TEEGALA KRISHNA REDDY ENGINEERING COLLEGE
(UGC-Autonomous)
Approved by AICTE, Affiliated by JNTUH, Accreditated by NAAC- 'A' Grade
Medbowli, Meerpet, Balapur, Hyderabad, Telangana- 500097
Mob: 8498085218. principal@tkrec.ac.in, ace@tkrec.ac.in

Software Engineering
(22AI301PC)

YEAR & SEM	**:**	**B. Tech II YEAR I SEM**
SUBJECT	**:**	**SOFTWARE ENGINEERING**
REGULATION	**:**	**R22**
DEPORTMENT & SEC	**:**	**AIML - C**
NAME OF THE FACULTY	**:**	**Mr. N NARESH**

UNIT- I

Introduction to Software Engineering

The Evolving Role of Software

Definition of Software

Software is a set of items or objects that form a "configuration" that includes

- Programs
- Documents
- Data

Software's Dual Role

- Software is a product
  - Delivers computing potential
  - Produces, manages, acquires, modifies, displays, or transmits information
- Software is a vehicle for delivering a product
  - Supports or directly provides system functionality
  - Controls other programs (e.g., an operating system)
  - Effects communications (e.g., networking software)
  - Helps build other software (e.g., software tools)

Software Characteristics:

1. **Software is developed or engineered; it is not manufactured in the classical sense.**

   Although some similarities exist between software development & hardware manufacturing both the activities are different. In both activities high quality is achieved through good design, but manufacturing phase will introduce quality problems that are non-existent in software. Both activities depend on people but relationship between people applied and work accomplished is different.

2. **Software does not wear out. How ever it deteriorates due to change**

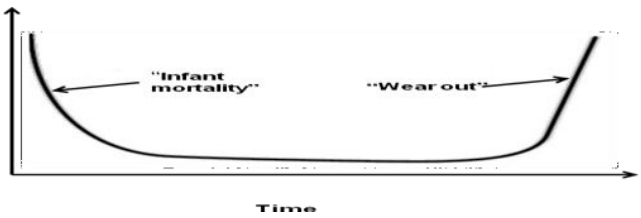

Fig 1.1: Failure curve for Hardware

It indicates that hardware exhibits high failure rates early in its life. Defects are corrected, and failure rate drops to a steady-state level for some period of time. As time passes, the failure rate rises again as hardware components suffer from cumulative effects of dust, temperature and other environmental maladies. Simply hardware begins to wear out.

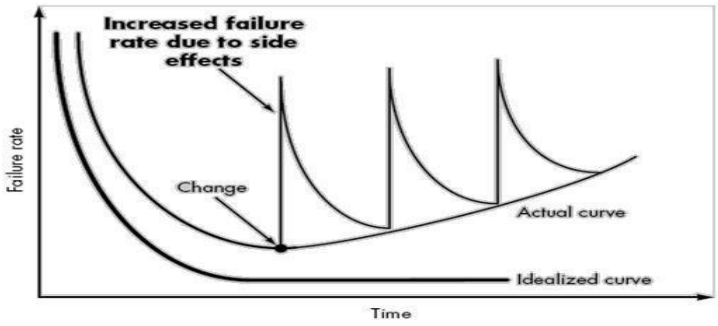

Fig 1.2: Failure Curve for Software

Software doesn't wear out so it should take form of idealized curve. Undiscovered defects will cause high failure rates early in its life of a program. However these are corrected and curve flattens the actual curve shows that during its life software will undergo change. As changes are made, it is likely that errors will be introduced, causing failure rate curve to spike again. Before curve can return to original state, another change is requested, causing curve to spike again. Slowly failure rate level begins to rise – the software is deteriorating due to change.

3. **Although the industry is moving towards component based construction, most software continues to be custom built**

   Custom-built: building according to needs of customer

   The main of component-based construction is reuse. In the hardware world, component reuse is a natural part of engineering process. In software world it has only begun to achieve on a broad scale.

Evolution of Software:

1950's-60's (mid): Batch orientation, limited distribution

1960's (mid)-1970's (mid): multiuser, realtime, database, product s/w

1970’s mid-1980’s (late): distributed systems, low cost h/w, and consumer impact s/w

1980’s (late)-2000: object-oriented techniques, artificial neural networks.

Changing Nature of Software

- **Seven Broad Categories of software are challenges for software engineers**

 System Software: System software is a collection of programs written to service other programs

  Software is determinate is the order and timing of its inputs, processing, and outputs is predictable (Ex compiler, file management utilities)
  Indeterminate if the order and timing of its input, processing, and outputs is not predictable in advance. (Ex: networking software, operating system components)

 Application Software: This is designed to help users in order to perform a specific task. Applications in this area process business or technical data.Ex:C,JAVA

 Engineering and Scientific Software: Engineering and scientific software have been characterized by "number crunching" algorithms (which performs complex, lengthy and complex numeric calculations).However modern applications are moving away from numeric algorithms to computer-aided design, system simulation.

 E**mbedded Software:** It resides within a product or system and is used to implement and control features and functions of end-user and system itself.

  Ex: Keypad control of microwave oven, fuel control etc

 Product-line Software: Designed to provide a specific capability for se by many different customers.

  Ex: Computer Graphics, entertainment, databasemanagement.

 Web-Applications: These are applications which are accessed over a network.

 Artificial Intelligence Software: Artificial intelligence (AI) software makes use of nonnumeric algorithms to solve complex problems that are not amenable to computation or straightforward analysis

  Ex: Robotics, Expert Systems, Game Playing etc.

Software Myths:

- Propagate misinformation and confusion
- Three types of myths
  - Management myth
  - Customer myth
  - Practitioner's myth

Myth (1)

-Already we have a book of standards and procedures for building software wont that provide my people with everything they need to know?

Reality: The book of standards may very well exist but is it used? Are software practitioners aware of its existence? Is it complete? Is it adaptable?

Myth (2)

-If we get behind schedule we can add more programmers and can catch up.

Reality: As new people are added, people who were working must spend time educating the newcomers, thereby reducing amount of time spent on productive development effort.

Myth (3)

-Outsourcing the software project to third party, we can relax and let that party build it.

Reality: If an organization doesn't understand how to manage and control software projects internally, will face struggle when it outsources projects.

Customer Myths:

Myth (1)

-General statement of objective is enough to begin writing programs, the details can be filled in later.

Reality: Unambiguous requirements can be developed only through efficient and continuous communication between developer and customer.

Myth (2)

-Software requirements continually change but change can be easily accommodated because software is flexible.

Reality: When requirement changes are requested early cost impact is relatively small. However as time passes cost impact grows rapidly.

Practitioner Myths:

Myth (1)

-Once the program is written, the job has been done.

Reality: Industry data indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to customer for first time.

Myth (2)

Until the program is running, there is no way of assessing the quality.

Reality: Formal technical reviews have been found more effective than actual testing

Myth (3)

-The only deliverable work product is the working program

Reality: A working program is only part of software configuration. Documentation provides a foundation for successful engineering.

Myth (4)

-Software Engineering creates voluminous and unnecessary documentation and invariably slows down software development.

Reality: S.E is not about creating documents. It is about creating quality. Better quality leads to reduced rework.

Software Engineering-A Layered Technology:

Software Engineering Definitions

Software Engineering:

(1) The application of systematic, disciplined quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software

(2) the study of approaches in (1).

Fig 1.3: A Layered Structure

Quality Focus: Bedrock that supports Software Engineering.

Process - Foundation for software Engineering. It defines a framework that must be established for effective delivery of software engineering technology. This provides a basis for management control of software projects.

Methods: Provide technical how-to's for building Software. Methods encompass a broad array of tasks that include

- Communication
- Requirements Analysis
- Design Modeling
- Program Construction
- Testing
- Support

Tools: Provide semi-automatic and automatic support to methods. When well-integrated, it is a CASE system. (Computer-Aided Software Engineering)

Process Framework

A **Process Framework** establishes the foundation for a complete software process by identifying a small number of **framework activities** that are applicable to all s/w projects, regardless of size/complexity and set of **umbrella activities which are** applicable across entire s/w process

Each framework activity is populated by a set of software engineering action-collection of related tasks that produce a major engineering work product

These are the 5 framework activities

Communication: This activity involves heavy communication and collaboration with customer and encompasses requirements gathering.

Planning: It establishes a software project plan for software engineering work that follows. It describes technical tasks to be conducted, risks that are likely, resources that will be required, work product to be produced and a work schedule.

Modeling: This activity encompasses creating models that allows customer to better understand software requirements and design.

Construction: This activity combines code generation and testing.

Deployment : The software is delivered to customer who evaluates the delivered product and provides feedback.

A *task set* defines the actual work to be done to accomplish the objectives of a software engineering action. For example, "requirements gathering" is an important software engineering action that occurs during the **communication** activity. The goal of requirements gathering is to understand what various stakeholders want from the software that is to be built.

For a small, relatively simple project, the task set for requirements gathering might look like this:

1. Make a list of stakeholders for the project.
2. Invite all stakeholders to an informal meeting.
3. Ask each stakeholder to make a list of features and functions required.
4. Discuss requirements and build a final list.
5. Prioritize requirements.
6. Note areas of uncertainty.

Each Software engineering action is represented by a number of task sets. The task set that best accommodates the needs o project and characteristics of team is chosen.

.

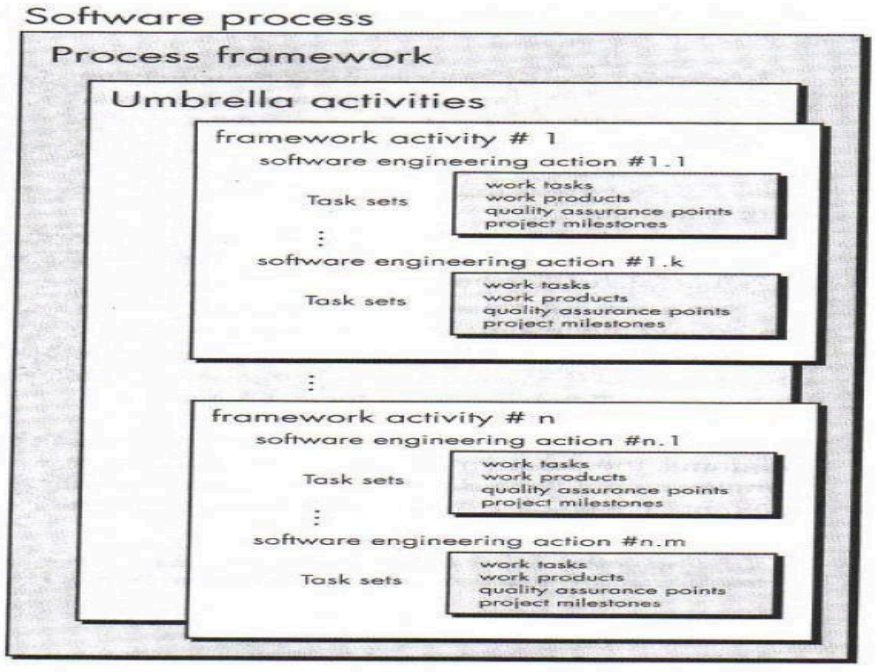

Fig 1.4: Process Framework

Umbrella Activities

Software Project Tracking and Control: Asses progress against project plan and take necessary action to maintain schedule.

Formal Technical Reviews: Assesses software engineering work products in an effort to uncover or remove errors before they are propagated to next action.

Software Quality Assurance: Defines and conducts activities required to ensure quality

Software Configuration Management: Manages change throughout the process.

Work Product Preparation and Production: encompasses activities required to create work products such as documents, forms, logs reports etc.

Reusability Management: Defines criteria for work product reuse and establishes mechanisms to achieve reusable components.

Measurement: Defines and collects process, project and product measures that assist team in delivering software that meets customer needs

Risk Management: Assesses risks that may affect outcome of project and quality

Capability Maturity Model Integration

Developed by SEI (Software Engineering Institute)

Assess the process model followed by an organization and rate the organization with different levels

A set of software engineering capabilities should be present as organizations reach different levels of process capability and maturity.

CMMI Process Meta Model can be represented in different ways

1. A Continuous Model
2. A Staged Model

<u>Continuous Model:</u>
- Levels are called capability levels.

-Describes a process in 2 dimensions

-Each process area is assessed against specific goals and practices and is rated according to the following capability levels.

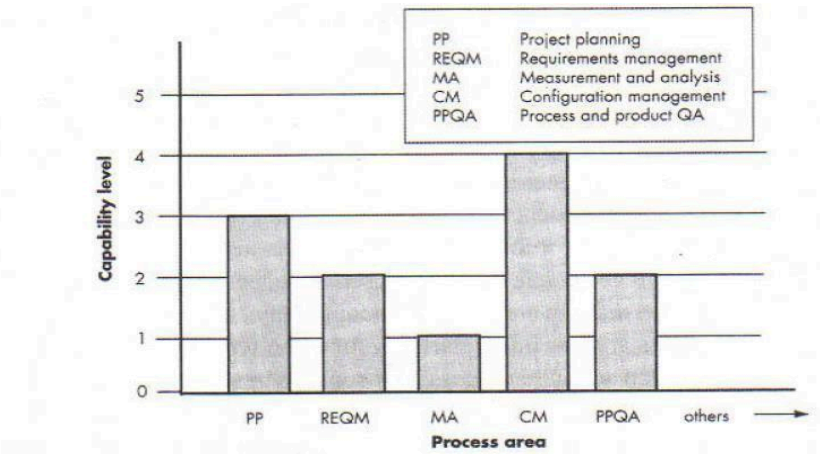

Fig 1.5: CMMI Continuous Model

Six Levels of CMMI

- **Level 0:Incomplete**
- **Level 1:Performed**
- **Level 2:Managed**
- **Level 3:Defined**
- **Level 4:Quantitatively managed**
- **Level 5:Optimized**

Incomplete

-Process is adhoc. Objective and goal of process areas are not known

Performed

All the specific goals and practices process area have been satisfied but performance may not be stable they do not meet some specific objectives such as quality, cost and schedule

Managed

-Activities are monitored, reviewed, evaluated and controlled to achieve a given purpose and cost, schedule, quality are maintained. These companies have some planned processes within teams and teams are made to represent them for projects handled by them. However processes are not standardized across organization.

Defined

-All level2 criteria have been satisfied and in addition process are well defined and are followed throughout the organization

Quantitatively Managed

-Metrics and indicators are available to measure the process and quality

Optimized (perfect&complete)

- Continuous process improvement based on quantitative feedback from the user

-Use of innovative ideas and techniques, statistical quality control and other methods for process improvement.

In addition to specific goals and practices for each process area 5 generic goals correspond to capability levels. In order to achieve a particular capability level the generic goal for that level and generic practices correspond to the goal must be achieved.

Staged Model

- This model is used if you have no clue of how to improve the process for quality software.
- It gives a suggestion of what things other organizations have found helpful to work first
- Levels are called maturity levels

Process areas required to achieve a maturity level		
Optimizing	*Continuous process improvement*	Organizational Innovation and Deployment Causal Analysis and Resolution
Quantitatively managed	*Quantitative management*	Organizational Process Performance Quantitative Project Management
Defined	*Process standardization*	Requirements Development Technical Solution Product Integration Verification Validation Organizational Process Focus Organizational Process Definition Organizational Training Integrated Project Management Integrated Supplier Management Risk Management Decision Analysis and Resolution Organizational Environment for Integration Integrated Teaming
Managed	*Basic project management*	Requirements Management Project Planning Project Monitoring and Control Supplier Agreement Management Measurement and Analysis Process and Product Quality Assurance Configuration Management
Performed		

Fig 1.5: CMMI Staged Model

Process Patterns

- The **software process** can be defined as a collection of patterns that define a set of activities, actions, work tasks, work products and/or related behaviors.
- A **process pattern** provides us with a template. A consistent method for describing an important characteristic of s/w process.
- Patterns can be defined at any level of abstraction. In some cases it is used to define complete process (e.g. prototyping).In other situations patterns can be used to describe an important framework activity (e.g. planning) or a task within a framework activity(e.g. project-estimating).
- **Pattern Template**

  -Pattern Name: The pattern given a meaningful name that describes its function within the software process. (e.g. requirements unclear)

  -Intent: The objective of pattern is described briefly. For ex the objective of pattern is to build a model that can be assessed iteratively by stakeholders.

  -Type: pattern type is specified. Suggests 3 types

  -**Task pattern**: defines a software engineering action or work task that is part o process (e.g. requirements gathering)

  - **Stage pattern:** defines a framework activity for the process (e.g. communication)

  -**Phase Pattern**: defines sequence of framework activities that occur with the process (ex spiral model or prototyping)

  Initial Context: The condition under which pattern applies are described. Prior to initiation of pattern the following conditions must be met.

  For example:

  1) Stakeholders have been identified
  2) Mode of communication between software team and stakeholder has been established.
  3) Problem to be solved has been identified by stakeholders.4) an initial understanding of requirements has been developed

  Problem: The problem to be solved by pattern is described.

  Ex: Requirements are hazy or non-existent i.e., stakeholders are unsure of what they want. that is they cannot describe software requirements in detail.

Solution: The implementation of pattern is described ex: A description of prototyping process is described here.

Resulting Context: The conditions that ill result once the pattern has been successfully implemented are described.

Ex: A software prototype that identifies basic requirements are approved by stakeholders And a prototype may evolve through a series of increments to become production software.

Related Patterns: A list of process patterns that are related to this one are provided.

Ex: customer-communication, iterative design&development, customer assessment, requirements extraction.

Known Uses and Examples: The specific instances in which pattern are applicable are indicated.

Process Assessment

- It attempts to keep a check on the current state of the software process with the intention of improving it.

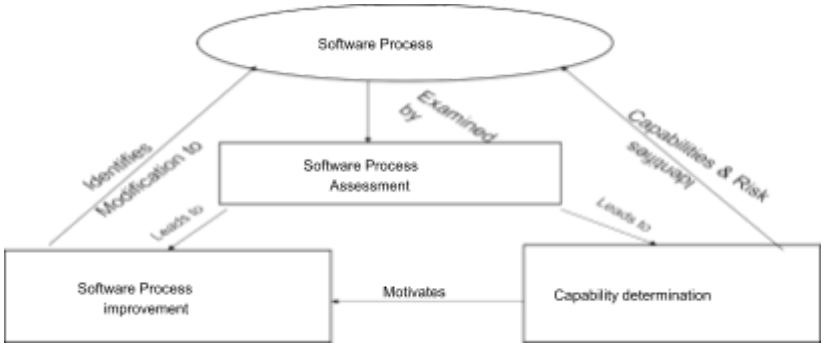

Fig 1.6: Process Assessment

- **Standard CMMI assessment for process improvement (SCAMPI):** provides a 5 step process assessment model that incorporates initiating, diagnosing, establishing, acting and learning. The SCAMPI uses SEI CMMI as basis for assessment.
- **CMM based appraisal for internal process improvement:** provides a diagnostic technique for assessing relative maturity of a software organization

- **SPICE(ISO/IEC 15504):**standard defines a set of requirements or software process assessment
- **ISO 9001:2000 for software:** is a generic standard that applies to any organization that wants to improve overall quality of products, systems or services that it provides.

Process Models

- Process Models help in the software development
- They Guide the software team through a set of framework activities
- Process Models may be linear, incremental or evolutionary

Waterfall Model

- Used when requirements are well understood in the beginning
- Also called classic life cycle model
- A systematic, sequential approach to Software development
- Begins with customer specification of Requirements and progresses through planning, modeling, construction and deployment.

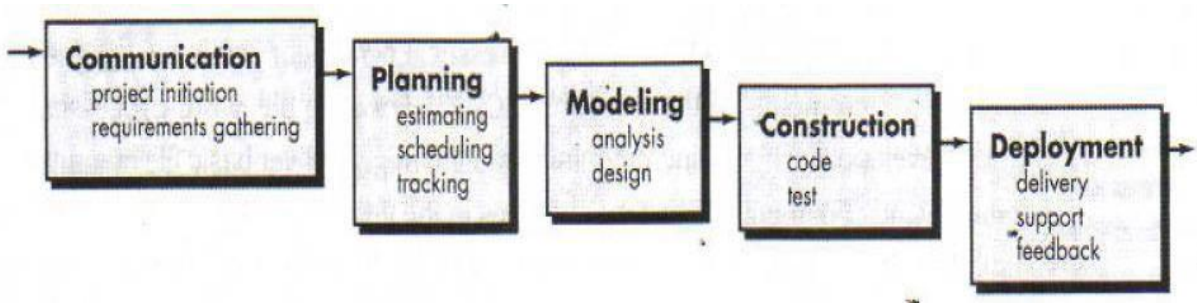

Fig 1.7: Waterfall Model

Disadvantages:

- Real projects rarely follow the sequential flow since they are always iterative
- The model requires requirements to be explicitly spelled out in the beginning, which is often difficult
- A working model is not available until late in the project time span

Incremental Process Model

- Linear sequential model is not suited for projects which are iterative in nature
- Incremental model suits such projects
- Used when initial requirements are reasonably well-defined and compelling need to provide limited functionality to users quickly and then refine and expand on that functionality in later releases

  It combines elements of waterfall model in an iterative fashion.

The incremental model applies linear sequences in a staggered ashion as calendar time progresses.Each linear sequence provides deliverable increments of software.For ex word processing sotware developed sing incremental paradigm might deliver basic file management,editing,and document production functions in 1st increment ;more sophisticated editing and document production capabilities in 2nd increment;spelling and grammar checking in 3rd increment; etc

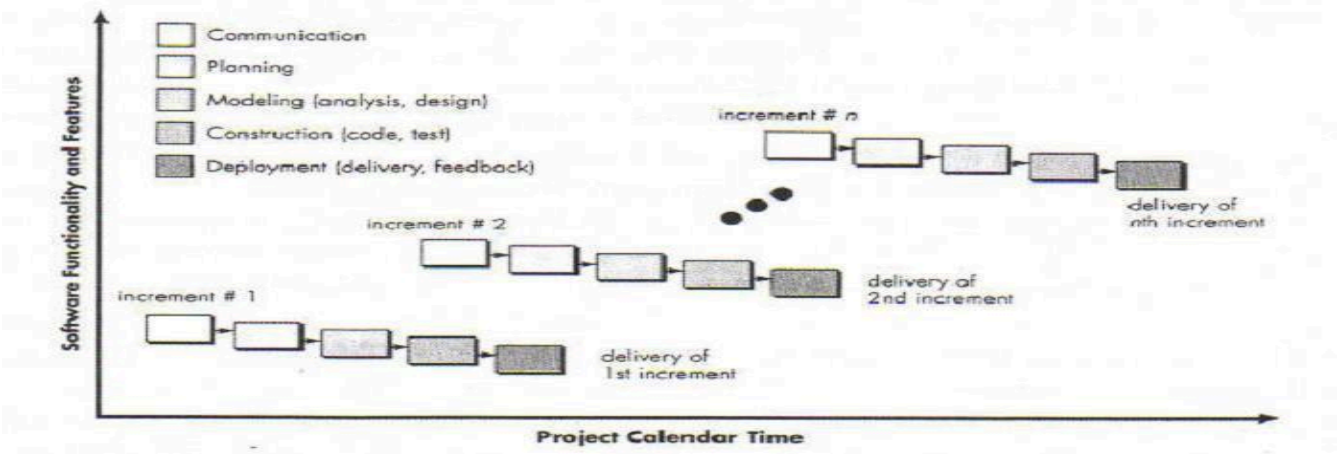

Fig 1.8: Incremental Model

- 1st increment constitutes Core product
- Basic requirements are addressed
- Core product undergoes detailed evaluation by the customer

As a result, plan is developed for the next increment Plan addresses the modification of core product to better meet the needs of customer

- Process is repeated until the complete product is produced

The incremental process model unlike prototyping and other evolutionary approaches,is iterative in nature.But unlike prototyping,the incremental model focuses on delivery of an operational product with each increment.

This model is particularly useul when staffing is unavailable for a complete implementation by business deadline that has been established for the project.

THE RAD MODEL (Rapid Application Development)

- An incremental software process model
- Having a short development cycle
- High-speed adoption of the waterfall model using a component based construction approach
- Creates a fully functional system within a very short span time of 60 to 90 days
- Multiple software teams work in parallel on different functions
- Modeling encompasses three major phases: Business modeling, Data modeling and process modeling
- Construction uses reusable components, automatic code generation and testing
- Problems in RAD
- Requires a number of RAD teams
- Requires commitment from both developer and customer for rapid-fire completion of activities otherwise it fails
- If system cannot be modularized properly project will fail.
- Not suited when technical risks are high

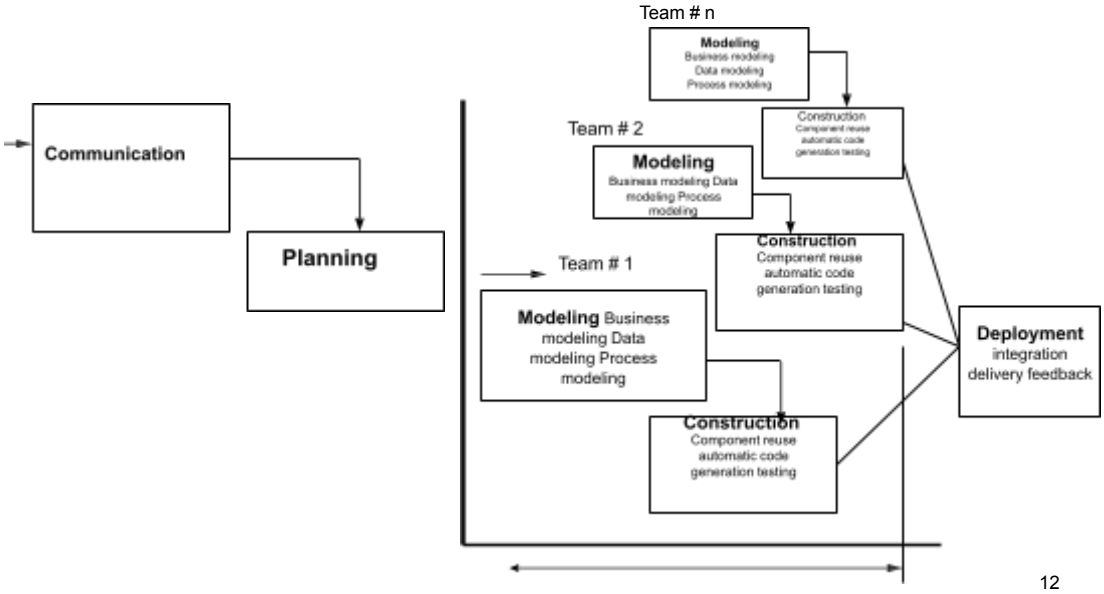

Fig 1.8: Rapid Application Development Model

Evolutionary Process Model

- Software evolves over a period of time
- Business and product requirements often change as development proceeds making a straight-line path to an end product unrealistic
- Evolutionary models are iterative and as such are applicable to modern day applications
- Types of evolutionary models
  - Prototyping
  - Spiral model
  - Concurrent development model

Prototyping Model

- Mock up or model(throw away version) of a software product
- Used when customer defines a set of objective but does not identify input,output,or processing requirements
- Developer is not sure of:
  - efficiency of an algorithm
  - adaptability of an operating system

– human/machine interaction

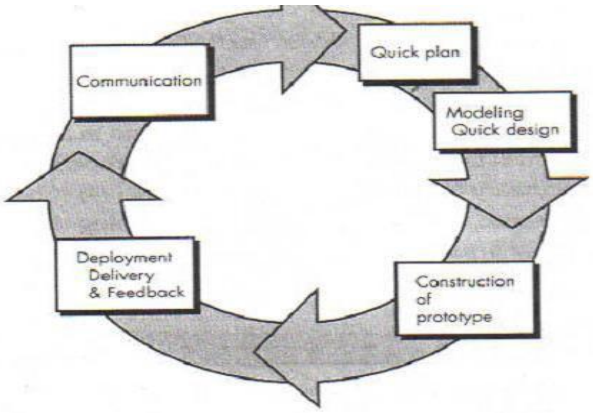

Fig 1.9: Prototyping Model

- Begins with requirement gathering
- Identify whatever requirements are known
- Outline areas where further definition is mandatory
- A quick design occur
- Quick design leads to the construction of prototype
- Prototype is evaluated by the customer
- Requirements are refined
- Prototype is turned to satisfy the needs of customer,while at the same time enabling developer to better understand hat needs to be done.
- Ideally prototype serves as a mechanism for identifying software requirements.

Disadvantages:

- In a rush to get it working, overall software quality or long term maintainability are generally overlooked
- The developer often makes implementation compromises in order to get a prototype working quickly.An inappropriate OS or PL may be used simply because it is available and known;an inefficient algorithm may be implemented simply to demonstrate capability.After a time,developer may become comfortable with these choices and forget all the reasons why they were inappropriate.

Spiral Model

- This is a model proposed by boehm it is an evolutionary model which combines the best feature of the classical life cycle and the iterative nature of prototype model
- Include new element : Risk element
- Starts in middle and continually visits the basic tasks of communication, planning,modeling,construction and deployment
- Using spiral model,software is developed in a series of evoltionary releases.During early iterations,the release might be a model or prototype.During later iterations increasinly more complete versions of engineered system are produced
- Realistic approach to the development of large scale system and software
- Unlike other process models that end when software is delivered,the spiral model can be adapted to apply through out the life of computer software.Therefore the first circuit around the spiral represent a “concept development project” which starts at the core of the spiral and contines for multiple iterations until concept development is complete.If the concept is to be developed into an actual product,process proceeds outward on the spiral and a “new product development project “commences.The new product will evolve through a number of iterations around the spiral.Later,a circuit around spiral might be used to represent a “product enhancement project”.The spiral remains operative until the software is retired.
- It is difficult to convince customers that the evolutionary approach is controllable**.**

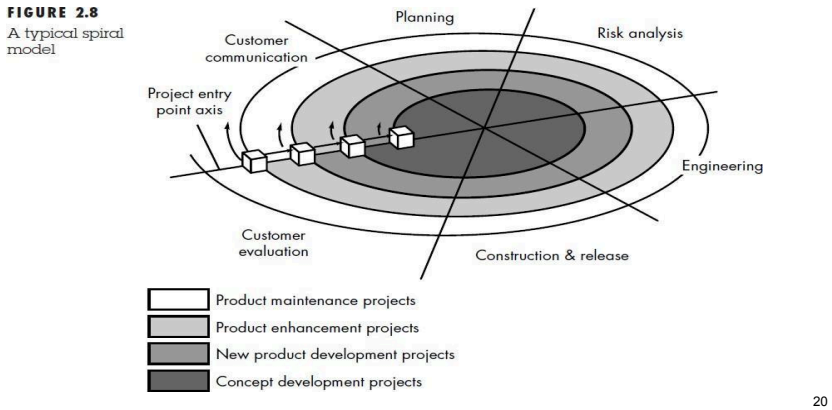

Fig 1.10: Spiral Model

Concurrent Development Model

- This is sometimes called concurrent engineering.This is represented schematically as a series of framework activities,software engineering actions and tasks,and their associated states.
- Fig provides schematic representation of one software engineering task within modelling activity.The activity modelling may in any one of these states at any given time.Similarly other activities can be represented in a similar manner.
- All activities exist concurrently but reside in different states.
- For ex.early in a project communication activity has completed its first iteration and exists in awaiting changes state.The modelling activity wchich existed in none state while initial communication was completed now makes a transition to underdevelopment state.If however customer indicates changes in requirements must be made,the modelling activity moves from under development to awaiting changes state
- The concurrent process model dfines a series of events that will trigger transition from state to state for each software engineering activities,actions and tasks.
- This model is applicable to all types of software development and provides correct pictre of accurate state of a project.

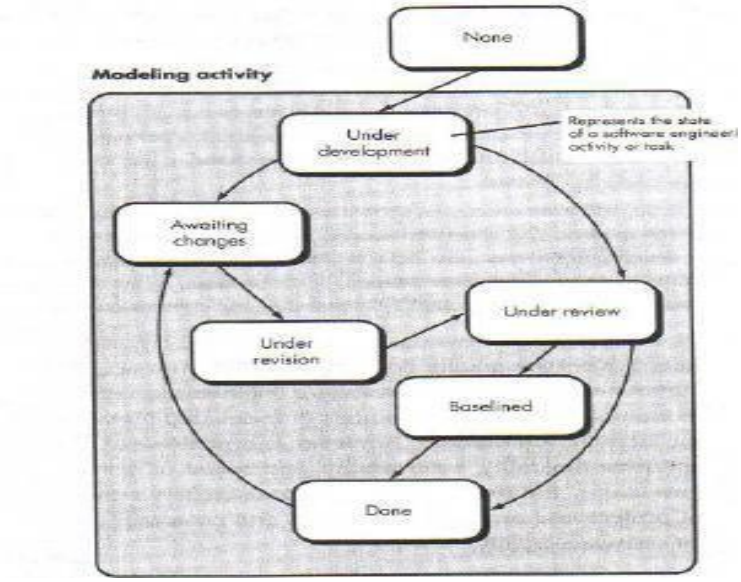

Fig 1.11: Concurrent Model

Unified Process Model

- This model is proposed by ivan Jacobson, Grady Booch, Rumbaugh
- This is a use-case driven, architecture-centric,iterative and incremental software process.
- This is a framework for object-oriented software engineering sing UML.
- The **inception** phase of Unified Process(UP) encompasses oth customer communication and planning activities.By colloborating with customer,business requirements for the software are identified.Fundamental business requirements are expressed through a set of preliminary use-cases that describes what features and functions are desired by major class of sers. Architecture at this point is nothing but outline of features and functions.Planning identifies resources,assesses major risks,defines a schedule.
- The **elaboration** phase encompasses communication,planning and modelling.Elaboration reines and expands preliminary use-cases

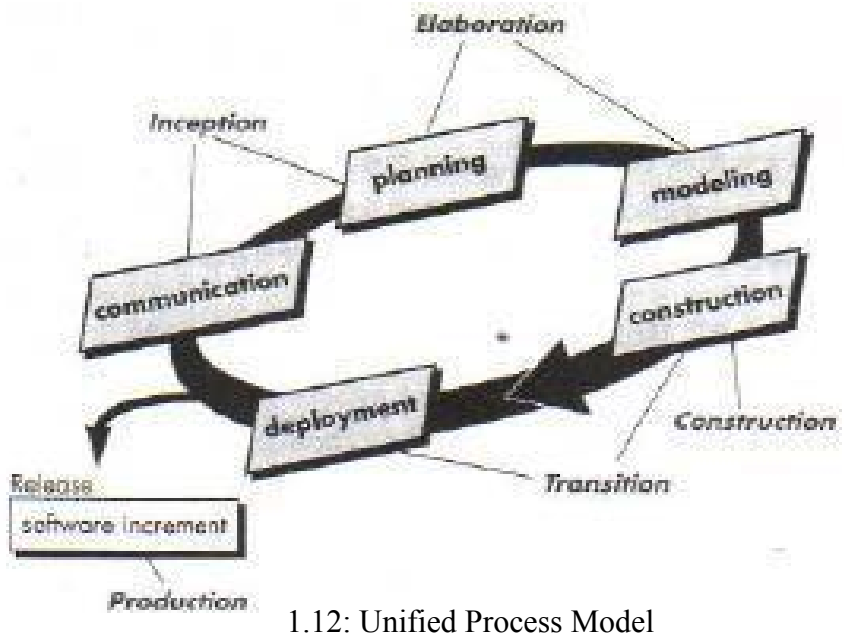

Fig 1.12: Unified Process Model

that were developed as part of inception phase.Expands architectural representation to include ive different views of software.the use-case model,the analysis model,the design model,the implementation model and the deployment model.Modification to the plan may be made at tis time

- **Construction** of UP is identical to constuction activity of generic process models.All necessary features and functions are implemented in souce code.As components are being developed unit test is bein conducted.
- The **transition** phase of UP encompasses later stages of construction activity and first part of deployment activity.Softwareis given to end-users for beta-testing.In addition software team creates installation procedres,user manuals etc.

- The **prodction** phase cooincides with deployment activity .During tis phase on-going use of software is monitored,and defect reports and request for changes are submitted and evaluated.

Personal and Team Software Process Models

The best software process is one that is close to people who will be doing the work. Watts Humphrey argues that it is possible to create personal and/or team software process.Both require ardwaork,training&co-ordination butoth are acievable. Personal s/w process Every developer uses some process to build computer software.The process may be inefficient,effective or successful but a process does exist. In order to change an ineffiecient process an individal mustmove through 4 phases,each require training and careful instrumentation.

Planning: Basedon requirements develop size and resource estimates.Metrics are recorded on worksheets.Finally development tasks are identified & a schedule is created.

High-Level Design: External specifications for each component to be constructed & developed & a component design is constructed.Prototypes are built when uncertainity exists.

High-Level Design Review: Formalverification methods are applied to uncover errors in design.Metrics are maintained for all important tasks and work result.

Development: Design is refined&reviewed.Code is generated,reviewed,compiled&tested.Metrics are maintained for all imp tasks.

Postmortem: Using measure & metrics collected,eectiveness of process is determined.

Team s/wprocess

Many projects are addressed by a team of practitioners

The goal of TSP is to ils a team that organises itself to produce high quality s/w.

TSP objectives

1) Build self directed teams that plan & track their goals
2) Show manager how to coach and motivate their teams.
3) Accelerate software process improvement by making CMM level 5 normal and expected.

4) Facilitate university teaching to upgrade team skills

TSP defines following framework activities project launch, high level design,implementation,integration,test & postmortem

TSP uses wide variety of scripts & standards that serve to guide team members in their work. Scripts define specific activities.

This is an example for project launch script.

- Review project objectives with management and agree on and document team goals.
- Establish team roles.
- Define the team's development process.
- Make a quality plan and set quality targets.
- Plan for the needed support facilities.
- Produce an overall development strategy.
- Make a development plan for the entire project.
- Make detailed plans for each engineer for the next phase.
- Merge the individual plans into a team plan.
- Rebalance team workload to achieve a minimum overall schedule.
- Assess project risks and assign tracking responsibility for each key risk.