**Software Engineering: A layered technology**

Software engineering is a *layered technology* as referred in fig shown below. It consists of four layers as

- A quality focus
- Process
- Methods
- Tools

**A quality Focus**
The bedrock that supports software engineering is a quality focus. Software engineering approach must rest on the organizational commitment to quality. Total Quality Management (TQM) philosophies that support continuous improvement culture leads to the development of increasingly more approaches to software engineering.

**Process Layer**
It is the foundation of software engineering that defines a framework for a set of *KPA* (*key process areas*) that must be established for effective delivery of software engineering technology. The KPA form the basis for management control of software projects and establish the context in which technical methods are applied, work products (models, documents, data, reports, forms etc) are produced, milestones are established, quality is ensured, and change is properly managed.

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

**Tools**
It provide automated or semi-automated support for the process and methods. Integrated tools establish computer-aided software engineering (CASE). CASE combines software, hardware and software engineering database to create a software engineering environment analogous to CAD/CAE (computer aided design/engineering) for hardware.

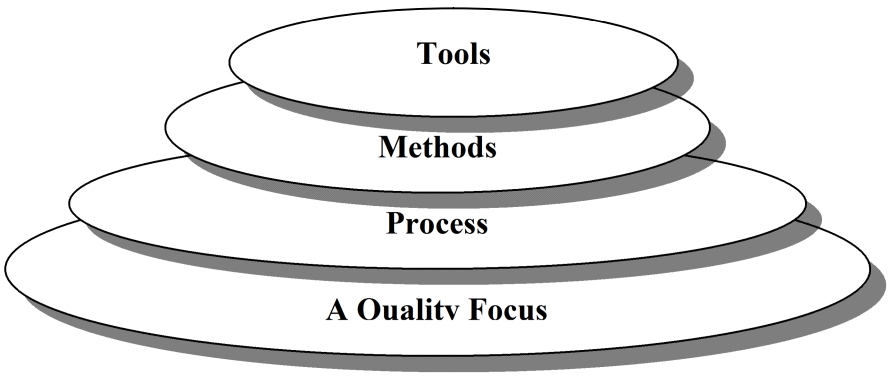
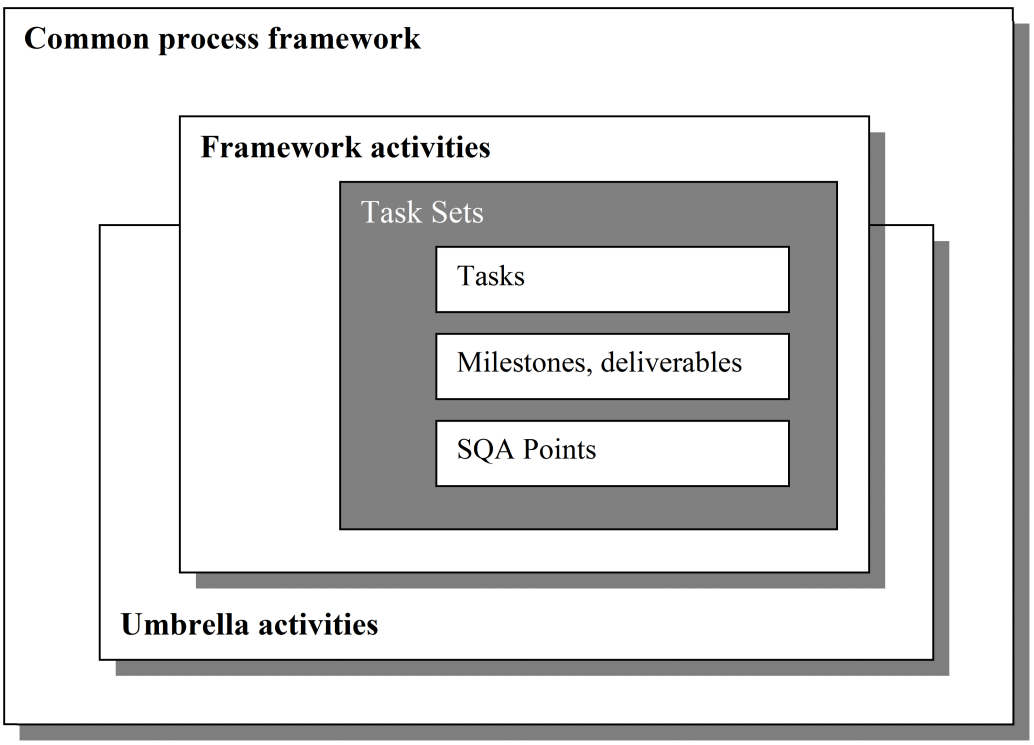



Fig: Software Engineering Layer

**Software Process**

- The roadmap to building high quality software products is software process.
- Software processes are adapted to meet the needs of software engineers and managers as they undertake the development of a software product.
- A software process provides a framework for managing activities that can very easily get out of control.
- Different projects require different software processes.
- The software engineer's work products (programs, documentation, data) are produced as consequences of the activities defined by the software process.
- The best indicators of how well a software process has worked are the *quality*, *timeliness*, and long-*term viability* of the resulting software product.
- **A structured set of activities whose goal is the development or evolution of software**
- Generic activities in all software processes are:
    - Specification - what the system should do and its development constraints
    - Design/Development - production of the software system
    - Validation - checking that the software is what the customer wants
    - Evolution - changing the software in response to changing demands

**Common Process Framework**




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.

A number of task sets includes:

- Software engineering work tasks

- Project milestones
- Work products
- Quality assurance points

These task sets enable the framework activities to be adopted to the characteristics of the software project and the requirements of the project team.

**SEI-CMM ( Capability Maturity Model)**

To determine an organization's current state of process maturity, the SEI uses an assessment that results in a five point grading scheme to determines compliance with a **CMM** ( capability Maturity Model)

SEI approach provides a measures of the global effectiveness of a company's software engineering practices and establishes five process maturity levels. CMM defines key activities required at different levels of process maturity.

| | |
|---|---|
| Level 1 : **Initial** | ☑ Ad hoc and occasionally even chaotic software processes.<br>☑ Few processes are defined, and success depends on individual effort. |
| Level 2 : **Repeatable** | ☑ Able to repeat earlier successes<br>☑ Establish basic project management to track cost, schedule & functionality |
| Level 3 : **Defined** | ☑ Management and engineering processes documented, standardized, and integrated into organization-wide software process |
| Level 4 : **Managed** | ☑ software process and products are quantitatively understood and controlled using detailed measures |
| Level 5 : **Optimizing** | ☑ Continuous process improvement is enabled by quantitative feedback from the process and testing innovative ideas & technologies. |

**Key Process Areas (KPA)**

Each maturity level is associated with KPA. KPA describe those software Engineering functions (e.g. software project planning, requirement management) that must be present to satisfy good practice at a particular level.
Characteristics of KPA includes:

- Goals : the overall objectives that the KPA must achieve.
- Commitments : requirements that must be met to achieve goals
- Abilities : those things( organizationally & technically) to meet the commitments
- Activities : specific tasks required to achieve KPA function.

- Monitoring : manner in which the activities are monitored.
- Verification : manner in which proper practice for the KPA can be verified.

18 KPAs are defined across the maturity model & mapped into different levels of process maturity:

**Process maturity level 2**

Software configuration management
Software quality assurance
Software subcontract management
Software project planning
Requirements management

**Process maturity level 3**

Peer review
Intergroup co-ordination
Software product engineering
Integrated software management
Training program
Organization process definition
Organization process focus

**Process maturity level 4**

Software quality management
Quantitative process management

**Process maturity level 5**

Process change management
Technology change management
Defect prevention

**Software Process Models**

A software process model or *software engineering paradigm* is a development strategy that encompasses the process, methods and tools layers.

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

All software development process can be characterized as a **problem solving loop** as shown in figure that consists of four distinct stages such as:

- Status Quo : represents the current state of affairs.
- Problem definition : identifies the specific problem to be solved.

- Technical development : Solve the problem through the application of some technology

- Solution Integration : delivers the results (e.g., document, programs, data, new business functions, new products)

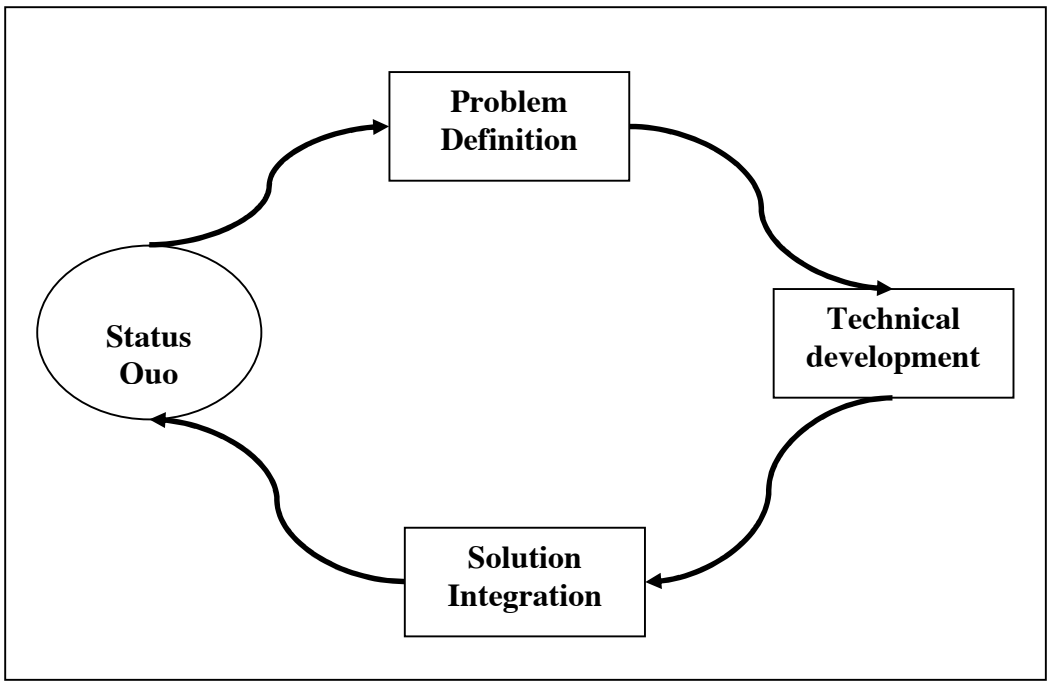



Fig : Problem solving loop

**Types of Process Models**

A process model is chosen based on the nature of the project and application, the methods and tools being used, and the controls and deliverables that are required.

The following types of process models are being used in the software development:

- **Linear Sequential Model**
  - old fashioned but reasonable approach when requirements are well understood
  - Separate and distinct phases of specification and development

- **Prototyping Model**
  - good first step when customer has a legitimate need, but is clueless about the details, developer needs to resist pressure to extend a rough prototype into a production product.

- **Rapid Application and Development (RAD) Model**
  - makes heavy use of reusable software components with an extremely short development cycle

- **Evolutionary Software process Model**
    - **Incremental Model**
        - delivers software in small but usable pieces, each piece builds on pieces already delivered
    - **Spiral Model**
        - couples iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model
- **Component-Based Development**
    - spiral model variation in which applications are built from prepackaged software components called classes

**1. Linear Sequential Model**

- This model is also known as the Waterfall model or software life cycle.
- The principal stages of the model map onto fundamental development activities:
    - **Requirements analysis and definition**
        - system's services, constraints and goals are established.
        - details are served as system specifications.
    - **System and software design**
        - partitions the requirements to either hardware or software systems.
        - establishes an overall system architecture.
        - identifies and describes the fundamental software system abstractions and their relationships
    - **Implementation and unit testing**
        - software design is realized as a set of programs or programs units.
        - Unit test is performed to verify each unit meet its specification.
    - **Integration and system testing**
        - Individual program units or programs are integrated.
        - Perform test to ensure that the requirements are met.
    - **Operation and maintenance**
        - It is the longest life cycle phase.
        - The system is installed and put into practical use.
        - Maintenance involve error correction and improving the implementation of system units
        - Enhance the system service to meet new requirements.
- The drawback of the waterfall model is the difficulty of accommodating change after the process is underway

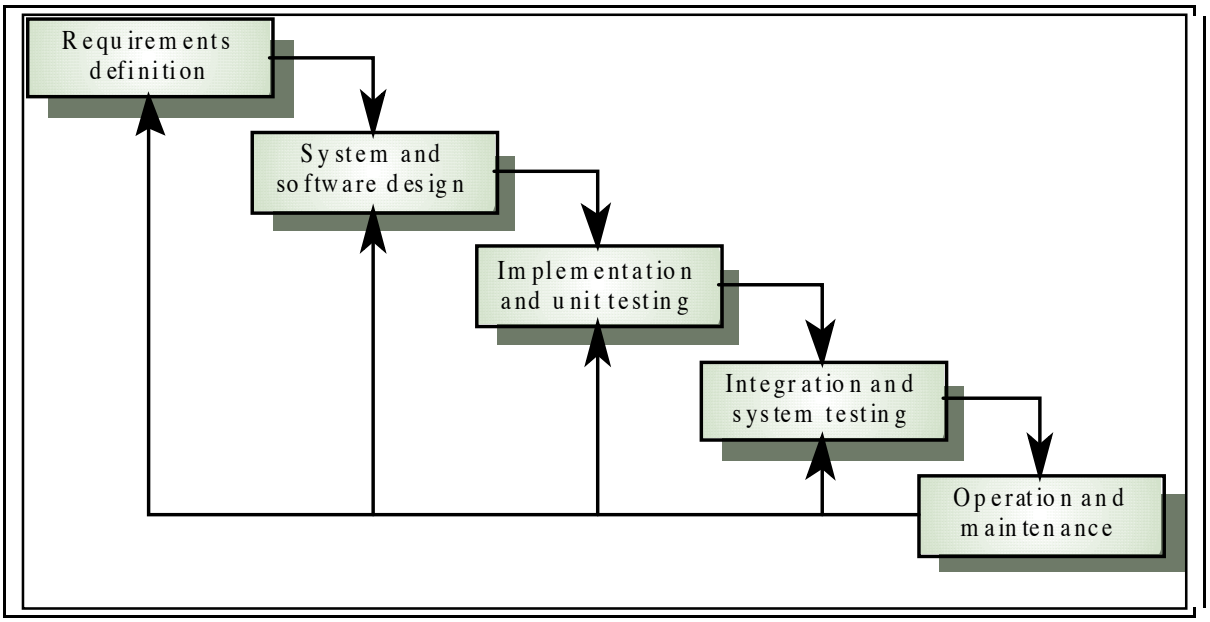



Fig: Waterfall Model

In principle, the result of each phase is one or more approved ("signed off") documents. The following phase should not start until the previous phase has finished. In practice, these stages overlap and feed information to each other. During design, problems with requirements are identified, during coding design problems are found and so on. The software process is not a simple linear model but involves a sequence of iterations of the development activities.

Because of the costs of producing and approved documents, iterations are costly and involve significant rework. Therefore, after a small number of iterations, it is normal to freeze parts of the development, such as the specification, and to continue with the later development stages. Problems are left for later resolution, ignored or are programmed around. This premature freezing of requirements may means that the system won't do what the user wants. It may also lead to badly structured systems.

**Waterfall Model problems**

- Inflexible partitioning of the project into distinct stages
- This makes it difficult to respond to changing customer requirements
- Therefore, this model is only appropriate when the requirements are well-understood

**2. Prototyping Model**

- A prototyping paradigm offer the best approach when.....
    - a customer defines a set of general activities for software but does not identify detailed input, processing, or output requirements.
    - the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system

- It begins with requirement gathering where developer and customer, together, define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory.

- Develop a " quick Design " that focus on input approaches and output formats that is visible to the customer/user. It leads to the construction of a prototype, that serves as a mechanism for identifying software requirements

- Prototype is evaluated by the customer/user and refine the requirements for the software to be developed.

- The prototype serves as " the first system" where users get a feel for the actual system and developers get to build something immediately.

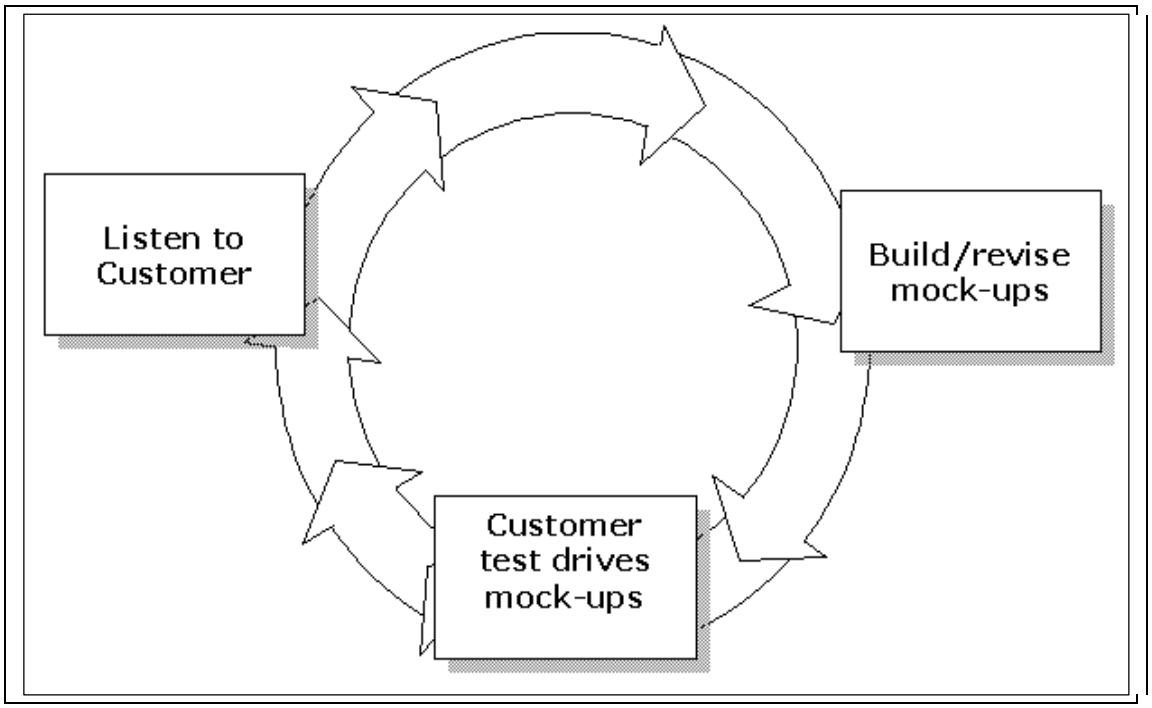



Fig: The prototyping paradigm

In this model, the key is to define the rules of the game at the beginning, that is, the customer and developer must agree that the prototype is built to serve as a mechanism for defining requirements. It is then discarded and the actual software is engineered considering quality and maintainability.

**The prototype Model problem**

- Prototype appears to be the working version of the software, but all software quality and long-term maintainability has not been considered. The final must be rebuilt.
- Implementation comprises for quick working of prototype.

- Use of inappropriate Operating system or programming language
- Implementation of inefficient algorithm to demonstrate capability.

**3. The RAD Model**

RAD ( Rapid application development) is an incremental software development process model with extremely short development cycle.

It is a "*high-speed*" adaptation of the linear sequential model in which rapid development by using component-based construction.

RAD process enable to create a "*fully functional system*" within very short time periods ( e.g. 60-90 days) if requirements are well understood and project scope is constrained.

Used primarily for information systems applications, the RAD approach encompasses the following phases:

- **Business modeling**
  - The information flow among the business functions is modeled to answer the following questions:
    - What information drives the business process?
    - What information is generated?
    - Who generates it?
    - Where does the information go?
    - Who process it?

- **Data modeling**
  - The information defined in the business-modeling phase is refined into a set of data objects that are needed to support the business.
  - The characteristics (attributes) of each object are identified and the relationship between these objects is defined.

- **Process modeling**
  - The data objects defined in the data-modeling phase are transformed to achieve the information flow necessary to implement a business function.
  - Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.

- **Application generation**
  - Use of forth generation techniques
  - Reuse existing program components or create reusable components.
  - Automated tools are used to facilitate construction of the software.

- **Testing and turnover**
  - Since RAD process emphasizes reuse, many of the program components have already been tested, that reduces overall testing time.

- Testing of new components and interfaces are done.

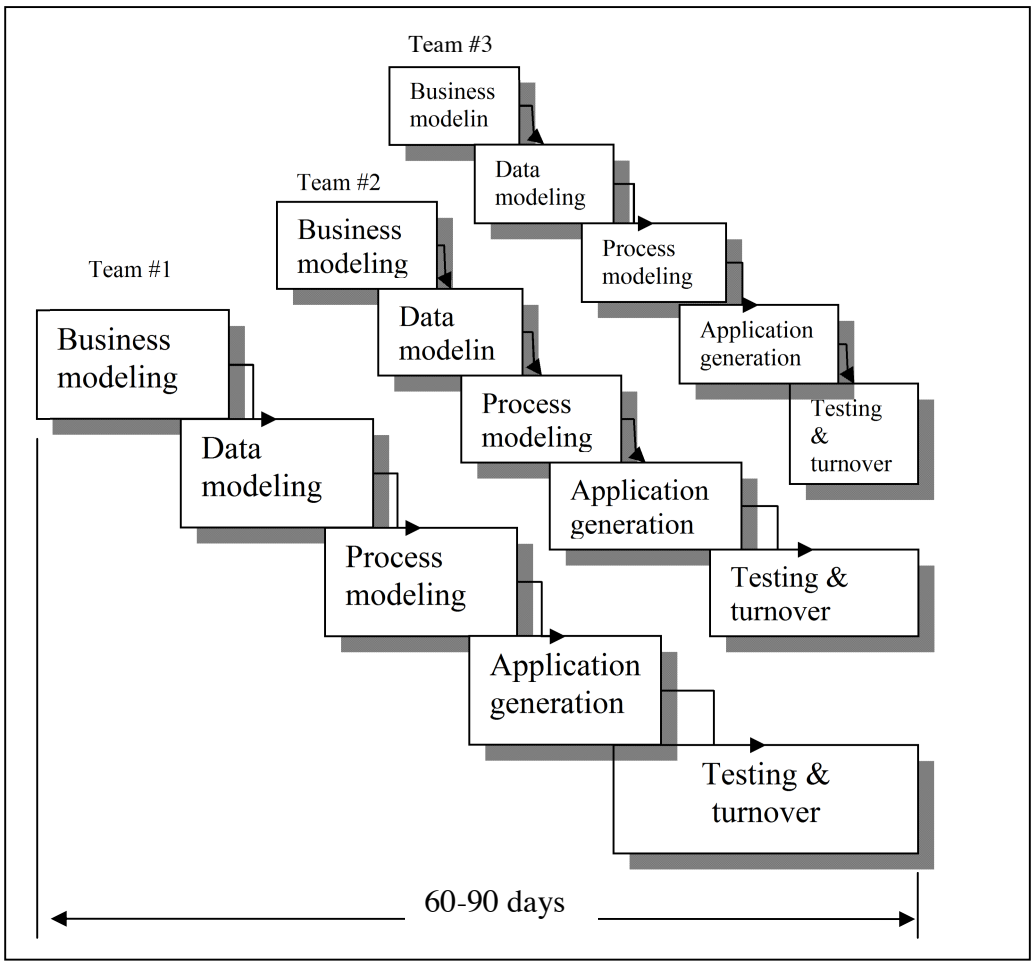



Fig: The RAD Model

**Drawback of RAD Model**

- For large and scalable projects, RAD requires sufficient human resources to create the right number of RAD teams.
- RAD requires developers and customers who are committed to the *rapid-fire* activities necessary to get a system complete in a much-abbreviated time frame. In the lack of commitment from either side, RAD project fails.
- If a system can not be properly modularized, building the components necessary for RAD will be problematic.
- RAD approach can not be used for high performance.
- RAD is not appropriate when technical risks are high.

**4. Evolutionary Software Process Models**

- The software system ALWAYS evolves over a period of time in the course of a project (due to change in the business and products requirements) so process iteration where earlier stages are reworked is always part of the process for large systems.

The linear sequential model is designed for straight for straight line requirement. In essence, this waterfall approach assumes that a complete system will be delivered after the linear sequence is completed. The prototyping model is designed to assist the customer ( or developer) in understanding requirements. In general, it is not designed to deliver a production system. The evolutionary nature of software is not considered in either case of these classic software engineering paradigms.

- Iteration can be applied to any of the generic process models
- Two (related) approaches
    - Incremental development model
    - Spiral model

### 4.1 Incremental Model

- It combines the elements of the linear sequential model with the iterative philosophy of prototyping.
- It applies the linear sequences in the staggered fashion as calendar time progresses.
- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality
- User requirements are prioritised and the highest priority requirements are included in early increments
- Each linear sequence produces a deliverable "increment" of the software, where the process flow for any increment can incorporate the prototyping paradigm.
- The first increment results in a *core product* where the basic requirements are addressed, but many supplementary features remains undelivered.
- After the deployment of the core product and its evaluation, a plan is developed for the next increment, which addresses the modification of the core product to better meet the needs of the customer and delivery of additional features and functionality. This process is repeated following the delivery of each increment, unit the complete product is produced.

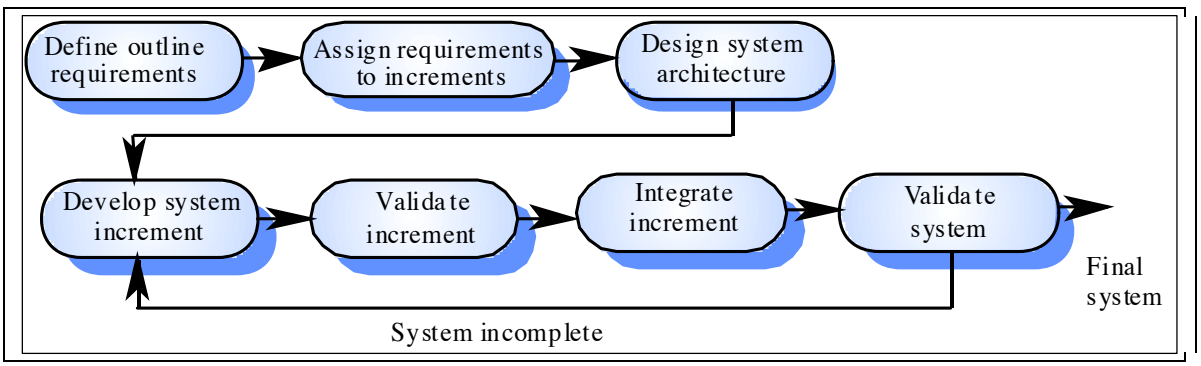



Fig: Incremental development

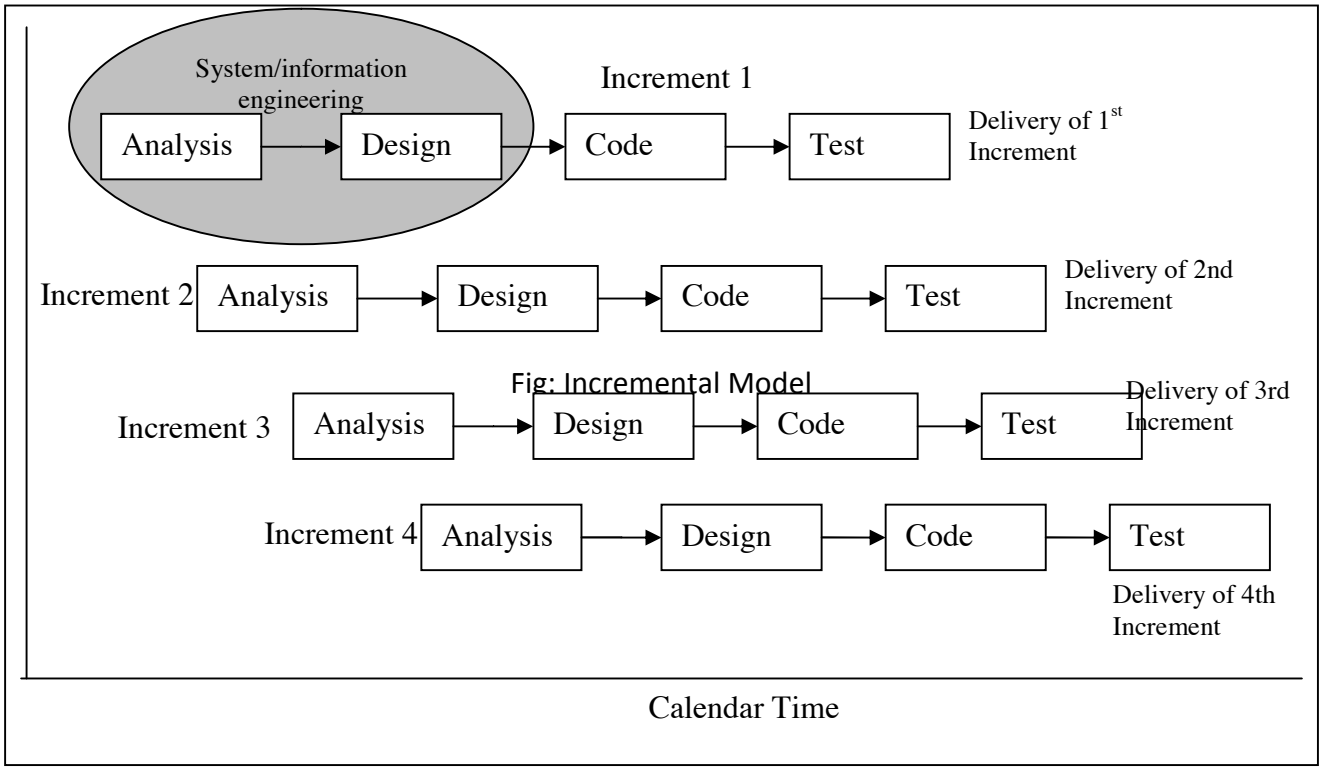



Fig: Incremental Model

**Incremental development advantages**

- Customer value can be delivered with each increment so system functionality is available earlier
- Early increments act as a prototype to help elicit requirements for later increments
- Lower risk of overall project failure
- The highest priority system services tend to receive the most testing

**4.2 Spiral Model**

- As proposed by Boehm, it is a risk-driven approach to software development that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model. i.e. it uses prototyping as a risk reduction mechanism and retains the systematic step-wise approach of the waterfall model.
- Software is developed in a series of incremental releases.
- Process is represented as a spiral rather than as a sequence of activities with backtracking
- Each loop in the spiral represents a phase in the process.
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required
- Risks are explicitly assessed and resolved throughout the process.
- Model is divided into a number of framework activities, called ***task regions*** that contains:
    - Customer communication: task required to establish effective communication between developer and customer.
    - Planning: task required to define resources, timeliness, and other project related information.

- Risk analysis: task required to assess both technical and management risk.
- Engineering: task 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)

**Spiral model of the software process**

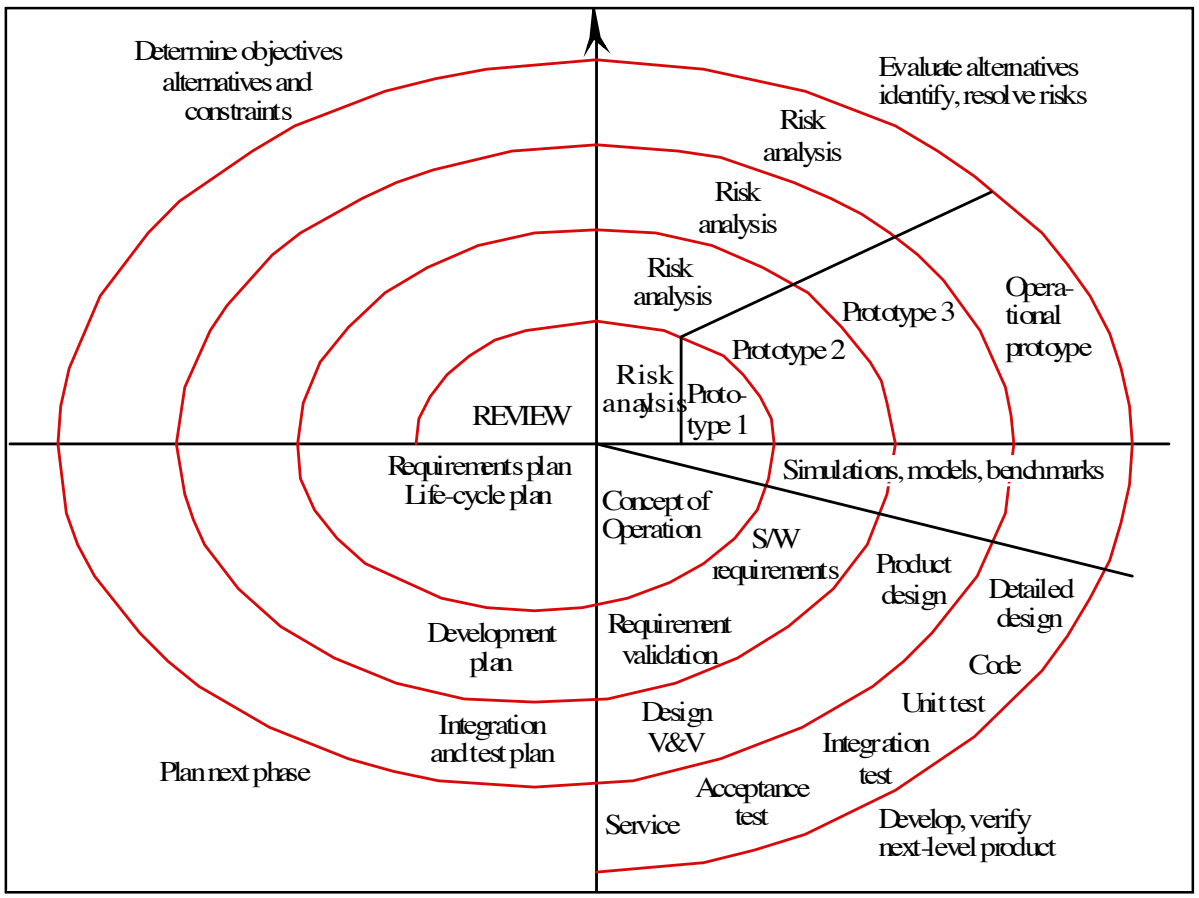



Fig: Boehm Spiral Model

**Spiral model sectors**

- **Objective setting**
  - Specific objectives for the phase are identified; alternative solutions and its constraints are defined.
  -
- **Risk assessment and reduction**
  - Alternative solutions are evaluated and the potential project risks are assessed and activities put in place to reduce the key risks by developing an appropriate prototype

- **Development and validation**
  - A development model for the system is chosen which can be any of the generic models
- **Planning**
  - The project is reviewed and the next phase of the spiral is planned

**Spiral Model as Meta Model**

Spiral Model can be viewed as a *Meta model* since it encompasses all the previously discussed models, and uses prototyping as a risk reduction mechanism. For example, a single loop spiral actually represents the waterfall model. The spiral model uses an evolutionary approach and iterations along the spiral can be considered as evolutionary levels through which the complete system is built. The spiral model enables the developer to understand and react to the risks at each evolutionary level (i.e. iteration along the spiral). The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks.

**Advantages of Spiral Model**

1. It could be used as effectively for system enhancement as for system development.
2. Most life cycle models can be considered as a special case of the spiral models, and
3. The embedded risk analysis built into the model avoids many of the difficulties that arise in other models.

**5. Component-based development (CBD) model**

Object-oriented technologies provide the technical framework for a component-based process model.
It emphasizes the creation of classes that encapsulate both data and the algorithms used to manipulate the data.
Properly designed and implemented object-oriented classes are reusable across different applications and computer-based system architectures.
It incorporates the characteristics of the spiral model and evolutionary in nature, demanding an iterative approach to the creation of software.
It composes applications from prepackaged software components (called classes)
It performs the following activities:

*Identification of candidate classes* which is accomplished by examining the data to be manipulated by the applications and the algorithms that will be applied to accomplish the manipulation.
*Search for the class library* from its library or repository of classes created earlier.
If available, *extract class components* for reuse, otherwise, develop using object-oriented methods.
Compose the first iteration of the application to be build, using the extracted classes from the library and any new classes built to meet the unique needs of the application.
Process flow returns to the spiral and re-enter the component assembly iteration during subsequent passes.

**Advantages of the CBD Model**

- Reusability of the software.
- Reduction in the development cycle time
- Reduction in the project costs.
- Increase in the productivity

**Comparison of different Life Cycle Models**

The classical waterfall model with feedback is the most widely used software development model evolved so far. However, the iterative waterfall model is suitable only for well-understood problems. Further, the waterfall model is too simplistic. There are no milestones in the model, no mention of the documents generated, or the reviews conducted, and no indication of the quality assurance activities. Establishing milestones, review points, standardized documents, and management sign-offs improve product visibility. Without sufficient product visibility, it becomes very difficult to manage a software development project. If we cannot see something, we cannot hope to manage it.

The prototype model is suitable for projects whose user requirements or the underlying technical aspects are not well understood.

The evolutionary approach is suitable for large problems, which can be decomposed into a set of modules that can be implemented incrementally, and where incremental delivery of the system is acceptable to the customer.

The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks.

**Summary**

- Software engineering is a discipline that integrates process, methods, and tools for the development of computer software.
- Software processes are the activities involved in producing and evolving a software system. They are represented in a software process model
- General activities are specification, design and implementation, validation and evolution
- Generic process models describe the organisation of software processes
- Iterative process models describe the software process as a cycle of activities
- Requirements engineering is the process of developing a software specification
- Design and implementation processes transform the specification to an executable program
- Validation involves checking that the system meets to its specification and user needs
- Evolution is concerned with modifying the system after it is in use.
- Spiral model uses prototyping as a risk reduction mechanism and retains the systematic step-wise approach of the waterfall model.