UNIT I SOFTWARE PROCESS

Introduction to Software Engineering- A Generic view of process: - A layered technology, a process framework Activities- S/W Engineering Paradigm .

Software Life Cycle Models : **Prescriptive Process Models** :The waterfall model, Incremental process models, Evolutionary process models .**Specialized process models** : Component based development, The Formal Methods Model.

INTRODUCTION

- **SOFTWARE**

  Computer programs and associated documentation are referred to as software. Software product may be developed for a particular customer or may be developed for a general market.

- **SOFTWARE PRODUCT**

  Software product may be developed for a particular customer or may be developed for a general market.

  There are two fundamental types of software product:

  1. ***Generic products***: These are stand-alone systems that are produced by development organisation and sold to any customer who is able to buy them.
     Examples: software for PCs such as databases, word processors, drawing packages.
  2. ***Customised (or bespoke) products***: These are systems which are developed for a particular customer. A software contractor develops the software especially for that customer.
     Examples: control systems for electronic devices, air traffic control systems.

- **SOFTWARE CHARACTERISTICS**

  Software development is a logical activity and therefore it is important to understand basic characteristics of software. Some important characteristics of software are

  1. Software is developed or engineered, it is not manufactured.
     Software development and hardware manufacture are 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.
  2. Software doesn't "wear out".

In fig 1the "bathtub curve", indicates that hardware exhibits relatively high failure rates early in its life defects are corrected and the failure rate drops to a steady-state level for some period of time. As time passes, however, the failure rate rises again as hardware components suffer from the environmental maladies like dust, vibration, abuse, temperature extremes, and many. Stated simply, ***the hardware begins to wear out.***

Software is not susceptible to the environmental maladies that cause hardware to wear out. Hence ideally it should have an "idealized curve" shown in Figure 2.But due to undiscovered errors the failure rate is high and drops down as soon as the errors are corrected. Hence failure rating of software the "actual curve" is shown below. However, the implication is clear, ***software doesn't wear out.***

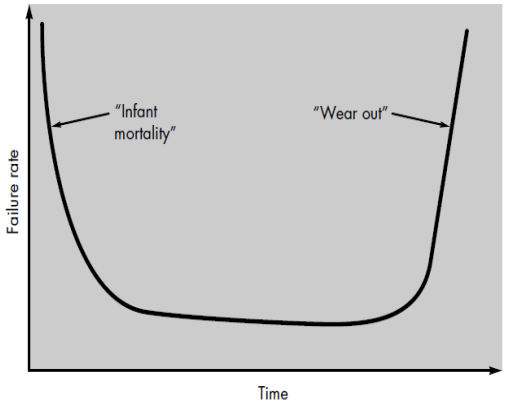


Fig 1: Failure curve for hardware

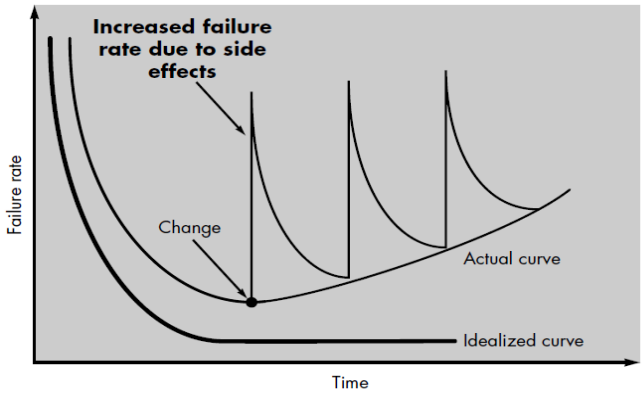


Fig 2: Idealized and actual failure curves for software

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

   While developing any hardware product, first the circuit design with desired functioning properties is created. Then requires the hardware component such as ICs, registers are assembled according to the design, but this is not done while developing software product. Most of the software is custom built.

   Software component should be designed and implemented so that it can be reused in many different programs. It is practiced to reuse algorithms and data structures. Today software industry is trying to make library of reusable components.

   For eg., today's graphical user interfaces are built using reusable components such as graphics windows, pull-down menus, and many more such components.

- **TYPES OF SOFTWARE / THE CHANGING NATURE OF SOFTWARE / VARIOUS CATEGORIES OF SOFTWARE**

1. System software

- ✓ It is a collection of programs written to service other programs. Some system software programs are compilers, editors, and file management utilities.
- ✓ The purpose of the system software is to establish a communication with the hardware.

2. **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
     - a monitoring component that coordinates all other components so that real-time response can be maintained.

3. **Business software**
   - ✓ Business information processing is the largest single software application area. 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.
   - ✓ Business software applications also encompass interactive computing (e.g., point of-sale transaction processing).

4. **Engineering and scientific software**
   - ✓ This software category has applications ranging from astronomy to volcanology, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing.
   - ✓ This software is based on the complex numeric computations.

5. **Embedded software.**
   - ✓ 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 functions (e.g., keypad control for a microwave oven)
   - ✓ Provides significant function and control capability (e.g., digital functions in an automobile such as fuel control, dashboard displays, and braking systems).

6. **Personal computer software**
   - ✓ Applications are word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, personal and business financial applications, external network, and database access.

7. **Web-based software**
   - ✓ The Web pages retrieved by a browser are software that includes executable instructions (e.g., CGI, HTML, Perl, or Java), and data (e.g., hypertext and a variety of visual and audio formats.

8. **Artificial intelligence software**

- ✓ Artificial intelligence (AI) software makes use of non-numerical algorithms to solve complex problems.
- ✓ Expert systems, also called knowledge based systems, pattern recognition (image and voice), artificial neural networks, theorem proving, and game playing are the applications within this category.

- **SOFTWARE ENGINEERING**

Software engineering is an engineering discipline that is concerned with all aspects of software production.

There are two key phrases:

1. ***Engineering discipline*** Engineers apply theories, methods and tools which are appropriate and always try to discover solutions to problems.

   Engineers must think of the organisational and financial constraints, so they look for solutions within these constraints.
2. ***All aspects of software production*** Software engineering is also concerned with activities such as software project management and with the development of tools, methods and theories to support software production.

The **IEEE** has developed a more comprehensive definition, it states:

Software Engineering: The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

- **Difference between software engineering and computer science?**
  - ✓ Computer science is concerned with theory and fundamentals;
  - ✓ Software engineering is concerned with the practicalities of developing and delivering useful software.
- **Difference between software engineering and system engineering?**
  - ✓ System engineering is concerned with all aspects of computer-based systems development, including hardware, software and process engineering.
  - ✓ Software engineering is part of this process.

- **SOFTWARE ENGINEERING: A LAYERED TECHNOLOGY**

Software engineering is a layered technology. Any software can be developed using these layered approaches. Various layers on which the technology is based are quality focus layer, process layer, methods layer, tools layer.

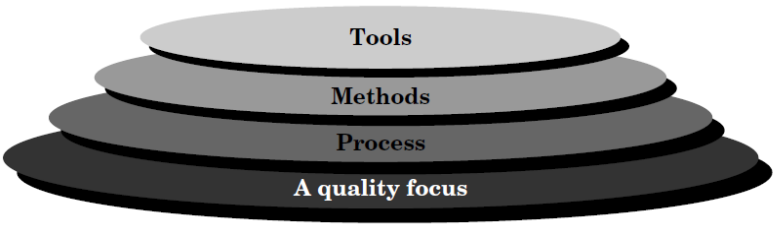


Fig 3: Software engineering Layers

- ✓ The bedrock that supports software engineering is a *quality focus*.
- ✓ The foundation for software engineering is the *process* layer. Software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software.
- ✓ 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 *tools* provide automated or semi-automated support for the process and the methods.

- **SOFTWARE PROCESS**

A software process is the set of activities and associated results that produce software product.

Four fundamental process activities are:

.

1. ***Software specification*** - where customers and engineers define the software to be produced and the constraints on its operation
2. ***Software development*** - where the software is designed and programmed.
3. ***Software validation*** - where the software is checked to ensure that it is what the customer requires.
4. ***Software evolution*** - where the software is modified to adapt it to changing customer and market requirements.

❖ ***A PROCESS FRAMEWORK:***

The process framework is required for representing the common process activities.

The software process is characterized by process framework activities, task set and umbrella activities.

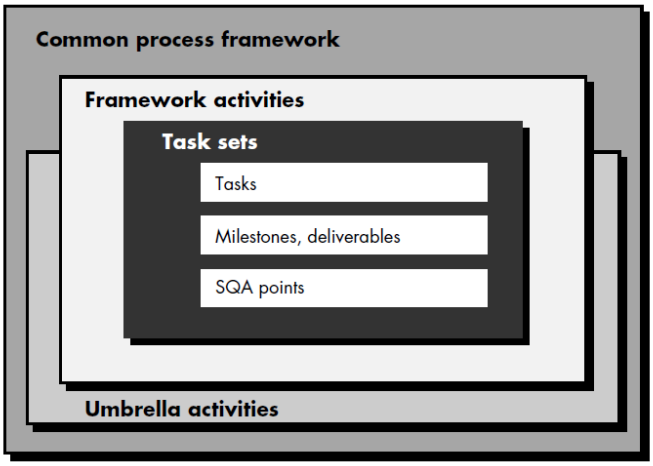


Fig 4: Software process framework

- ✓ A number of ***task sets***—each a collection of software engineering work tasks, project milestones, work products, and quality assurance points.

- ✓ This enables ***the framework activities*** to be adapted to the characteristics of the software project and the requirements of the project team.

-The Generic process framework activities are

1. Communication
   The customer requirement information gathering is done by communication.
2. Planning
   The planning activity defines the engineering work plan, describes technical risks, list resource requirements, work products produced and defines work schedule.
3. Modeling
   The modeling activity defines the requirement analysis and design.
4. Construction
   The construction activity implements the corresponding coding and testing.
5. Deployment
   The software delivered for customer evaluation and feedback is obtained.

- ✓ 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.

-The umbrella activities are

1. Software project tracking and control
This activity helps to assess the software team progress and take corrective action to maintain schedule.

2.Formal technical reviews
This activity helps to analyze the engineering work products to uncover and clear errors before they proceed to next activity.

3.Software quality assurance
This activity maintain the required software quality.

4.Software configuration management
This activity manages the configuration process when any change in the software process.

5. Document preparation and production
This activity defines the model, document forms and lists to be carried out.

6.Reusability management
This activity defines the work product reuse.

7.Measurement

This activity defines the project and product measure to assist the software team in delivery of the required software.

8.Risk management

This activity analyzes the risk that may affect the project quality.

❖ ***CAPABILITY MATURITY MODEL (CMM):***

Defines key activities required at different levels of process maturity.

Five process maturity levels,

Level 1: Initial

Few processes are defined, and success depends on individual effort.

Level 2: Repeatable

This level helps to keep track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

Level 3: Defined

The software process for both management and engineering activities is documented, standardized, and integrated into an organization wide software process.

All projects use a documented and approved version of the organization's process for developing and supporting software. This level includes all characteristics defined for level 2.

Level 4: Managed

Detailed measures of the software process and product quality are collected.

Both the software process and products are quantitatively understood and controlled using detailed measures. This level includes all characteristics defined for level 3.

Level 5: Optimizing

Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technologies. This level includes all characteristics defined for level 4.

2.SOFTWARE ENGINEERING PARADIGM

- ✓ Software process model is a software development.
- ✓ Software paradigm or process model is an abstract representation of a process.
- ✓ A process model for software engineering is chosen based on the nature of the project and application, the methods and tools to be used, and the controls and deliverables that are required.

All software development can be characterized as a problem solving loops (Fig 5) in which four distinct stages are encountered: Existing status, problem definition, technical development, and solution integration.

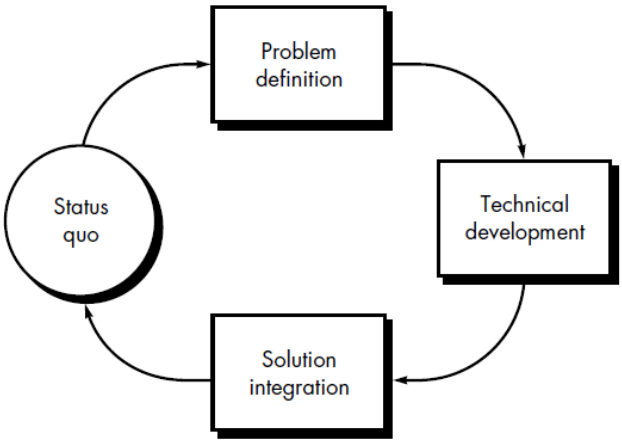


Fig 5: Problem Solving Loop

- ✓ **Status quo:** "represents the current state of affairs".
- ✓ **Problem definition:** identifies the specific problem to be solved.
- ✓ **Technical development:** solves the problem through the application of some technology.
- ✓ **Solution integration:** delivers the results (e.g., documents, programs, data) to those who requested the solution in the first place.

The problem solving loop is very difficult to implement in software development process because we can't strictly categorize the development in these phase.

There may be a requirement of crosstalk within and across stages. Hence various software process models are suggested depending upon nature of the software.

LIFE CYCLE MODELS / SOFTWARE PROCESS MODELS

A software process is a set of activities that leads to the production of a software product. Although there are many software processes, some fundamental activities are common to all software processes:

1. *Software specification:* The functionality of the software and constraints on its operation must be defined.
2. *Software design and implementation:* The software to meet the specification must be produced.
3. *Software validation:* The software must be validated to ensure that it does what the customer wants.
4. *Software evolution:* The software must evolve to meet changing customer needs.

Various process models are:

1. Water fall model

2. Incremental process model
    a) Incremental model
    b) RAD model
3. Evolutionary process model
    a) Prototyping
    b) Spiral model
    c) Concurrent development model
4. Specialized process model
    a) Component based development
    b) The Formal methods model

1.Water fall model:

The waterfall model is also called as 'linear-sequential model' or 'classic life cycle model'. It is the oldest software paradigm. This model suggests the systematic, sequential approach to software development. Various phases of water fall model are:

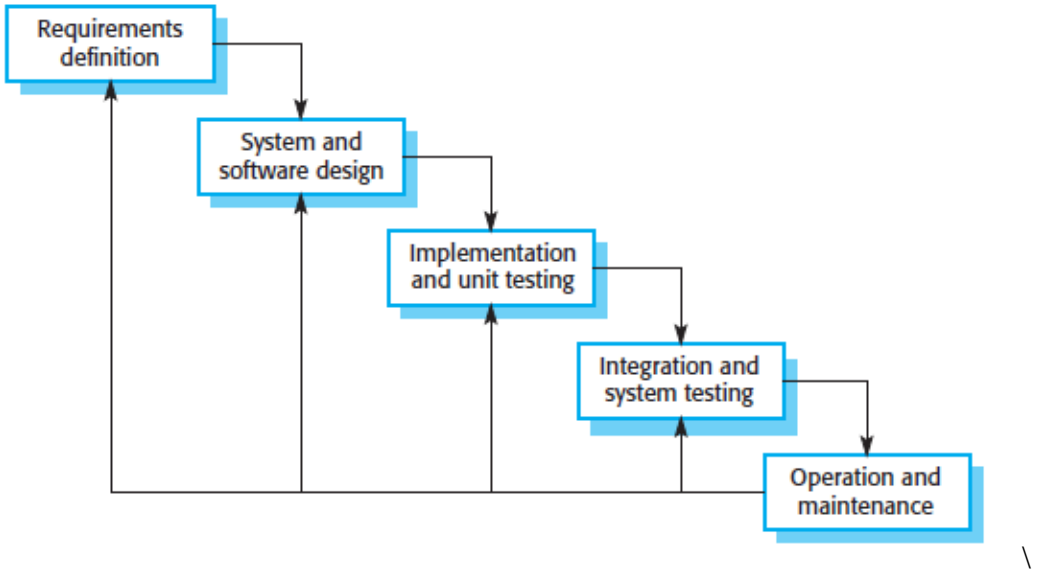
a) Requirements analysis and definition:

The system's services, constraints and goals are established by consultation with system users. They are then defined in detail and serve as a system specification.

b) System and software design:

The systems design process partitions the requirements to either hardware or software systems. It establishes the overall system architecture.

Software design involves identifying and describing the fundamental software system abstractions and their relationships.



c) Implementation and unit testing:

During this stage, the software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.

d) ***Integration and system testing*****:**

The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.

e) ***Operation and maintenance:***

Normally this is the longest life-cycle phase. The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units and enhancing the system's services as new requirements are discovered.

The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. One phase has to be complete before moving onto the next phase.

Advantages:

1. Easy to understand and implement.
2. Documentation available at the end of each phase.
3. Widely used and known

Disadvantages:

1. Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
2. Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
3. Few business systems have stable requirements.
4. The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites**.**

2. Incremental process model

a) Incremental model

The *incremental model* combines elements of the linear sequential model with the iterative philosophy of prototyping.

Referring to Fig 6, the incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces a deliverable "increment" of the software

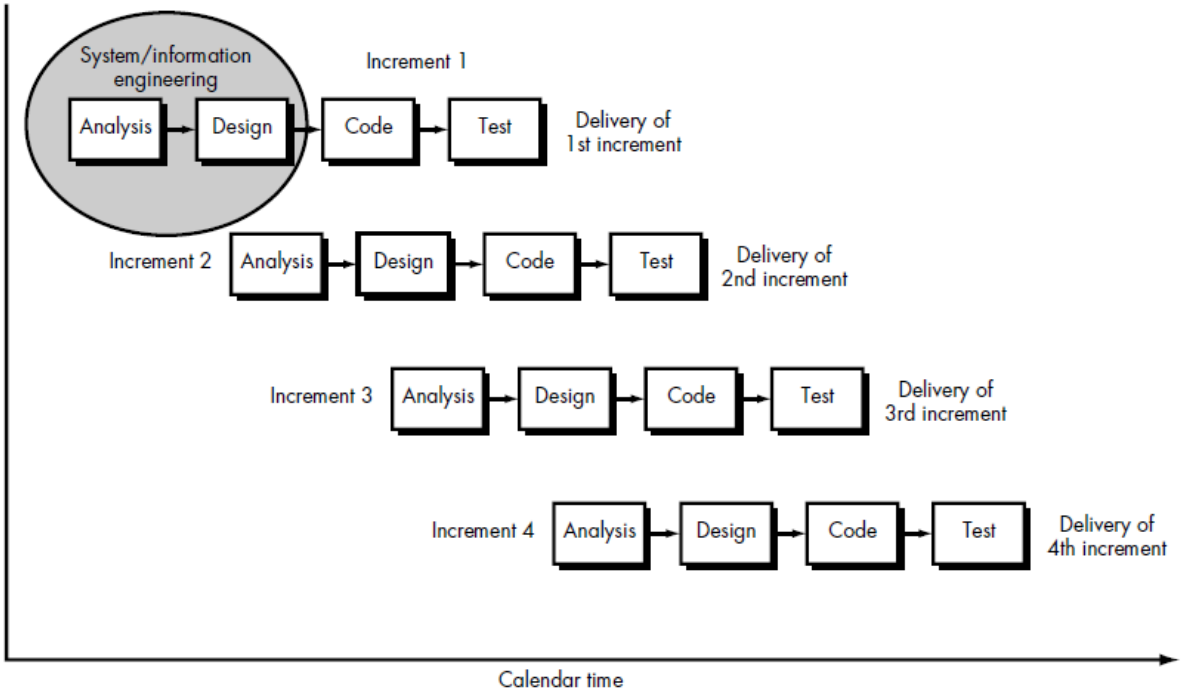


Fig 6: The Incremental model

Eg: word-processing software developed using the incremental method developed as follows

1.Basic file management, editing, and document production functions in the first increment;
2.More sophisticated editing and document production capabilities in the second increment;
3. Spelling and grammar checking in the third increment;
4. Advanced page layout capability in the fourth increment.

- ✓ When an incremental model is used, the first increment is often a *core product* and delivered to customer use and successively a plan is made for next increment to enhance its features and functionality.
- ✓ The process is repeated following the delivery of each increment, until the complete product is produced.
- ✓ The incremental process model, like prototyping and other evolutionary approaches, is iterative in nature. But unlike prototyping, the incremental model focuses on the delivery of an operational product with each increment.
- ✓ Early increments are stripped down versions of the final product, but they do provide capability that serves the user and also provide a platform for evaluation by the user.
- ✓ Incremental development is particularly useful when staffing is unavailable for a complete implementation by the business deadline that has been established for the project.
- ✓ Early increments can be implemented with fewer people. If the core product is well received, then additional staff (if required) can be added to implement the next increment.
- ✓ In addition, increments can be planned to manage technical risks. It might be possible to plan early increments in a way that avoids the use of this hardware, thereby enabling partial functionality to be delivered to end-users without inordinate delay.

Advantages :

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

b)RAD Model

- ✓ Rapid application development (RAD) is an incremental software development process model.
- ✓ The RAD model is a "high-speed" adaptation of the linear sequential model in which rapid development is achieved by using component-based construction.
- ✓ If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a "fully functional system" within very short time periods (e.g., 60 to 90 days).

The RAD approach encompasses the following phases

1. Business modeling.
2. Data modeling.
3. Process modeling.
4. Application generation
5. Testing and turnover.

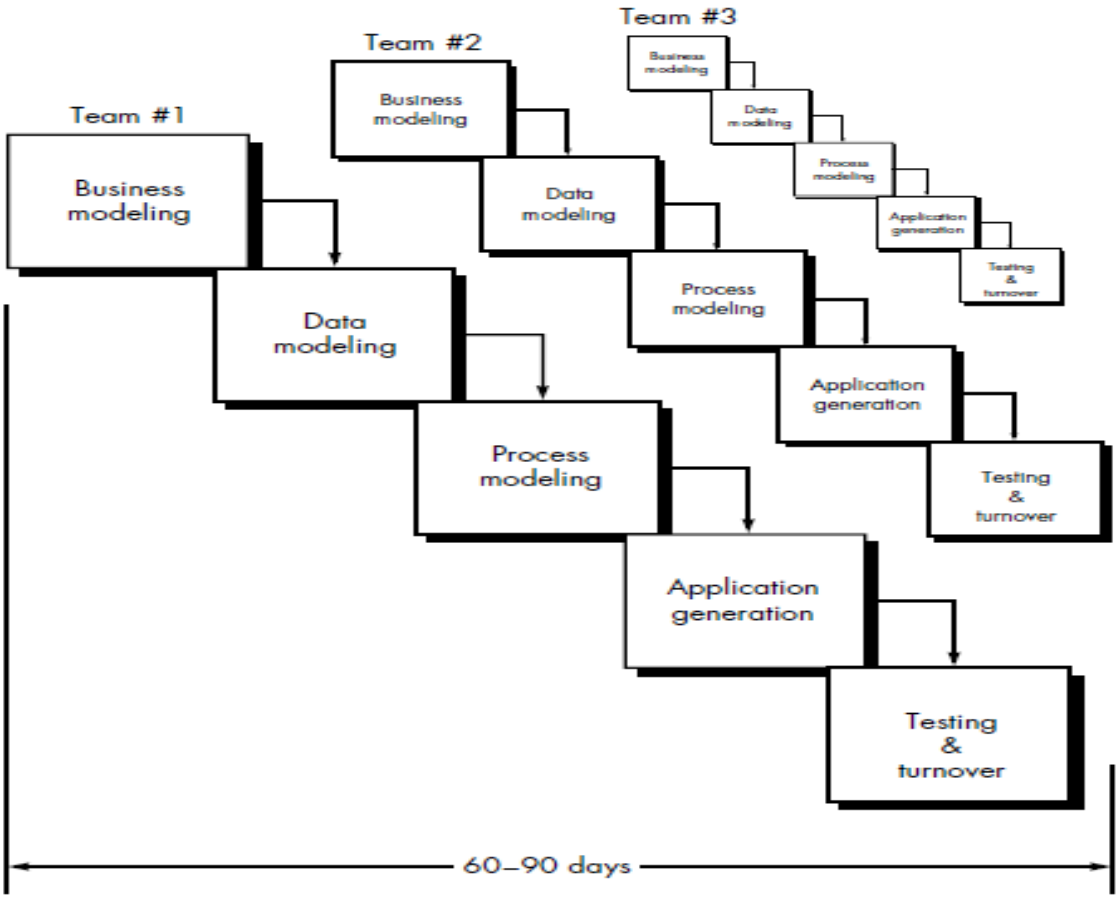


Fig 7: The RAD Model

Business modeling

- ✓ The information flow among business functions is modeled in a way that answers the following questions:
  - ➢ What information drives the business process?
  - ➢ What information is generated?
  - ➢ Who generates it?
  - ➢ Where does the information go?
  - ➢ Who processes it?

Data modeling

- ✓ The information flow defined as part of the business modeling phase is refined into a set of data objects that are needed to support the business.
- ✓ The characteristics (called *attributes*) of each object are identified and the relationships between these objects defined.

Process modeling

- ✓ 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

- ✓ RAD assumes the use of fourth generation techniques.
- ✓ Rather than creating software using conventional third generation programming languages the RAD process works to reuse existing program components (when possible) or create reusable components (when necessary).

Testing and turnover

- ✓ Since the RAD process emphasizes reuse, many of the program components have already been tested.
- ✓ This reduces overall testing time. However, new components must be tested and all interfaces must be fully exercised.

Advantages

1. Extremely short development time.
2. Uses component-based construction and emphasizes reuse and code generation.

Disadvantages

1. Large human resource requirements (to create all of the teams).
2. Requires strong commitment between developers and customers for "rapid-fire" activities.
3. High performance requirements maybe can't be met (requires tuning the components).
4. RAD is not appropriate when technical risks are high.

3.Evolutionary process model

Evolutionary models are iterative. They are characterized in a manner that enables software engineers to develop increasingly more complete versions of the software.

a)Prototyping

A customer defines a set of general objectives for software but does not identify detailed input, processing, or output requirements.

- ✓ The prototyping paradigm (Figure 8) begins with **requirements gathering**.
- ✓ Developer and customer meet and define the overall objectives for the software, identify areas needed more requirement gathering.
- ✓ A "**quick design**" then occurs. The quick design focuses on a representation of those aspects of the software that will be *visible to the customer/user* (e.g., input and output formats).
- ✓ The quick design leads to the **construction of a prototype**.
- ✓ Customer/user **evaluates** the prototype in order to **refine** requirements.
- ✓ Iteration occurs as the prototype is tuned to satisfy the needs of the customer.
- ✓ The prototype serves as a mechanism for identifying software requirements.

- ✓ If a working prototype is built, the developer uses the existing program fragments or applies tools (e.g., report generators, window managers) that enable working programs to be generated quickly.

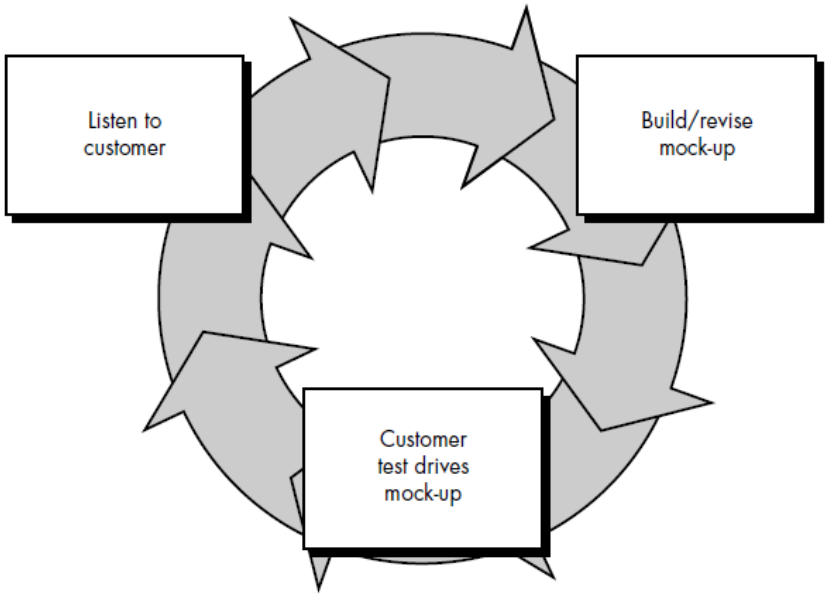


Fig 8: The Prototyping Paradigm

Prototyping can also be problematic for the following reasons:

1. The customer sees what appears to be a working version of the software product, so software development management is necessary.

2. The developer often makes implementation quickly. So results in the usage of inefficient algorithms.

b)The Spiral Model

- The spiral model, originally proposed by Boehm, is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model.
- The spiral development model is a risk driven process model.
- 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 9 depicts a spiral model that contains six task regions:
    1. Customer Communication
    2. Planning
    3. Risk analysis
    4. Risk engineering
    5. Construction and release
    6. Customer evaluation

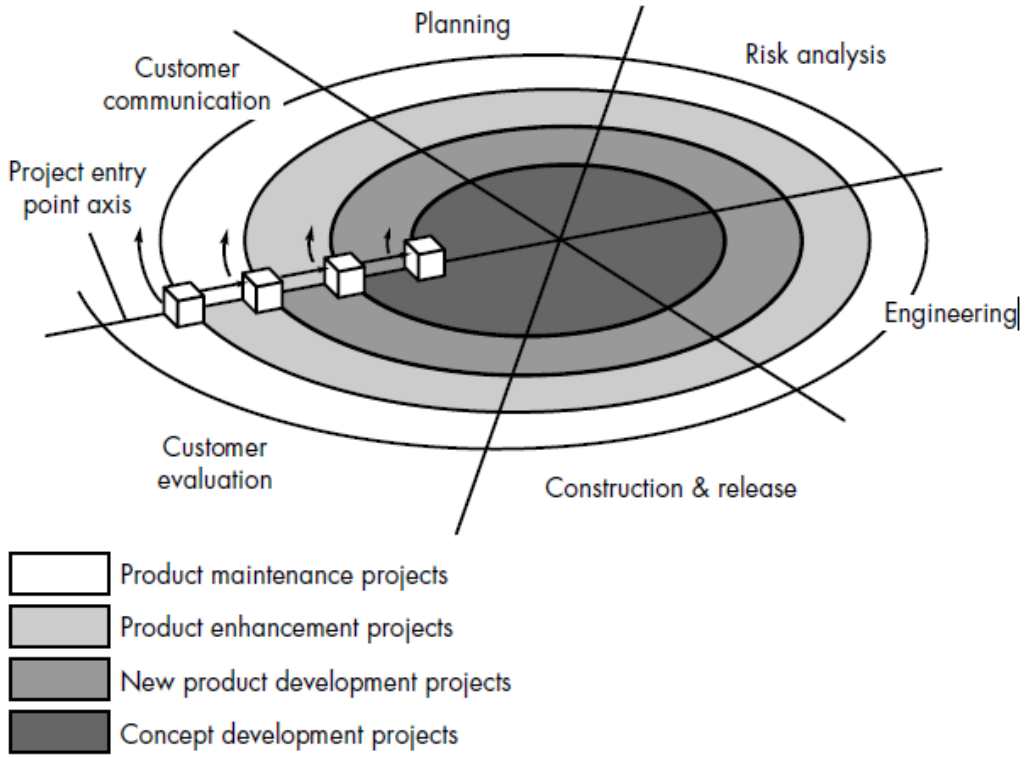


Fig 9: The Typical Spiral Model

1.Customer communication

-Tasks required to establish effective communication between developer and customer.

2.Planning

- The tasks required to define resources, timelines, and other project related information.

3.Risk analysis

- Tasks required to assess both technical and management risks..

4.Risks engineering

- Tasks required to build one or more representations of the application.

5.Construction & release

- Tasks required to construct, test, install and provide user support (e.g documentation and training) .

6.Customer evaluation

- Customer feedback collected every stage.

- Based on evaluation of the software representations created during the engineering stage and implemented during the installation stage.

As this evolutionary process begins, the software engineering team moves around the spiral in a **clockwise direction**, beginning at the center.

- 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 more sophisticated 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 customer evaluation.

An alternative view of the spiral model can be considered by examining the ***project entry point axis,*** also shown in Figure 9.

- Each cube placed along the axis can be used to represent the starting point for different types of projects.
- A "**concept development project**" starts at the core of the spiral and will continue until concept development is complete.
- If the concept is to be developed into an actual product, the process proceeds through the next cube (new product development project entry point) and a "**new development project**" is initiated.
- Whenever a change is initiated, the process starts at the appropriate entry point (e.g., **product enhancement**).

Advantages:
1. High amount of risk analysis.
2. Good for large and mission-critical projects.
3. Software is produced early in the software life cycle.
4. Reusability of the software.

Disadvantages:
1. Can be a costly model to use.
2. Risk analysis requires highly specific expertise.
3. Project's success is highly dependent on the risk analysis phase.
4. Doesn't work well for smaller projects.

c)Concurrent Development Model

- The concurrent development model, sometimes called *concurrent engineering*
- The concurrent process model can be represented schematically as a series of major technical activities, tasks, and their associated states.

- For eg, the engineering activity defined for the spiral model (Fig 9) is accomplished by invoking the following tasks: prototyping and/or analysis modeling, requirements specification, and design

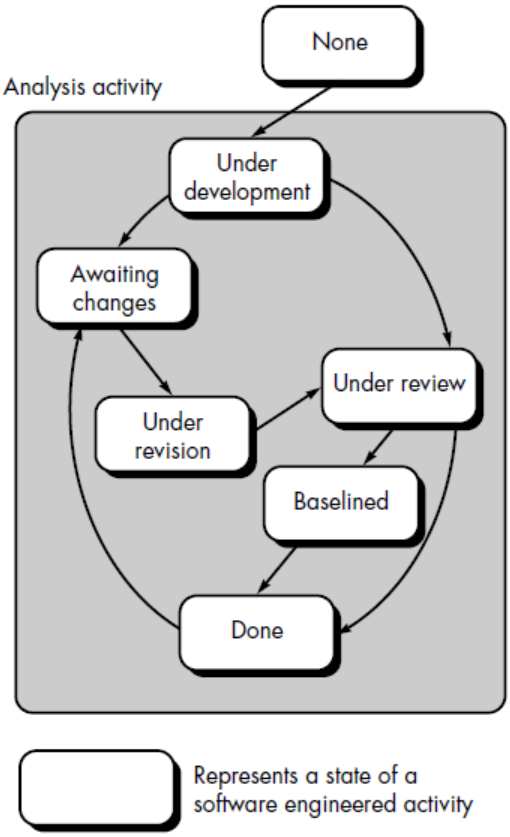


Fig 10: One Element of The Concurrent Process Model

- Fig 10 provides a schematic representation of one activity with the concurrent process model. The activity—analysis—may be in any one of the states noted at any given time. All activities exist concurrently but reside in different states.

- For eg., early in a project the *customer communication* activity has completed its first iteration and exists in the **awaiting changes** state.

- The *analysis* activity which existed in the **none** state while initial customer communication was completed now makes a transition into the **under development** state.

- If the customer indicates that changes in requirements must be made, the *analysis* activity moves from the **under development** state into the **awaiting changes** state.

The concurrent process model defines a series of events that will trigger transitions from state to state for each of the software engineering activities.

Specialized process models

Specialized process models take on many of the characteristics of one or more of the conventional models presented in the following.

a) Component based development

- The component-based development (CBD) model (Fig 11) incorporates many of the characteristics of the spiral model.
- It is evolutionary in nature, demanding an iterative approach to the creation of software.
- However, the component-based development model composes applications from prepackaged software components (called *classes*).
- The engineering activity begins with the identification of candidate classes. This is accomplished by examining the data to be manipulated by the application and the algorithms that will be applied to accomplish the manipulation. Corresponding data and algorithms are packaged into a class.

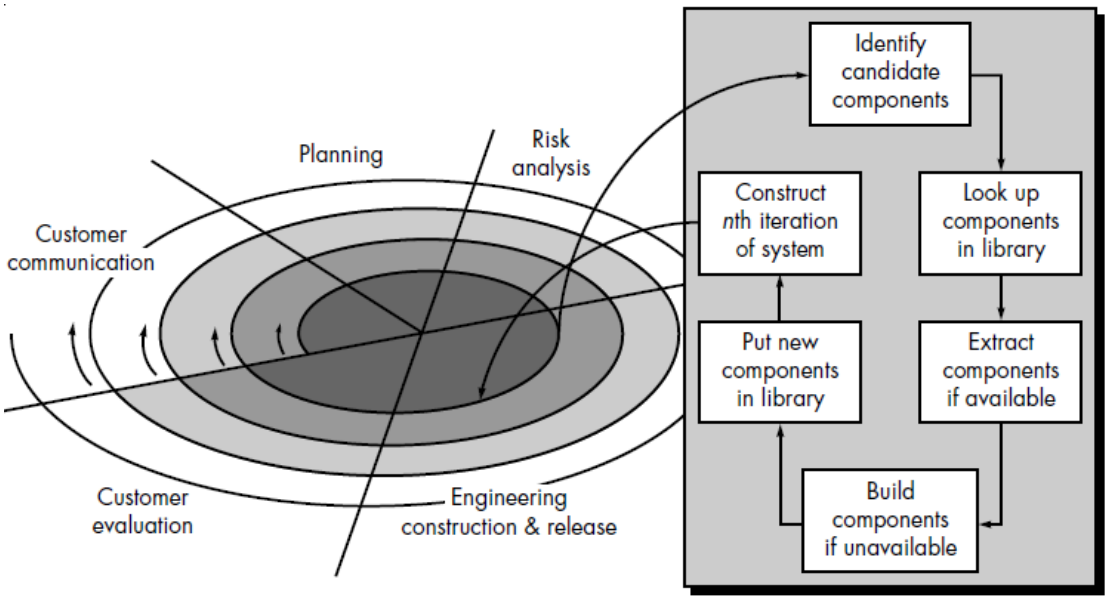


Fig 11: Component based Development

- Classes created in past software engineering projects are stored in a class library or repository.
- Once candidate classes are identified, the class library is searched to determine if these classes already exist.
- If they do, they are extracted from the library and reused. If a candidate class does not reside in the library, it is engineered using object-oriented methods.

- The first iteration of the application to be built is then composed, using classes extracted from the library and any new classes built to meet the unique needs of the application.
- Process flow then returns to the spiral and will ultimately re-enter the component assembly iteration during subsequent passes through the engineering activity.
- The component-based development model leads to software reuse and reusability provides software engineers with a number of measurable benefits.

b)The Formal Methods Model

- The formal methods model encompasses a set of activities that leads to formal mathematical specification of computer software.
- 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 *cleanroom 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 go undetected.

  The formal methods model offers the promise of defect-free software.

Applicability in a business environment:

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