approaches to software engineering.
- The bedrock that supports software engineering is a quality focus

Process

- Software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software.
- Process defines a framework for a set of key process areas (KPAs) [PAU93] that must be established for effective delivery of software engineering technology.

Method

- Software engineering methods provide the technical “how-to's” for building software. Methods encompass a broad array of tasks that include requirements analysis, design, program construction, testing, and support.
- Software engineering methods rely on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.

Tools

- Software engineering tools provide automated or semi-automated support for the process and the methods.
- When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer-aided software engineering, is established. CASE combines software, hardware, and a software engineering database to create a software engineering environment analogous to CAD/CAE (computer-aided design/engineering) for hardware.

- ## Software Process
  - A software process is a collection of activities, actions, and tasks that are required to build high-quality software.
  - A process defines who is doing what when and how to reach a certain goal.
  - The aim of software process is effective on-time delivery of software with quality

Elements of a software Process

Activity:

- An activity helps to achieve a broad objective (e.g., communication with stakeholders) and is applied regardless of the application domain, size, complexity and degree of rigor with which software engineering is to be applied.

Action

- Actions consist of the set of tasks that is used to produce the product.

Task

- A task focuses on a small, but well-defined objective that produces a tangible outcome.

- **Process & Generic Process Model:**

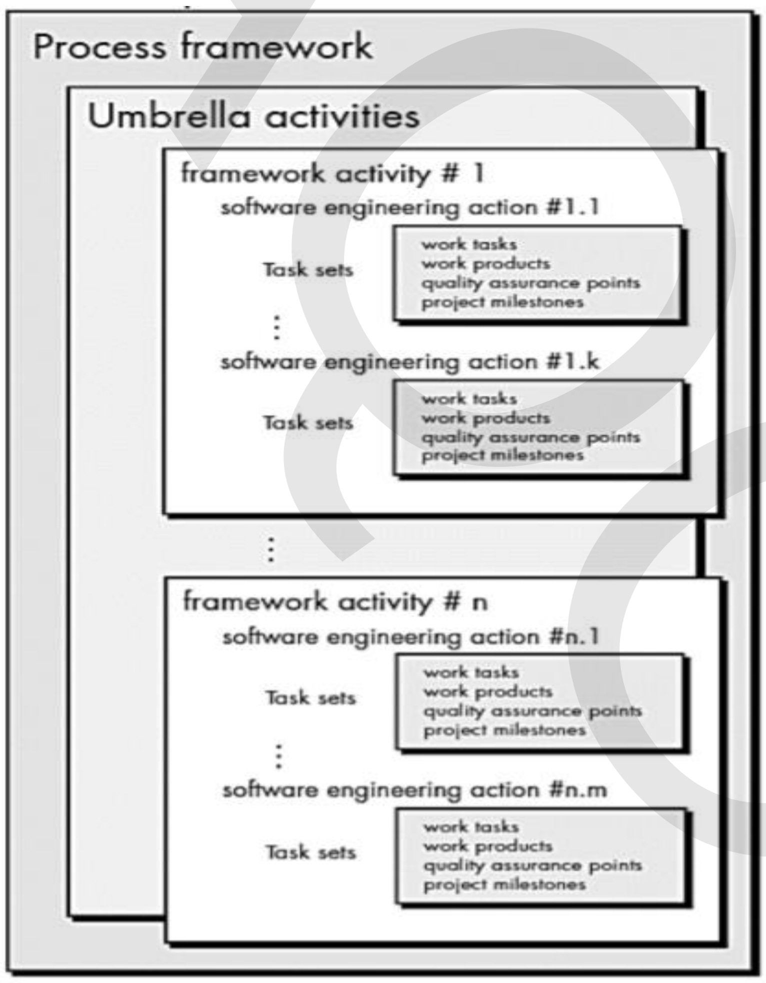
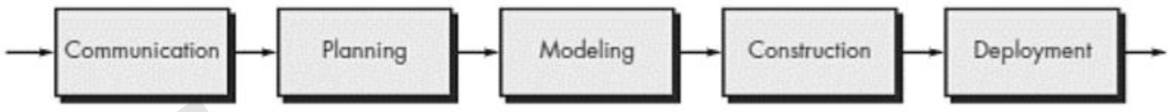


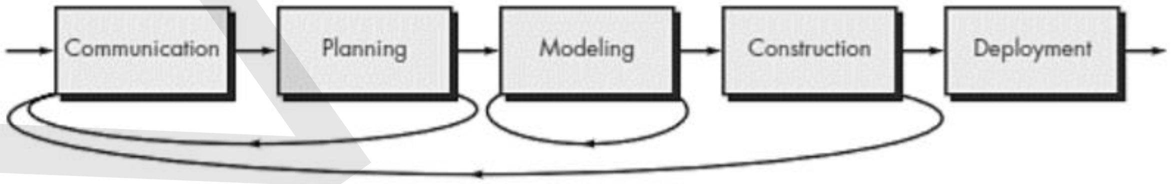
Figure 1.3Generic Process Model

- A software process is defined as a collection of work activities, actions, and tasks that are performed when some work product is to be created.
- A software process can be characterized as shown in Figure 1.3. A common process framework is established by defining a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.
- Each of these activities, actions, and tasks reside within a framework or model that defines their relationship with the process and with one another. Each framework activity is populated by a set of software engineering actions.
- Each software engineering action is defined by a task set that identifies the work tasks that are to be completed, the work products that will be produced, the quality assurance points that will be required and the milestones that will be used to indicate progress.
- A generic process framework for software engineering defines five framework activities-communication, planning, modeling, construction, and deployment. In addition, it defines a set of umbrella activities.
- Finally, umbrella activities such as software quality assurance, software configuration management, and measurement overlay the process model. Umbrella activities are independent of any one framework activity and occur throughout the process.

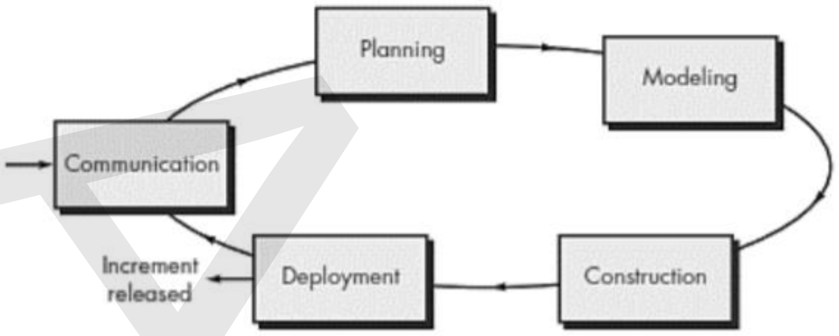
Process Framework:



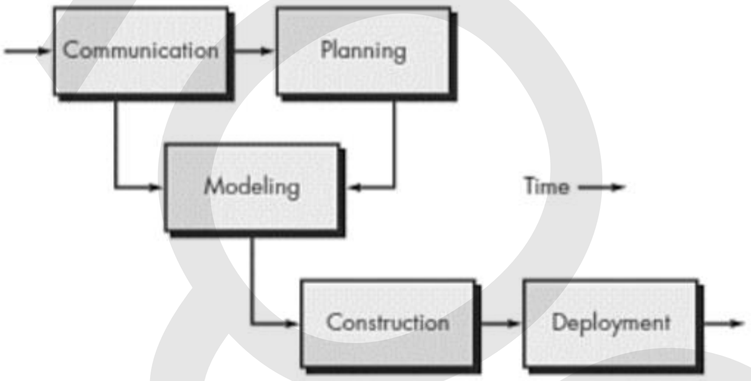
(a) Linear process flow



(b) Iterative process flow



(c) Evolutionary process flow



(d) Parallel process flow

Process flow describes how the framework activities and the actions and tasks that occur within each framework activity are organized with respect to sequence and time.

- A linear process flow executes each of the five framework activities in sequence, beginning with communication and culminating with deployment.
- An iterative process flow repeats one or more of the activities before proceeding to the next.
- An evolutionary process flow executes the activities in a "circular" manner. Each circuit through the five activities leads to a more complete version of the software.
- A parallel process flow executes one or more activities in parallel with other activities.

Process framework activities:

Communication -

- Before starting any technical work, it is important to communicate and collaborate with the customer and other stakeholders.
- The main objective is to understand stakeholders' objectives for the project and to gather requirements that help to define software features and functions.

Planning -

- Software project plan defines the software engineering work by describing the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.

Modeling -

- Model helps to better understand software requirements and the design that will achieve those requirements.

Construction -

- This activity combines code generation and the testing that is required to uncover the errors in the code.

Deployment –

- The software is delivered to the customer, who evaluates the product and provides feedback based on the evaluation.
- Generally, the framework activities are applied iteratively as a project progresses.
- Each project iteration produces a software increment that provides stakeholders with a subset of overall software features and functionality.
- As each increment is produced, the software becomes more and more complete.
- The Process framework activities are complemented by a number of umbrella activities which helps to manage and control progress, quality, change, and risk.

Umbrella Activities

- Software engineering process framework activities are complemented by a number of umbrella activities.
- In general, umbrella activities are applied throughout a software project and help a software team manage and control progress, quality, change, and risk.

Typical umbrella activities include:

Software project tracking and control- allows the software team to assess progress against the project plan and take any necessary action to maintain the schedule.

Risk management- assesses risks that may affect the outcome of the project or the quality of the product.

Software quality assurance- defines and conducts the activities required to ensure software quality.

Technical reviews- assesses software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.

Measurement- defines and collects process, project, and product measures that assist the team in delivering software that meets stakeholders' needs; can be used in conjunction with all other framework and umbrella activities.

Software configuration management- manages the effects of change throughout the software process.

Reusability management- defines criteria for work product reuse (including software components) and establishes mechanisms to achieve reusable components.

Work product preparation and production- encompasses the activities required to create work products such as models, documents, logs, forms, and lists.

- **Prescriptive Process Models**

Prescriptive process models were originally proposed to bring order to the chaos of software development.

Traditional models have brought a certain amount of useful structure to software engineering work and have provided a reasonably effective road map for software teams. Following are prescriptive process models:

- Waterfall model
- Incremental Model
  - RAD Model
- Evolutionary Process Model
  - Prototype Model
  - Spiral Model
  - Concurrent Process Model

Waterfall Process Model

The waterfall model, sometimes called the classic life cycle, suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment, culminating in ongoing support of the completed software.

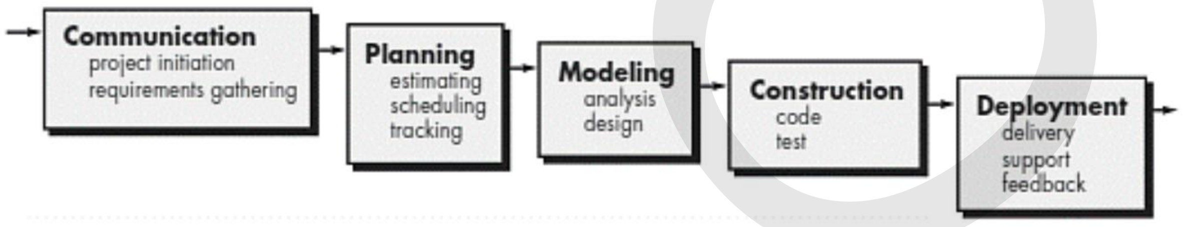


Fig. Waterfall Model

Limitations:

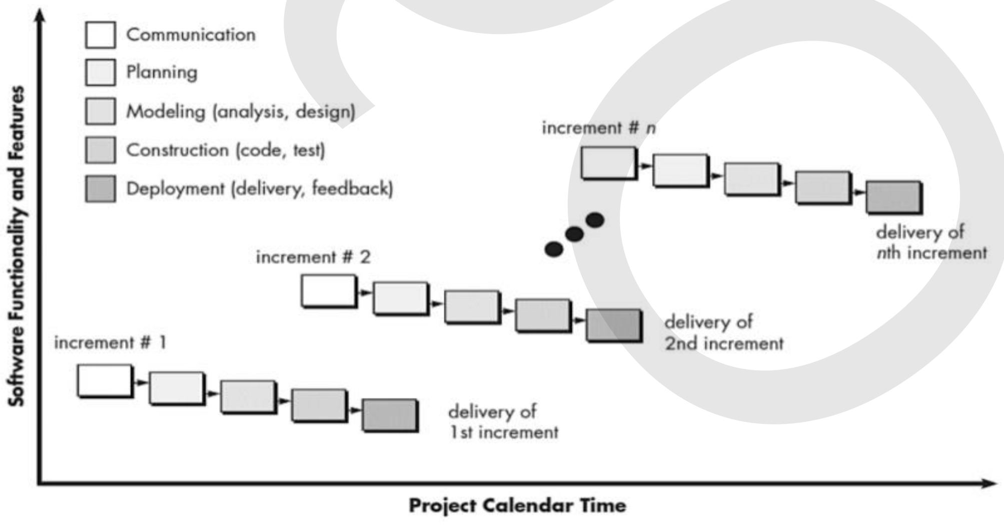
- The nature of the requirements will not change very much during development; during evolution.
- The model implies that you should attempt to complete a given stage before moving on to the next stage.
- Does not account for the fact that requirements constantly change.
- It also means that customers cannot use anything until the entire system is complete.
- The model implies that once the product is finished, everything else is maintenance.
- Surprises at the end are very expensive
- Some teams sit ideal for other teams to finish
- Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.

When to use waterfall model?

- Requirements are very well known, clear and fixed
- Product definition is stable
- Technology is understood
- There are no ambiguous requirements
- Ample resources with required expertise are available freely
- The project is short

Incremental Process Model

- The incremental model combines elements of linear and parallel process flows.
- The incremental model applies linear sequences in a staggered fashion as calendar time progresses.
- Each linear sequence produces deliverable "increments" of the software.
- For example, word-processing software developed using the incremental paradigm might deliver basic file management, editing, and document production functions in the first increment; more sophisticated editing and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advanced page layout capability in the fourth increment.



Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
Software Functionality and Features
increment # n
delivery of nth increment
increment # 2
delivery of 2nd increment
increment # 1
delivery of 1st increment
Project Calendar Time

Fig. Incremental Process Model

Advantages:

- Generates working software quickly and early during the software life cycle.
- This model is more flexible – less costly to change scope and requirements.
- It is easier to test and debug during a smaller iteration.
- In this model customer can respond to each built.
- Lowers initial delivery cost.
- Easier to manage risk because risky pieces are identified and handled during iteration.

Disadvantages:

- Needs good planning and design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than waterfall.

When to use Incremental Process Model?

- This model can be used when the requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some details can evolve with time.
- There is a need to get a product to the market early.
- A new technology is being used.
- Resources with needed skill set are not available.
- There are some high risk features and goals.

RAD (Rapid Application Development) Process Model:

- It is a type of incremental model. In RAD model the components or functions are developed in parallel as if they were mini projects.
- The developments are time boxed, delivered and then assembled into a working prototype.
- This can quickly give the customer something to see and use and to provide feedback regarding the delivery and their requirements.

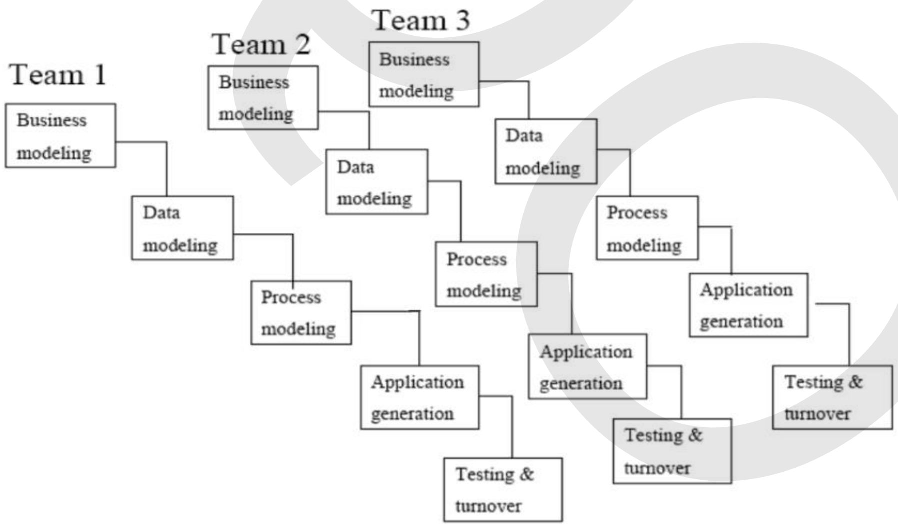


Fig. RAD Process Model

Advantages:

- Reduced development time.
- Increases reusability of components
- Quick initial reviews occur
- Encourages customer feedback
- Integration from very beginning solves a lot of integration issues.

Disadvantages:

- For large but scalable projects RAD requires sufficient human resources.
- Projects fail if developers and customers are not committed in a much shortened time-frame. ⍰ Problematic if system cannot be modularized.
- Not appropriate when technical risks are high (heavy use of new technology).
- When to use RAD model?
- RAD should be used when there is a need to create a system that can be modularized in 2-3 months oftime.
- It should be used if there’s high availability of designers for modeling and the budget is high enough toafford their cost along with the cost of automated code generating tools.
- RAD SDLC model should be chosen only if resources with high business knowledge are available andthere is a need to produce the system in a short span of time (2-3 months).

Prototype process model

- The basic idea here is that instead of freezing the requirements before a design or coding can proceed, athrowaway prototype is built to understand the requirements.
- This prototype is developed based on the currently known requirements.
- By using this prototype, the client can get an “actual feel” of the system, since the interactions withprototype can enable the client to better understand the requirements of the desired system.
- Prototyping is an attractive idea for complicated and large systems for which there is no manual processor existing system to help determining the requirements.
- The prototype are usually not complete systems and many of the details are not built in the prototype.The goal is to provide a system with overall functionality.

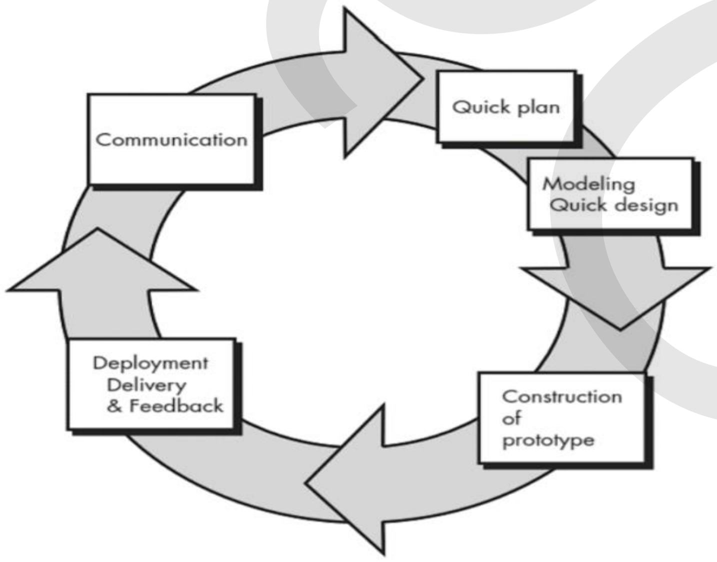


Fig. Prototype process model

Advantages:

- Users are actively involved in the development
- Since in this methodology a working model of the system is provided, the users get a better understanding of the system being developed.
- Errors can be detected much earlier.
- Quicker user feedback is available leading to better solutions. Missing functionality can be identified easily
- Confusing or difficult functions can be identified

Disadvantages:

- Leads to implementing and then repairing way of building systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- Incomplete application may cause application not to be used as the full system was designed.

When to use Prototype Model?

- Prototype model should be used when the desired system needs to have a lot of interaction with the endusers.
- Typically, online systems, web interfaces have a very high amount of interaction with end users, are bestsuited for Prototype model. It might take a while for a system to be built that allows ease of use and needs minimal training for the end user.
- Prototyping ensures that the end users constantly work with the system and provide a feedback which isincorporated in the prototype to result in a useable system. They are excellent for designing good human computer interface systems.

Spiral Process Model:

- It was originally proposed by Barry Boehm, the spiral model is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of thewaterfall model.

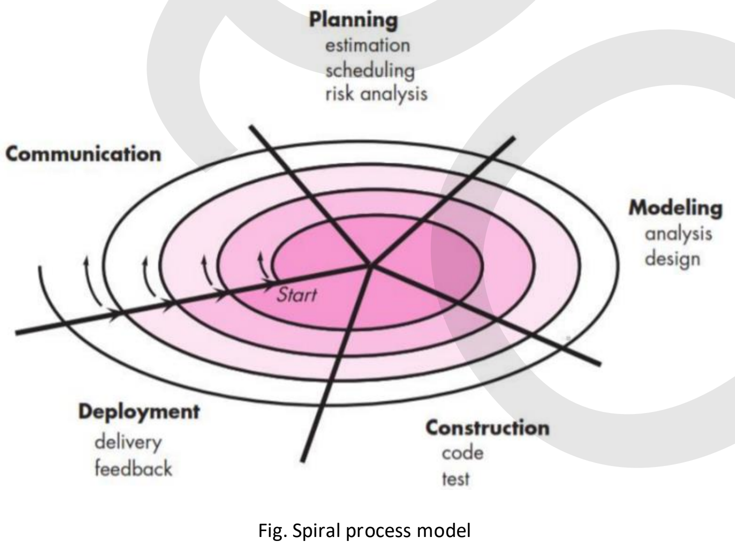


Fig. Spiral process model

- It provides the potential for rapid development of increasingly more complete versions of the software.

- 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 distinguishing features. One is a cyclic approach for incrementally growing a system'sdegree of definition and implementation while decreasing its degree of risk.
- The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible andmutually satisfactory system solutions.
- The first circuit around the spiral might result in the development of a product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively moresophisticated versions of the software.
- Each pass through the planning region results in adjustments to the project plan.
- Cost and schedule are adjusted based on feedback derived from the customer after delivery.
- In addition, the project manager adjusts the planned number of iterations required to complete thesoftware.

Advantages:

- High amount of risk analysis hence, avoidance of Risk is enhanced. Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the software life cycle.

Disadvantages:

- 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.

When to use spiral process model?

- When costs and risk evaluation is important for medium to high-risk projects.
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs.
- Requirements are complex
- New product line Significant changes are expected (research and exploration)

Concurrent Process Model:

- The concurrent development model, sometimes called concurrent engineering, allows a software teamto represent iterative and concurrent elements of any of the process models.
- For example, the modelling activity defined for the spiral model is accomplished by invoking one or moreof the following software engineering actions: prototyping, analysis, and design.
- The activity modelling may be in any one of the states noted at any given time.
- Similarly, other activities, actions, or tasks (e.g., communication or construction) can be represented inan analogous manner.
- All software engineering activities exist concurrently but reside in different states.

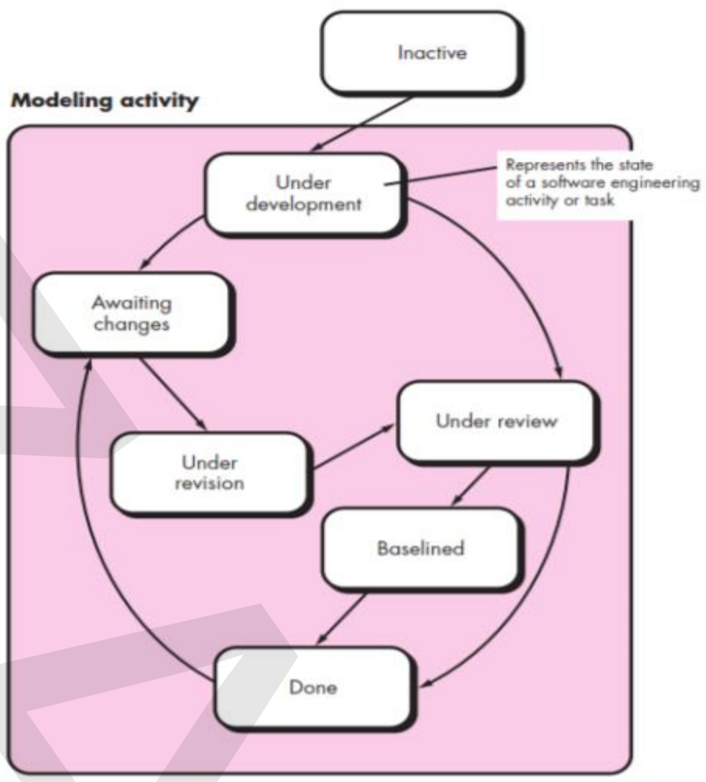


Fig. Concurrent Process Model

- **The Unified Process :**

Booch, Jacobson, and Rumbaugh later developed the The unified process, which is a framework for object-oriented software engineering using UML which draws on the best features and characteristics of conventional software process models Emphasizes the important role of software architecture Consists of a process flow that is iterative and incremental, thereby providing an evolutionary feel Consists of five phases: inception, elaboration, construction, transition, and production.

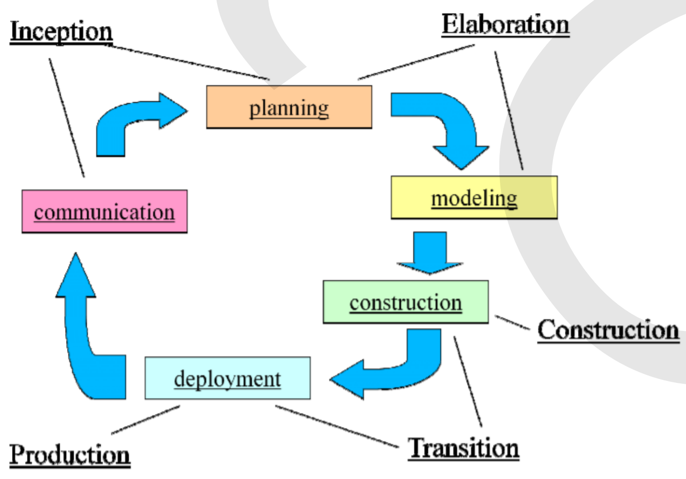


Fig.**Unified Process**

Inception Phase

- Encompasses both customer communication and planning activities of the generic process.
- Business requirements for the software are identified
- A rough architecture for the system is proposed
- A plan is created for an incremental, iterative development
- Fundamental business requirements are described through preliminary use cases-
    - A use case describes a sequence of actions that are performed by a user

Construction Phase

- Encompasses the construction activity of the generic process
- Uses the architectural model from the elaboration phase as input
- Develops or acquires the software components that make each use-case operational
- Analysis and design models from the previous phase are completed to reflect the final version of the increment
- Use cases are used to derive a set of acceptance tests that are executed prior to the next phase

Transition Phase

- Encompasses the last part of the construction activity and the first part of the deployment activity of the generic process
- Software is given to end users for beta testing and user feedback reports on defects and necessary changes
- The software teams create necessary support documentation (user manuals, troubleshooting guides, installation procedures)
- At the conclusion of this phase, the software increment becomes a usable software release

Production Phase

- Encompasses the last part of the deployment activity of the generic process
- On-going use of the software is monitored
- Support for the operating environment (infrastructure) is provided
- Defect reports and requests for changes are submitted and evaluated

Chapter 1

A. MCQ's

1. Which of the following is not a characteristics of a software
   a. Probabilistic b. Deterministic
   c. Versions are possible d. Customised
2. Which of the following is not considered in four software engineering layers
   a. Quality b. Tools
   c. Methods d. Cost
3. In a waterfall model, the process is structured as a cascade of phases done is _______ fashion.
   a. Concurrent b. Circular
   c. Linear d. Parallel
4. In evolutional process model, limited version of software is introduced which is called a ________.
   a. Final product b. Core product
   c. Sample product d. Dummy product
5. During software development process using spiral model_______ is considered as each revolution is made.
   a. Cost b. Quality
   c. Adaptability d. Risk

Ans.

1.a 2.d 3.c 4.b 5.d

B. True or False:

1. Software is customized.
2. The primary goal of software engineering is to increase the cost if the software
3. The waterfall model is the oldest paradigm for software engineering
4. The spiral model non-realistic approach to the development of large scale systems and software
5. More iteration means more efforts in spiral model.

Ans.

1. True 2. False 3. True 4. False 5. True

C. 1 Mark questions.

1. Give any two advantages of incremental model
2. What do you mean by stakeholder?
3. Explain use of prototype model
4. What is meant by concurrent development model?
5. What does the radius of the spiral indicate in the spiral model?

D. Explain in brief

1. Explain waterfall Model
2. Give advantages of Waterfall model
3. Explain Spiral Model
4. Explain advantages of Incremental Model

5. Explain characteristics of software
6. Explain Framework activities of process
7. What are the goals of software engineering
8. Explain Umbrella Activities