UNIT - I
Software and Software Engineering

The Nature of Software:

Software takes on a dual role.
It is i) A product and at the same time
ii) The vehicle for delivering product.

As a product, it resides within a mobile phone or operates inside the computer, software is an information transformer- producing, managing, acquiring, modifying, displaying or transmitting information.

As the vehicle used to deliver the product, software acts as the basis for the control of the computer(operating systems), the communication of information (networks) and the creation and control of other programs(Software tools and environments).

Software:

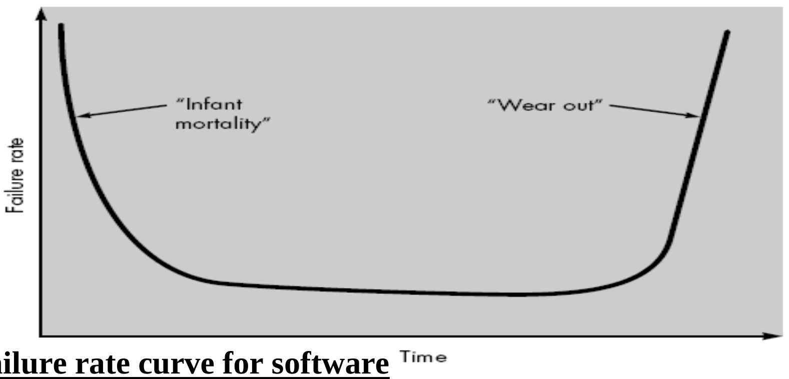
Software is

- Instructions or programs that when executed provide desired function and performance
- Data structures that enable the programs to adequately manipulate information
- Documents that describe the operation and use of the program

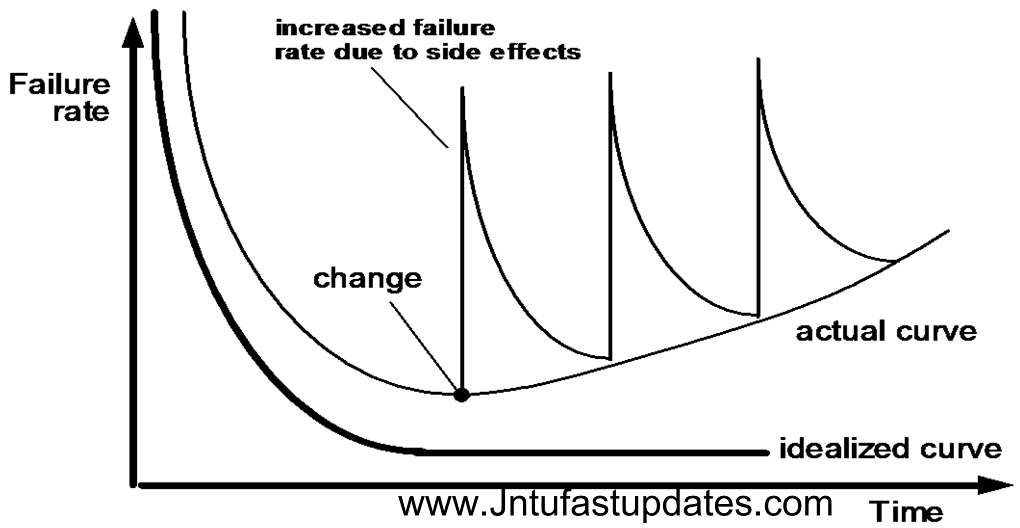
Software Characteristics:

1. Software is a logical rather than a physical system element.
2. Software is developed or engineered, it is not manufactured in the classical sense.
   - Manufacturing phase of a product introduces quality problems.
   - Software costs are concentrated in engineering, hence cannot be managed as if they are manufacturing projects.
3. Software doesn't "wear out". It deteriorates.

Failure rate curve for hardware



Failure rate curve for software



Hardware: Worn out hardware components can be replaced.
Software: Software maintenance requires changing the design and or code.

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

Software Applications: (or Changing Nature of Software):

1. **System Software:** System software is a collection of programs written to service other programs. e.g. compilers, editors, operating system components, drivers, networking software etc.

2. **Application Software:** Application software is a standalone programs that solve a specific business need. Applications software like data processing applications, point of sale transaction processing, real time manufacturing process control helps in managing and controlling business function.

3. **Engineering and Scientific Software:** Engineering and scientific software have been characterized by number “crunching" algorithms. Applications range from astronomy to volcanology, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing.

4. **Embedded Software:** Embedded software resides in read-only memory and is used to control products. Eg. Keypad control for a microwave oven, digital functions in an automobile such as fuel control, dashboard displays, and braking systems.

5. **Product-line software:** Product line software designed to provide a specific capability for use by many different customers. Product-line software can focus on limited and esoteric marketplace (eg. Inventory control products) or address mass consumer markets(eg. Word processing, spreadsheets, computer graphics, multimedia, entertainment, database management and personal and financial applications.).

6. **Web Applications: The** Web pages retrieved by a browser are software that incorporates executable instructions (e.g., CGI, HTML, Perl, or Java), and data (e.g., hypertext and a variety of visual and audio formats).

7. **Artificial Intelligence Software:** Artificial intelligence (AI) software makes use of non numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis. Eg. Expert systems or knowledge based systems, pattern recognition (image and voice), artificial neural networks, theorem proving, and game playing.

Software—New Categories

1. **Open-world computing:** It is the rapid growth of wireless networking may soon lead to true pervasive, distributed computing. The challenge for software engineers will be to develop systems and application software that will allow mobile devices, personal computers and enterprise systems to communicate across vast networks.

2. **Netsourcing:** It is the world wide web is rapidly becoming a computing engine as well as a content provider.

3. **Open- Source:** Open source is a growing trend that results in distribution of source code for systems applications so that many people can contribute to its development.

4. **Ubiquitous computing:** Wireless computing

Software Issues:

- Problems associated with software development are not just restricted to software that doesn't function but it also includes
  - How we develop software?
  - How we support a growing volume of existing software?
  - How we keep pace with a growing demand for more software?

Participants in Software Development:

- Software Managers – manage budgets, monitor progress made against project plans, always look for time to market without compromising quality
- Customers - users
- Practitioners – developers, testers

Legacy Software:

The older programs are referred to as Legacy software.

Why must it change?

- software must be adapted to meet the needs of new computing environments or technology.
- software must be enhanced to implement new business requirements.
- software must be extended to make it interoperable with other more modern systems or databases.
- software must be re-architected to make it viable within a network environment**.**

The Unique Nature of WebApps:

WebApps have evolved into sophisticated computing tools that not only provide stand-alone function to the end user, but also have be integrated with corporate databases and business applications.

The following attributes are found in majority of WebApps:

- **Network intensiveness.** A WebApp resides on a network and must serve the needs of a diverse community of clients.
- **Concurrency.** A large number of users may access the WebApp at one time.
- **Unpredictable load.** The number of users of the WebApp may vary by orders of magnitude from day to day.
- **Performance.** If a WebApp user must wait too long (for access, for server-side processing, for client-side formatting and display), he or she may decide to go elsewhere.
- **Availability.** Although expectation of 100 percent availability is unreasonable, users of popular WebApps often demand access on a "24/7/365" basis.
- **Data driven.** The primary function of many WebApps is to use hypermedia to present text, graphics, audio, and video content to the end-user.
- **Content sensitive.** The quality and aesthetic nature of content remains an important determinant of the quality of a WebApp.
- **Continuous evolution.** Unlike conventional application software that evolves over a series of planned, chronologically-spaced releases, Web applications evolve continuously.

- **Immediacy.** Although *immediacy*—the compelling need to get software to market quickly—is a characteristic of many application domains, WebApps often exhibit a time to market that can be a matter of a few days or weeks.
- **Security.** Because WebApps are available via network access, it is difficult, if not impossible, to limit the population of end-users who may access the application.
- **Aesthetics.** An undeniable part of the appeal of a WebApp is its look and feel.

Software Myths:

Manager's Myths

1. We have documented set of standards and procedures to build software
2. We have state-of-art software development tools
3. We can add more programmers and catch up if we are behind schedule (Mongolian horde Concept)
4. Outsource a software project to a 3rd party

Reality: A simple 'No'. Questions to be answered or issues to be understood up-front:

- Are the standards up-to-date?
- Are the standards being followed?
- Are the practitioners aware of the standards?
- Do the standards in place help speed up development time, reduce development cost, and improve quality?
- Are the tools being used effectively?
- Is the staff trained to use the tools effectively?
- Most often than not, adding more people to a delayed project delays it further as the new people need to be trained by the current project staff for them to be productive
- Organizations have to understand how to manage and control software projects internally when outsourcing a project so as to better manage the negotiations and monitor the progress

Customer's Myths

1. It is sufficient to begin developing a system with a general statement of objectives. Details can be filled in later.
2. Requirement Changes can be easily accommodated as software is flexible.

Reality: A simple 'No'. Questions to be answered or issues to be understood up-front

- Majority of software projects fail due to lack of up-front detailed definition – including function, behavior, performance, interfaces, assumptions etc
- Every change impacts the delivery date and the impact is significantly large as the project is nearing completion.

Practitioner's Myths

1. Once we write a program and get it to work our job is done
2. Until I get the program running I have no way of assessing its quality
3. The only deliverable for a successful product is a working program
4. Software Engineering will make us create voluminous and unnecessary documentation which slows us down.

Reality: A simple 'No'. Questions to be answered or issues to be understood up-front

- The sooner you begin coding the longer it takes to complete
- 60-80% of effort expended on a project is after it is delivered to the customer
- Formal reviews at various stages of a software development cycle help assess the quality of the system being implemented
- Documentation provides a guidance for maintaining and supporting the software built
- Software Engineering is about building good quality systems. Documentation helps accomplish that objective.

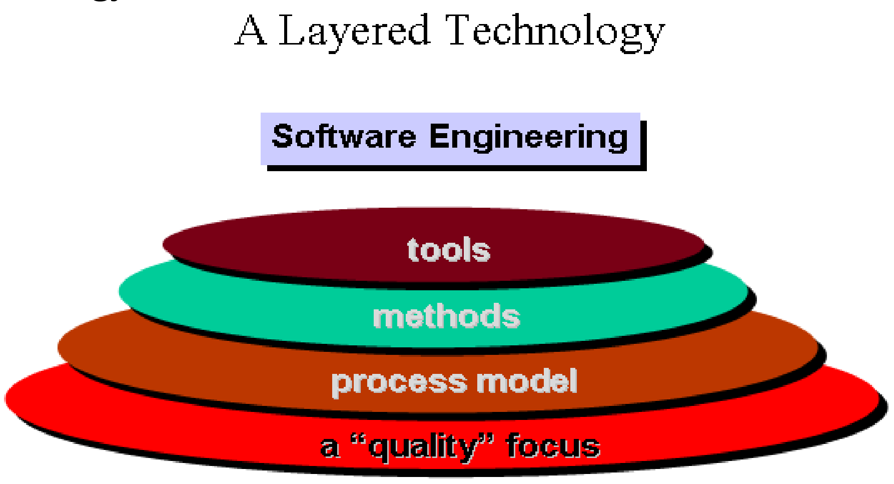
A Generic View of Software

Software Engineering:

Software Engineering is

- The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; i.e., the application of engineering to software.
- The study of approaches as in above.

A Layered Technology:



Quality:

- Any Engineering approach rests on organizational commitment to quality
- Bed rock that supports Software Engineering

Process:

Defines a framework for a set of Key Process Areas (KPAs) for effective delivery of software

- To manage software development,
- establish milestones,
- determine the work products to produce,
- manage quality and change

Methods:

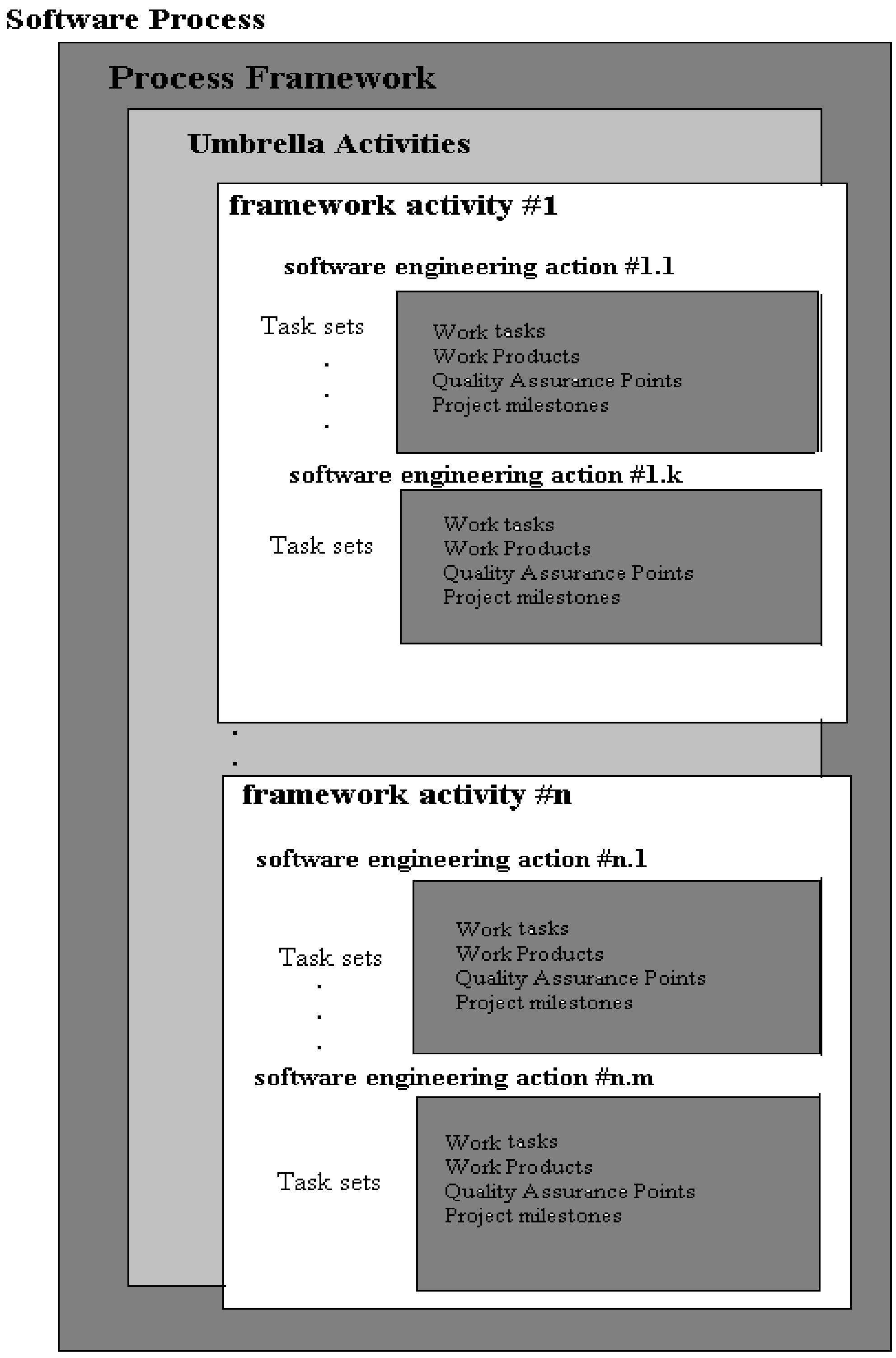
- Provide the technical how-to's for building Software
- Include tasks such as
  - analysis,
  - design,
  - program construction,
  - Testing,
  - support

Tools:

- Support the process and the methods
- Could be automated or semi-automated
- e.g., Computer Aided Software Engineering (CASE) Tool

Software Process:

A Process Framework establishes the foundation for a complete software engineering process by identifying a small number of framework activities that are applicable to all software projects.



A Process Framework establishes the foundations for a complete software process by identifying a small number of framework activities that are applicable to all software projects. It also encompasses a set of umbrella activities that are applicable across the entire software process.

Generic process framework activities:

1. **Communication:** This framework activity involves heavy communication and collaboration with the customer and encompasses requirement gathering etc.
2. **Planning:** This activity establishes a plan for the technical tasks to be conducted, the risks that are likely to be occurred, the resources that will be required, the work products to be produced and a work schedule.
3. **Modeling:** This activity encompasses the creation of models and the design that will achieve the requirements.
4. **Construction:** This activity combines code generation and and the resting that is required to uncover errors in the code.
5. **Deployment:** The software is delivered to the customer who evaluates the delivered product and provides feed back.

Each framework activity is composed of software engineering actions. And each software engineering action encompasses a set of work tasks.
Example: ***Modeling*** Framework activities composed of two software engineering actions – ***Analysis and Design.***
Analysis (Software Engineering Action) encompasses a set of work tasks—requirement gathering, elaboration, negotiation, specification and validation.
Design(Software Engineering Action) encompasses a set of work tasks—data design, architectural design, interface design and component-level design.

Umbrella Activities:

1. **Software project tracking and control:** allows the software team to asses the progress against the project plan and take necessary action to maintain schedule.
2. **Risk Management:** assesses risks that may effect the outcome of the project or the quality of the product.
3. **Software quality assurance:** defines and conducts the activities required to ensure software quality.
4. **Formal technical reviews:** is an effort to uncover and remove errors before they are propagated to the next action or activity.
5. **Measurement:** defines and collects process, project and products measures that assist the team in delivering software that meets customer's needs.
6. **Software configuration management:** manages the effects of change throughout the software process.
7. **Reusability management:** defines and establishes to achieve reusable components.
8. **Work product preparation production:** encompasses the activities required to create work products such as models, documents, logs, forms and lists.

Umbrella Activities are applied throughout the software process. All process models can be characterized within process framework.

Adapting a Process Model

- the overall flow of activities, actions, and tasks and the interdependencies among them
- the degree to which actions and tasks are defined within each framework activity
- the degree to which work products are identified and required
- the manner which quality assurance activities are applied
- the manner in which project tracking and control activities are applied
- the overall degree of detail and rigor with which the process is described

- the degree to which the customer and other stakeholders are involved with the project
- the level of autonomy given to the software team
- the degree to which team organization and roles are prescribed

Software Engineering Practice

The Essence of Practice

Polya outlined the essence of problem solving as follows:

1.*Understand the problem* (communication and analysis).
2.*Plan a solution* (modeling and software design).
3.*Carry out the plan* (code generation).
4.*Examine the result for accuracy* (testing and quality assurance).

1.Understand the Problem

- *Who has a stake in the solution to the problem?* That is, who are the stakeholders?
- *What are the unknowns?* What data, functions, and features are required to properly solve the problem?
- *Can the problem be compartmentalized?* Is it possible to represent smaller problems that may be easier to understand?
- *Can the problem be represented graphically?* Can an analysis model be created?

2.Plan the Solution

- *Have you seen similar problems before?* Are there patterns that are recognizable in a potential solution? Is there existing software that implements the data, functions, and features that are required?
- *Has a similar problem been solved?* If so, are elements of the solution reusable?
- *Can subproblems be defined?* If so, are solutions readily apparent for the subproblems?
- *Can you represent a solution in a manner that leads to effective implementation?* Can a design model be created?

3.Carry Out the Plan

- *Does the solution conform to the plan?* Is source code traceable to the design model?
- *Is each component part of the solution provably correct?* Has the design and code been reviewed, or better, have correctness proofs been applied to algorithm?

4.Examine the Result

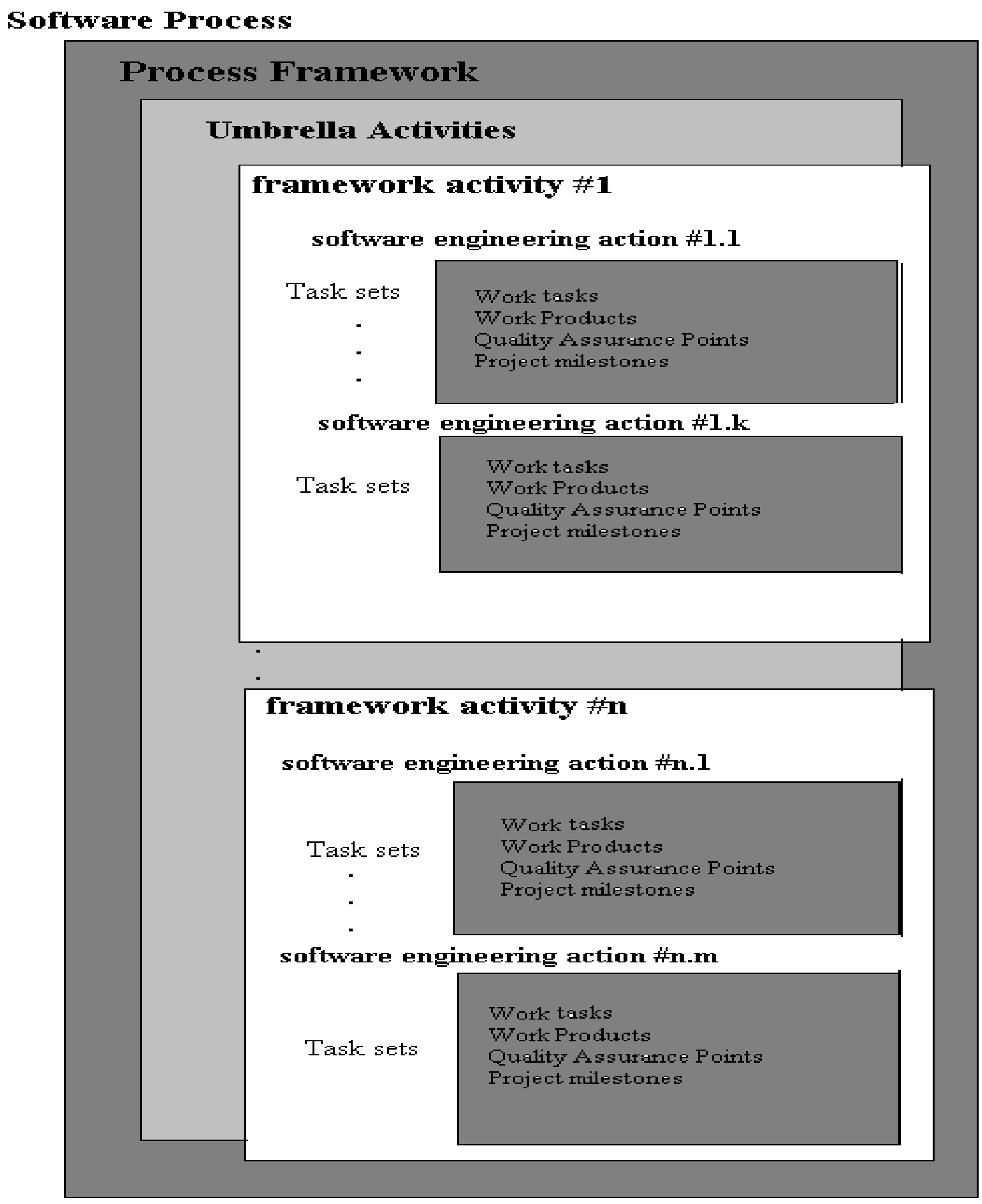
- *Is it possible to test each component part of the solution?* Has a reasonable testing strategy been implemented?
- *Does the solution produce results that conform to the data, functions, and features that are required?* Has the software been validated against all stakeholder requirements?

Hooker’s General Principles

1: *The Reason It All Exists*
2: *KISS (Keep It Simple, Stupid!)*
3: *Maintain the Vision*
4: *What You Produce, Others Will Consume*
5: *Be Open to the Future*
6: *Plan Ahead for Reuse*
7*: Think!*

PROCESS MODELS

A Process Framework establishes the foundation for a complete software engineering process by identifying a small number of framework activities that are applicable to all software projects.



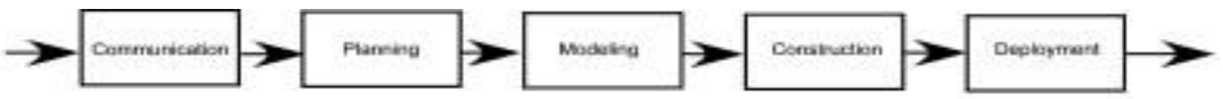
A Process Framework establishes the foundations for a complete software process by identifying a small number of framework activities that are applicable to all software projects. It also encompasses a set of umbrella activities that are applicable across the entire software process.

Generic process framework activities:

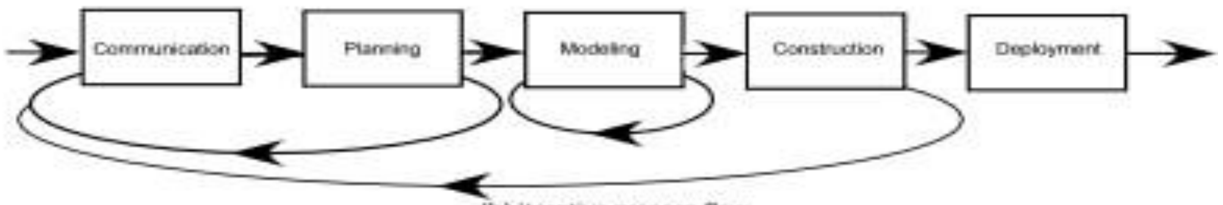
1. **Communication:** This framework activity involves heavy communication and collaboration with the customer and encompasses requirement gathering etc.
2. **Planning:** This activity establishes a plan for the technical tasks to be conducted, the risks that are likely to be occurred, the resources that will be required, the work products to be produced and a work schedule.
3. **Modeling:** This activity encompasses the creation of models and the design that will achieve the requirements.

4. **Construction:** This activity combines code generation and and the resting that is required to uncover errors in the code.
5. **Deployment:** The software is delivered to the customer who evaluates the delivered product and provides feed back.

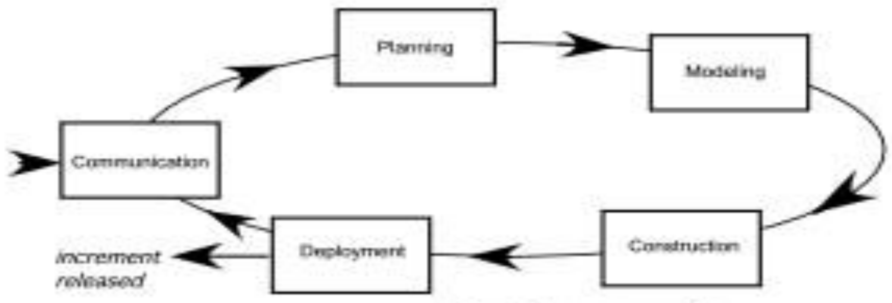
Process Flow



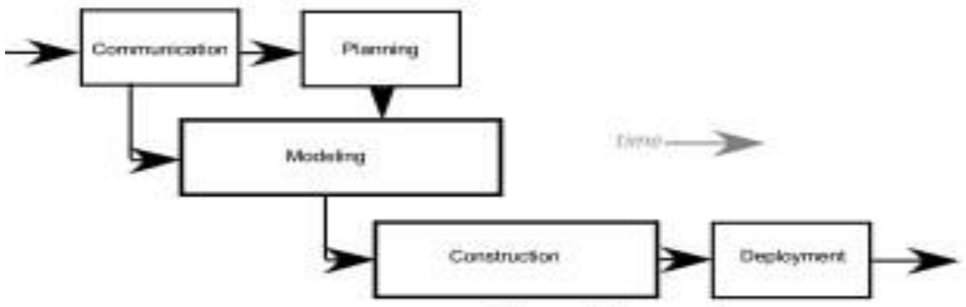
(a) linear process flow



(b) iterative process flow



(c) evolutionary process flow



(d) parallel process flow

- Linear process flow executes each of the five activities in sequence.
- An iterative process flow repeats one or more of the activities before proceeding to the next.
- An evolutionary process flow executes the activities in a circular manner. Each circuit leads to a more complete version of the software.
- A parallel process flow executes one or more activities in parallel with other activities (modeling for one aspect of the software in parallel with construction of another aspect of the software.

Identifying a Task Set

Before you can proceed with the process model, a key question: what **actions** are appropriate for a framework activity given the nature of the problem, the characteristics of the people and the stakeholders?

A task set defines the actual work to be done to accomplish the objectives of a software engineering action.

- A list of the task to be accomplished
- A list of the work products to be produced
- A list of the quality assurance filters to be applied

For example, a small software project requested by one person with simple requirements, the communication activity might encompass little more than a phone all with the stakeholder. Therefore, the only necessary action is phone conversation, the work tasks of this action are:

1. Make contact with stakeholder via telephone.
2. Discuss requirements and take notes.
3. Organize notes into a brief written statement of requirements.
4. E-mail to stakeholder for review and approval.

The task sets for Requirements gathering action for a **simple** project may include:

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.

The task sets for Requirements gathering action for a **big** project may include:

1. Make a list of stakeholders for the project.
2. Interview each stakeholders separately to determine overall wants and needs.
3. Build a preliminary list of functions and features based on stakeholder input.
4. Schedule a series of facilitated application specification meetings.
5. Conduct meetings.
6. Produce informal user scenarios as part of each meeting.
7. Refine user scenarios based on stakeholder feedback.
8. Build a revised list of stakeholder requirements.
9. Use quality function deployment techniques to prioritize requirements.
10. Package requirements so that they can be delivered incrementally.
11. Note constraints and restrictions that will be placed on the system.
12. Discuss methods for validating the system.

Process Patterns:

A process pattern provides us with template – a consistent method for describing an important characteristic of the software process.

Patterns can be defined at any level of abstraction. In some cases pattern might be used to describe a complete process and in other cases patterns can be used to describe an important framework activity or a task within a framework activity.

Ambler has proposed the following template for describing a process pattern:

Pattern Name: The pattern is given a meaningful name that describes its function within the software process.

Intent: The objective of the pattern is described briefly.

Type: The pattern type is specified. Ambler suggests three types:

- **Task patterns:** It defines a software engineering action or work task that is part of the process. Ex: Requirements gathering is a task pattern for the Communication framework activity.

- **Stage patterns:** It defines a framework activity for the process. Ex: Communication is a stage pattern. This pattern would incorporate the task pattern like requirement gathering etc.
- **Phase patterns:** It defines the sequence of framework activities that occur with the process. Ex: Spiral modeling or prototyping is a phase pattern.

Initial Context: The conditions under which the pattern applies are described.
Like 1. what organizational or team related activities have already occurred. 2. what is the entry state for the process. 3. what software engineering information or project information already exists.

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

Solution: The implementation of the pattern is described. It describes how project information that is available before the initiation of the pattern is transformed as a consequence of the successful execution of the pattern.

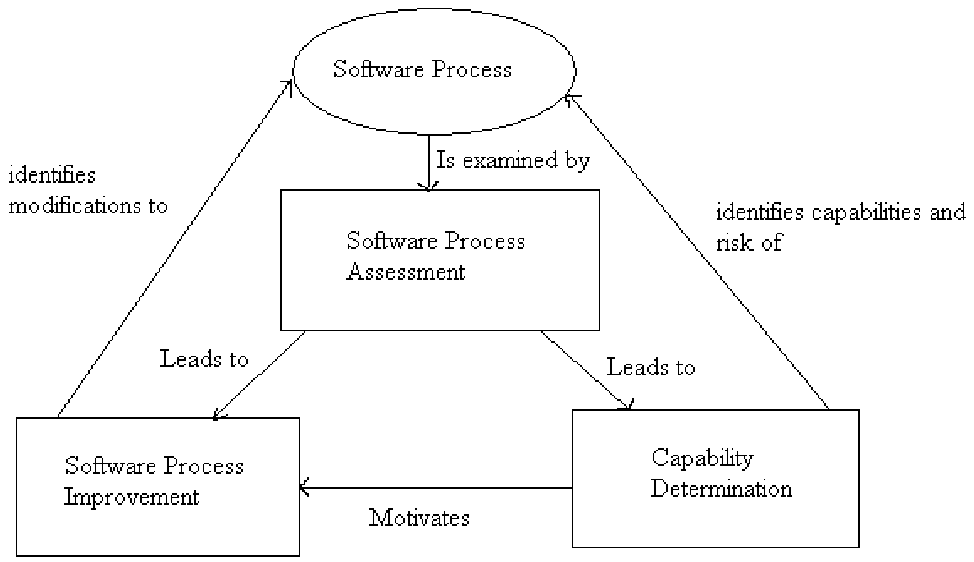
Resulting context: The conditions that will result once the pattern has been successfully implemented are described**.**

Related Patterns: The list of all process patterns that are directly related to this pattern are provided as a hierarchy form.

Known Uses/Examples: The specific instances in which the pattern is applicable are indicated. Ex: Communication pattern is mandatory at the beginning of the every software project.
Process patterns provide an effective mechanism for describing any software process.

Process Assessment:

The process itself should be assessed to ensure that it meets a set of basic process criteria that are essential for a successful software engineering. The relationship between the software process and the methods applied for assessment and improvement is shown in the following figure.


Software Process
Is examined by
identifies modifications to
identifies capabilities and risk of
Software Process Assessment
Leads to
Leads to
Software Process Improvement
Capability Determination
Motivates

Different Approaches to software process assessment:

1.<u>Standard CMMI Assessment Method for Process Improvement(SCAMPI):</u>

It provides a five-step process assessment model i.e.

- Initiating
- Diagnosing
- Establishing
- Acting
- Learning.

SEI CMMI is the basis for assessment.

2. <u>CMM-Based Appraisal for Internal Process Improvement(CBA IPI):</u>

It provides a diagnostic technique for assessing the relative maturity of a software organization. It also uses SEI CMMI as the basis for assessment.

3. <u>SPICE (ISO/IEC 15504):</u>

This standard defines a set of requirements for software process assessment.

4. <u>ISO 9001:2000 for software:</u>

It is a generic standard that applies to any organization.

The International Organization for Standardization (ISO) has developed the ISO 9001:2000 standard to define the requirements for a quality management system that will serve to produce higher quality products and thereby improve customer satisfaction.

ISO 9001:2000 has adopted a **"plan-do-check-act"** cycle.

<u>Plan</u> establishes the process objectives, activities and tasks necessary to achieve high quality software.

<u>Do</u> implements the software process.

<u>Check</u> monitors and measures the process to ensure that all requirements established for quality management have been achieved.

<u>Act</u> initiates software process improvement activities that continually work to improve the process.

Prescriptive Process Models:

Every software engineering organization should describe unique set of framework activities for the software process it adopts.

The framework activities should populate with set of software engineering actions and define each engineering actions in terms of task sets

The tasks (and degree of rigor) for each activity will vary based on:

- the type of project (an "entry point" to the model)
- nature of the project
- people who work on the project
- the environment in which the work will be conducted.

Process models are called to be **"prescriptive"** because they **prescribe**

1. a set of process elements – framework activities, software engineering actions, tasks, work products, quality assurance and change control mechanisms for each project.

2. a workflow – that is the manner in which the process elements are interrelated to one another.

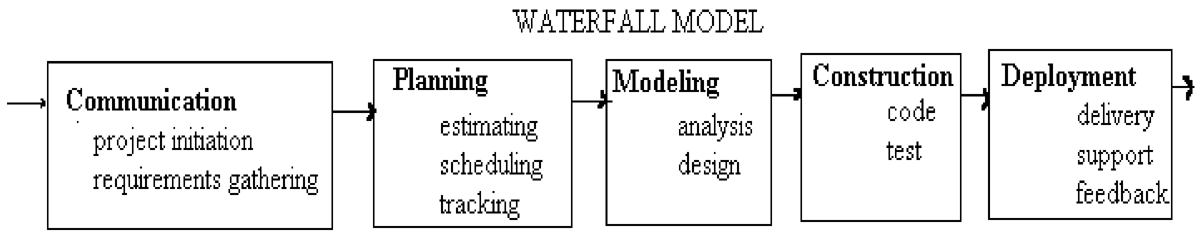
Software Process Models:

- Sequential/Linear Process Models
- Incremental Process Models
- Evolutionary Process Models

Sequential/Linear Process Models:

a) The Waterfall Model:

It is **also** known as Linear Sequential Model or Classic Life Cycle.



Communication: This framework activity involves heavy communication and collaboration with the customer and encompasses requirement gathering etc.

Planning: This activity establishes a plan for the technical tasks to be conducted, the risks that are likely to be occurred, the resources that will be required, the work products to be produced and a work schedule.

Modeling: This activity encompasses the creation of models and the design that will achieve the requirements.

i) Analysis

– Provide a clear definition of the require functionality, performance and interface. Focus is on Software.

– Documented and Reviewed with the Customer.

ii)Design

a. Translates requirements into a representation of software e.g., data structures, algorithms, interfaces etc.
b. Documented and Reviewed.
c. Serves as a quality check point before coding begins.

Construction: This activity combines code generation and and the resting that is required to uncover errors in the code.

Code Generation

– Translation of the design into machine executable code

– Could be mechanized

Testing

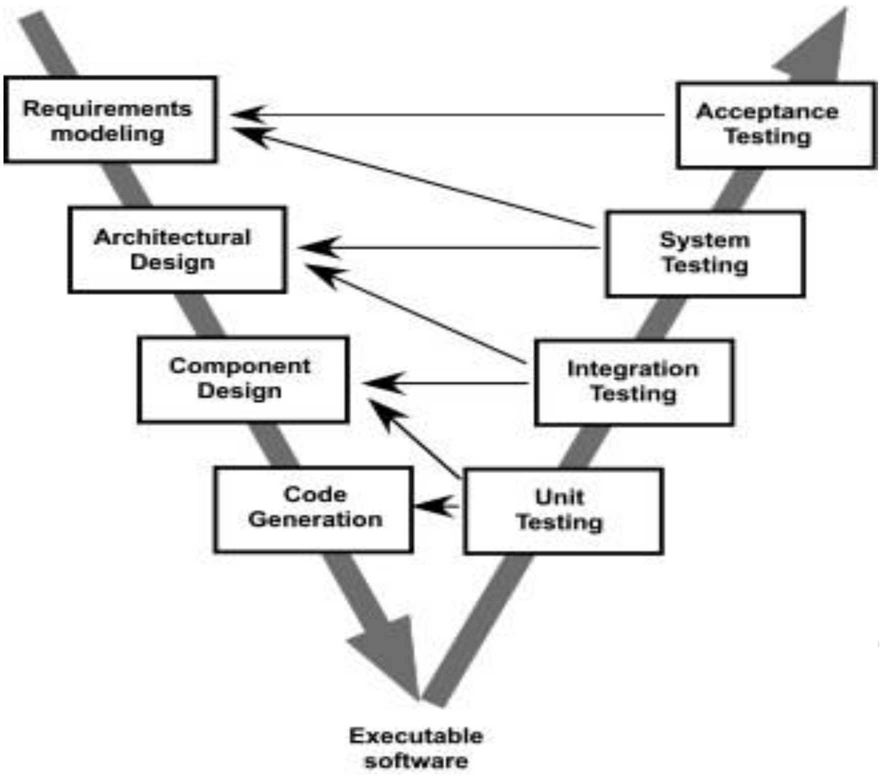
a. Check that all functional requirements are met.
b. Ensure that defined input will produce results that agree with required results.
c. Focus is on the logical internals of the software

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

Drawbacks or Issues:

- Requires that the customer state all requirements clearly upfront. In reality, it is very difficult.
- A working version of the software will not be available until late in the project life cycle.
- Difficult to accommodate changes in mid-stream as changes have to wait till the end of the current cycle.
- Leads to blocking states as some team members have to wait for the others to complete dependent tasks.

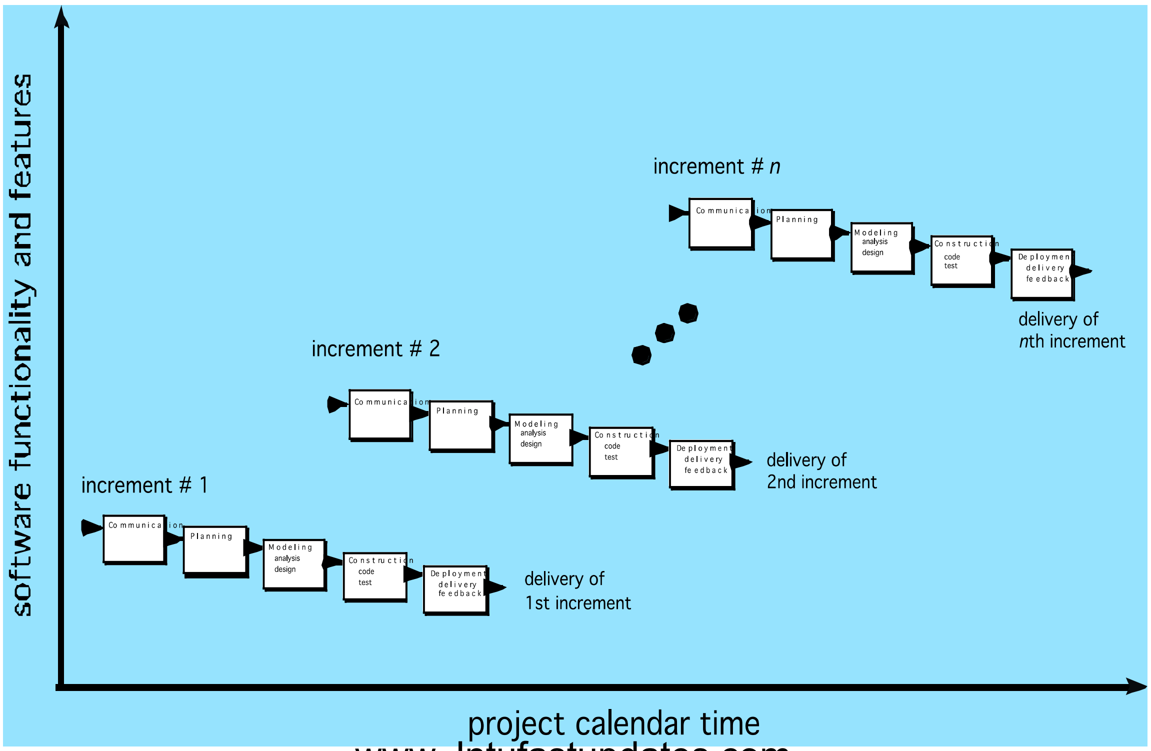
The V-Model



A variation of waterfall model depicts the relationship of quality assurance actions to the actions associated with communication, modeling and early code construction activates.
Team first moves down the left side of the V to refine the problem requirements. Once code is generated, the team moves up the right side of the V, performing a series of tests that validate each of the models created as the team moved down the left side.

Incremental Process Models:

Incremental Model:


software functionality and features
increment # n
Communication
Planning
Modeling analysis design
Construction code test
Deployment delivery feedback
delivery of nth increment
increment # 2
delivery of 2nd increment
increment # 1
delivery of 1st increment
project calendar time

- When initial requirements are reasonably well defined, but the overall scope of the development effort precludes a purely linear process. A compelling need to expand a limited set of new functions to a later system release.
- It combines elements of linear and parallel process flows. Each linear sequence produces deliverable increments of the software.
- Applies Linear Sequences in a staggered fashion resulting in multiple incremental deliverables
- The first increment is often a core product with many supplementary features. Users use it and evaluate it with more modifications to better meet the needs.
- Develops a plan based on feedback from the use of the product (both development and customer feedback) to fix problems, augment functionality in the next deliverable.

Suitable for

- Where there is short of skilled resources
- Where new technology is being used
- When business deadlines are aggressive

b) RAD Model:

- Rapid Application Development
- Emphasizes extremely short development cycle-60 to 90 days
- Incremental Software Development Process
- Uses Component Based Software Development
- High-speed adaptation of Linear Sequential Model
- Used primarily for Information Systems Applications
  - Students Academic Information
  - Library Information
- Suitable for applications that can be modularized (say one module for each function) and can be worked upon by multiple independent teams.

Business Modeling:

- Model the information flow among the business functions
  - Add a new Title to the Database
  - Issue a book to a member of the library,
  - Generate a report of all overdue books; send notifications to members who borrowed those books

Data Modeling:

- Define a set of tables, data structures, and data objects that support the data and business functions
- Identify the attributes of each data object
- Establish relationships between the data objects
  - e.g., Books table, Authors Table, Member Information Table

Process Modeling:

- Create process descriptions for operating on the data to fulfill business functions
- e.g., add, modify, delete, retrieve data from the tables

Application Generation:

- Employs automated tools to generate the code
- The tools generate template code where the programmer fills in the details
- e.g., user interfaces, objects that operate on the data

Testing:

- Overall testing time is smaller as fully tested components are used.
- Test new components, user interfaces and business functions.

Drawbacks:

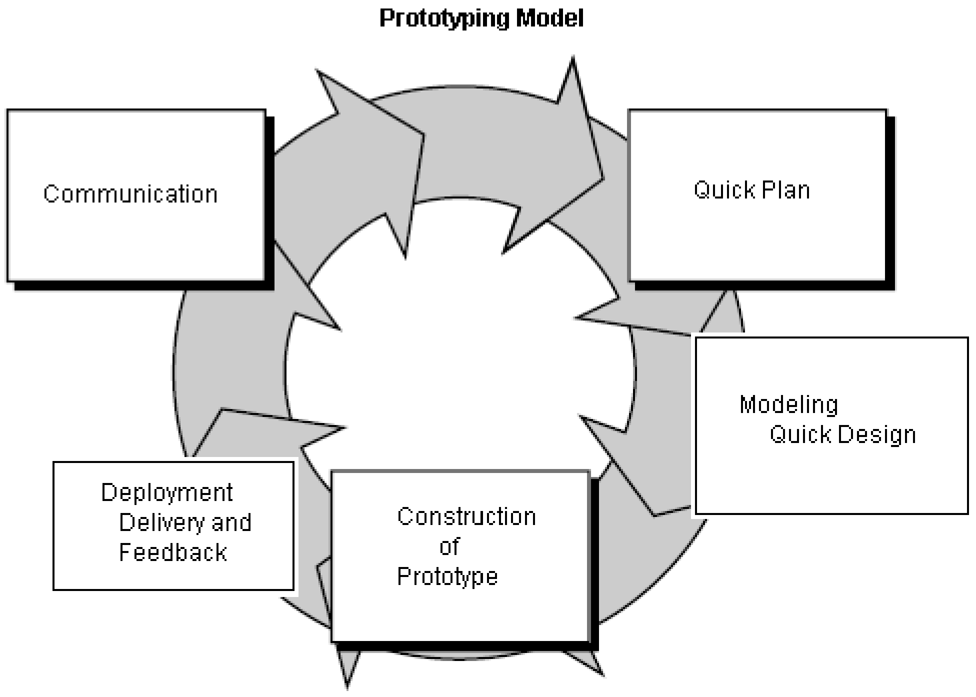
- Requires large manpower for larger projects to create sufficient number of RAD teams
- Requires high commitment to develop software in a shorter time span
- Not suitable when application relies on new technology and requires high degree of interoperability with existing software.

Evolutionary Models:

a) Prototyping Model:

The prototyping paradigm begins with communication. A “quick plan” and 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.
Two types of prototypes exist:

1. Throw away prototype
2. Evolutionary prototype

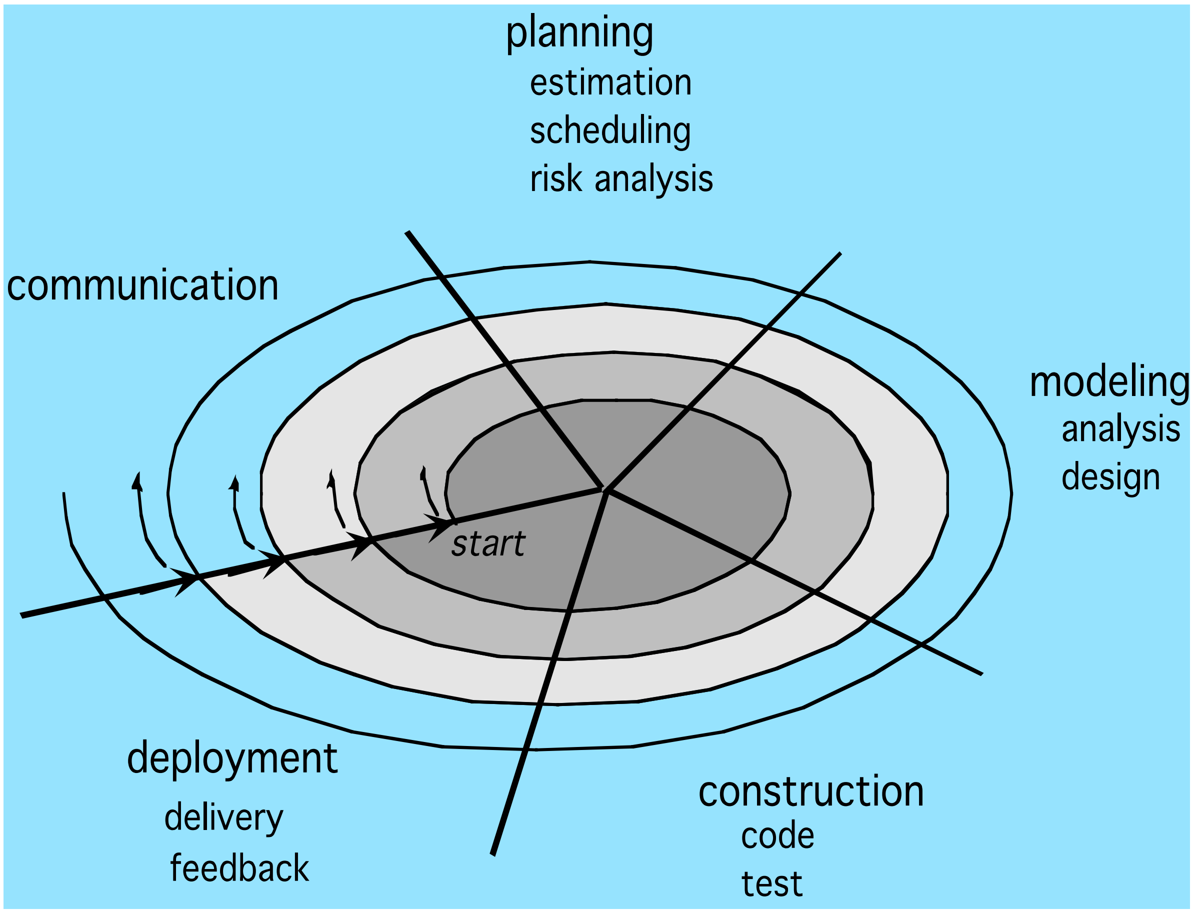


Prototyping can also be problematic for the following reasons:
1. The customer sees what appears to be a working version of the software, unaware that the prototype is held together unaware that in the rush to get it working no one has considered overall software quality or long-term maintainability. When informed that the
product must be rebuilt so that high levels of quality can be maintained, the customer cries foul and demands that "a few fixes" be applied to make the prototype a working product.
2. The developer often makes implementation compromises in order to get a prototype working quickly. After a time, the developer may become familiar with these choices and forget all the reasons why they were inappropriate.

The key is to define the rules of the game at the beginning; that is, the customer and developer must both agree that the prototype is built to serve as a mechanism for defining requirements. It is then discarded (at least in part) and the actual software is engineered with an eye toward quality and maintainability.

b) Spiral Model:

- Proposed by Boehm
- Combines the iterative nature of Prototype model with the Linear Sequential Model
- Increment releases from early iterations will mostly be either paper models or prototypes and later releases will deliver operational products.
- Divide into a number of activities – task regions
- Each region has a task set with a deliverable
- Can be adapted for the entire life of the software
- Supports multiple entry points
  - Concept development
  - New product development
  - Product enhancements
  - Product Maintenance
- A realistic approach to the development of large scale systems
- Not widely adapted as yet
- It couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model and is a risk-driven process model generator that is used to guide multi-stakeholder concurrent engineering of software intensive systems.
- Two main distinguishing features: one is cyclic approach for incrementally growing a system’s degree of definition and implementation while decreasing its degree of risk. The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.
- A series of evolutionary releases are delivered. During the early iterations, the release might be a model or prototype. During later iterations, increasingly more complete version of the engineered system are produced.
- The first circuit in the clockwise direction might result in the 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 results in adjustments to the project plan. Cost and schedule are adjusted based on feedback. Also, the number of iterations will be adjusted by project manager.
- Good to develop large-scale system as software evolves as the process progresses and risk should be understood and properly reacted to. Prototyping is used to reduce risk.
- However, it may be difficult to convince customers that it is controllable as it demands considerable risk assessment expertise.



Three Concerns on Evolutionary Processes

- First concern is that prototyping poses a problem to project planning because of the uncertain number of cycles required to construct the product.
- Second, it does not establish the maximum speed of the evolution. If the evolution occur too fast, without a period of relaxation, it is certain that the process will fall into chaos. On the other hand if the speed is too slow then productivity could be affected.
- Third, software processes should be focused on flexibility and extensibility rather than on high quality. We should prioritize the speed of the development over zero defects. Extending the development in order to reach high quality could result in a late delivery of the product when the opportunity niche has disappeared.

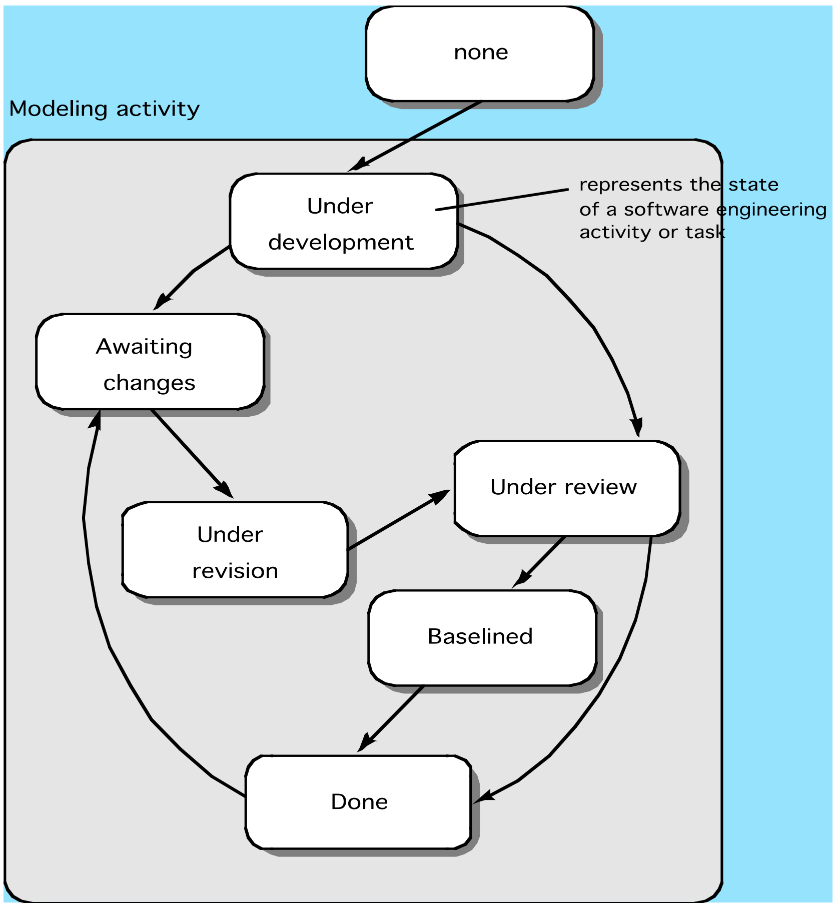
Concurrent Model:

The concurrent process model can be represented schematically as a series of major technical activities, tasks, and their associated states. It allow a software team to represent iterative and concurrent elements of any of the process models. For example, the modeling activity defined for the spiral model is accomplished by invoking one or more of the following actions: prototyping, analysis and design.

The following figure provides a schematic representation of one activity with the concurrent process model. The activity—modelling—may be in any one of the states10 noted at any given time.
All activities exist concurrently but reside in different states.

For example, early in a project the *customer communication* activity (not shown in the figure) has completed its first iteration and exists in the **awaiting changes** state. The modeling activity (which existed in the **none** state while initial customer communication was completed) now makes a transition into the **under development** state. If, however, the customer indicates that changes in requirements must be made, the modeling activity moves from the **under development** state into the **awaiting changes** state.

Concurrent modeling is applicable to all types of software development and provides an accurate picture of the current state of a project. Rather than confining software engineering activities, actions and tasks to a sequence of events, it defines a process network. Each activity, action or task on the network exists simultaneously with other activities, actions or tasks. Events generated at one point trigger transitions among the states.

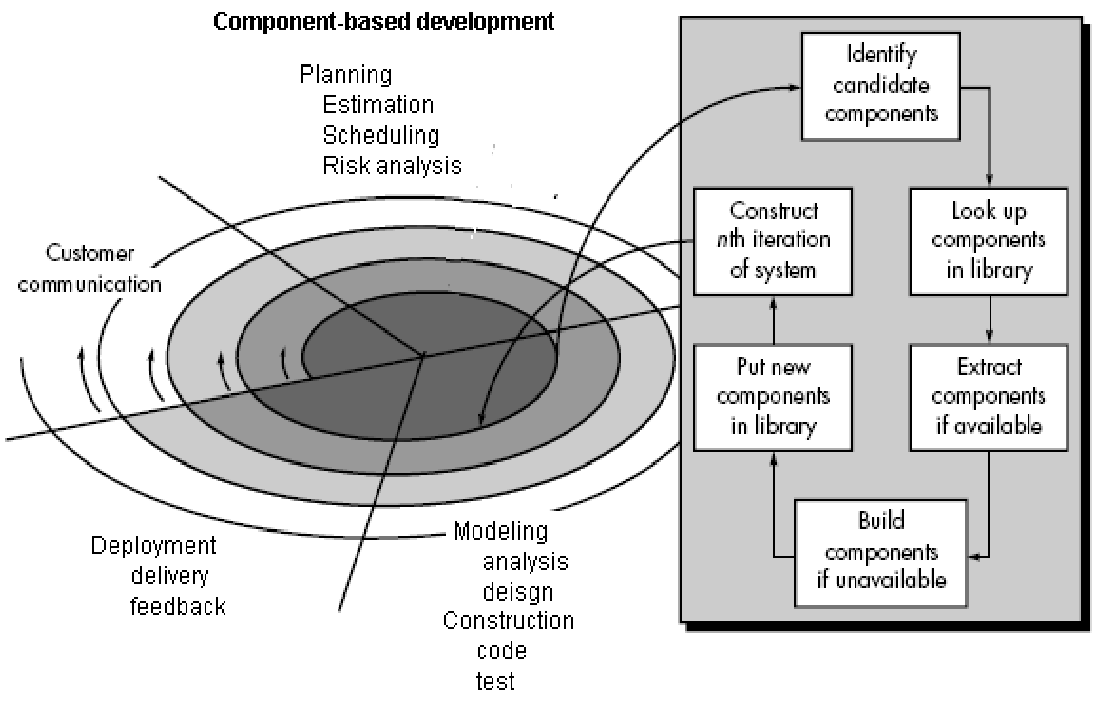


Specialized Process Models:

a) Component based development model:

The component-based development model incorporates many of the characteristics of the spiral model. It is evolutionary in nature.
This model composes applications from pre-packaged software components.
It incorporates the following steps:

1. Available component-based products are researched and evaluated for the application domain.
2. Component issues are considered.
3. Software architecture is designed to accommodate the components.
4. Components are integrated into the architecture.
5. Comprehensive testing is conducted to ensure proper functionality.


Component-based development
Planning
Estimation
Scheduling
Risk analysis
Customer communication
Deployment
delivery
feedback
Modeling
analysis
deisgn
Construction
code
test
Identify candidate components
Look up components in library
Extract components if available
Build components if unavailable
Put new components in library
Construct nth iteration of system

The component based development model leads to software reuse. It provides software engineers with a number of measurable benefits.

b) The Formal Method 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 clean room software engineering.

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 following concerns about its applicability in a business environment:

- The development of formal models is currently quite time consuming and expensive.
- Because few software developers have the necessary background to apply formal methods, extensive training is required.
- It is difficult to use the models as a communication mechanism for technically unsophisticated customers.
- These concerns notwithstanding, it is likely that the formal methods approach will gain adherents among software developers who must build safety-critical software for e.g., developers of aircraft avionics and medical devices.

c) Aspect-Oriented Software Development:

Aspect-oriented software development (AOSD), referred to as aspect-oriented programming (AOP) is relatively new software engineering paradigm that provides a process and methodological approach for defining, specifying, designing and constructing aspects.

As modern computer-based systems become more sophisticated certain concerns i.e.., customer required properties or areas of technical interest span the entire architecture.

- Some concerns are high-level properties of the system. Eg: Security, Fault tolerance etc.

- Other concerns affect functions. Eg: The application of business rules.
- And other concerns are systemic. Eg: Task synchronization or Memory management.

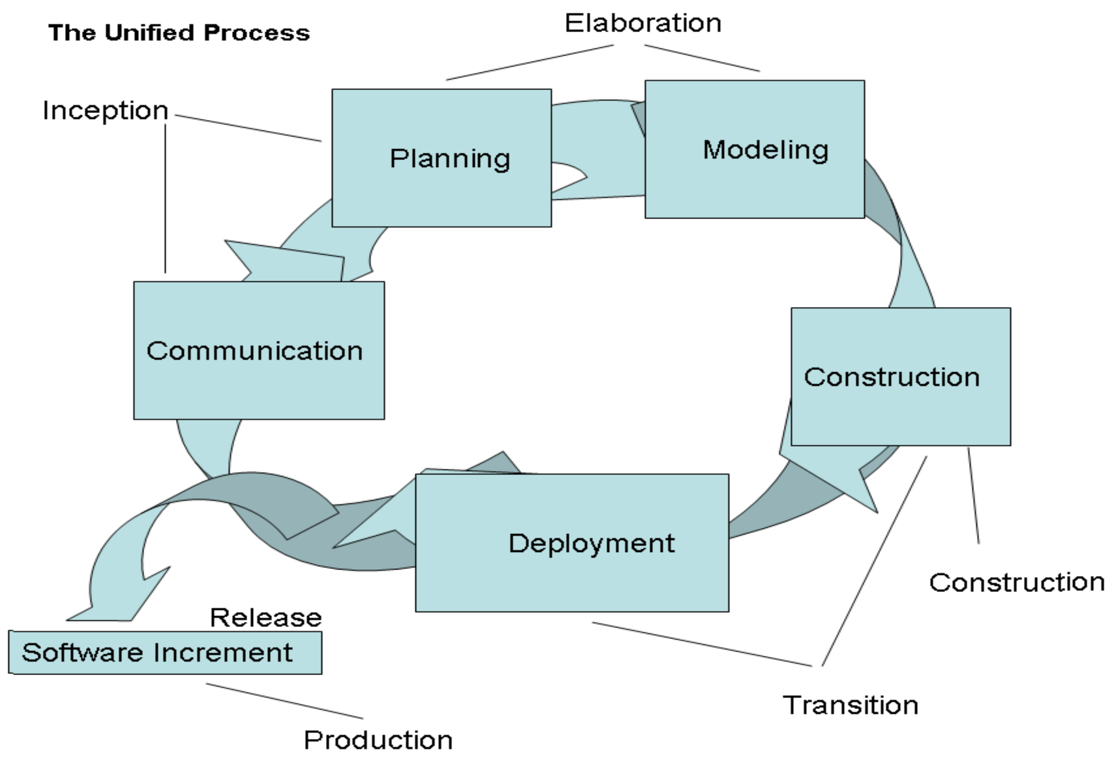
When concerns cut across multiple system functions, features and information, they are referred to as crosscutting concerns. Aspectual requirements define those crosscutting concerns.
This process will adopt characteristics of both the spiral and concurrent process model. The evolutionary nature of the spiral is appropriate as aspects are identified and then constructed. The parallel nature of concurrent development is essential because aspects are engineered independently of localized software components and yet aspects have a direct impact on these components.

The Unified Process:

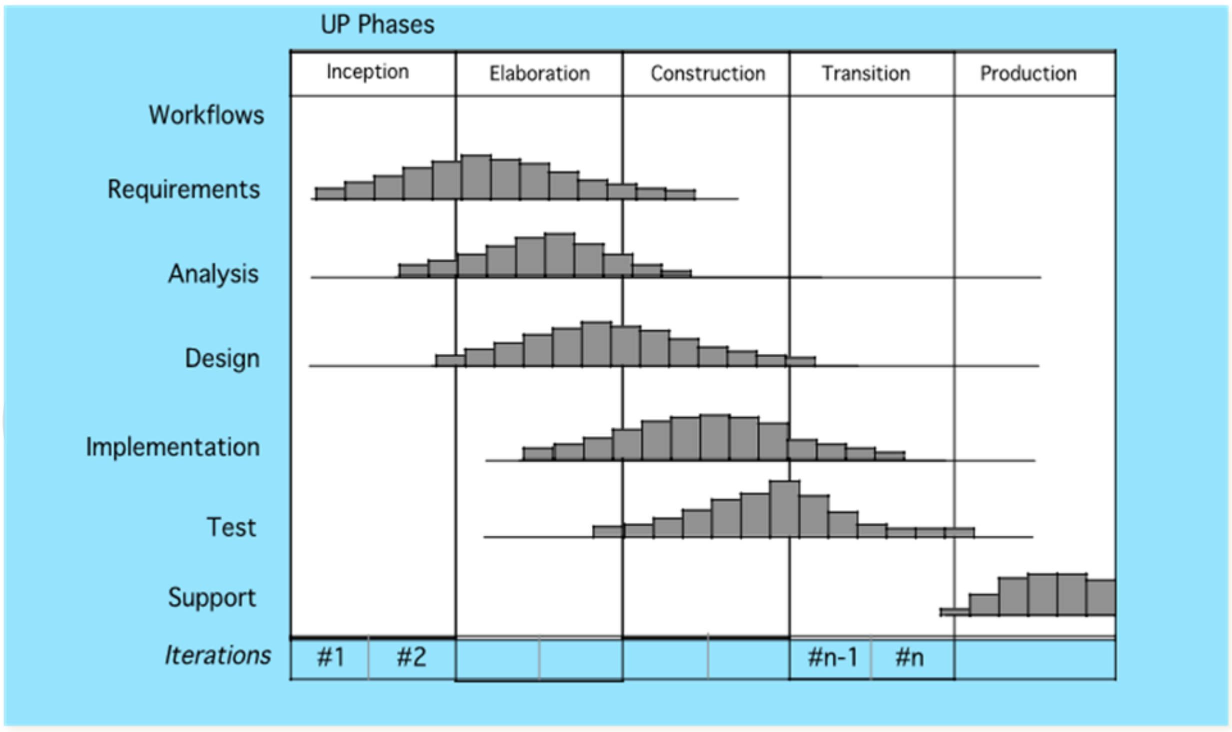
The Unified Process (UP) is an attempt to draw on the best features and characteristics of conventional software process models. The UP recognizes the importance of customer communication and streamlined methods for describing the customer's view of a system through use-cases. A use-case is a text narrative or template that describes a system function or feature from the user's (or actor's) point of view. The UP came about as a result of the need to meet a growing demand for bigger and more complex systems.

The UP consists of five (5) phases which are similar to the five generic phases discussed earlier. These phases are: Inception, Elaboration, Constructions, Transition and Production.

Phase Name	**Description**
Inception Phase	Encompasses both customer communication and planning activities. Through collaboration with customer and end-users, business requirements for the software are identified through the development of use-cases, a rough architecture is proposed and a plan for iterative, incremental nature of the project is developed.
Elaboration Phase	At this stage the preliminary use-cases developed in previous phase are refined and expanded. In addition the architecture and plans are refined to ensure that the scope, risks, and delivery times remain reasonable.
Construction Phase	Using the architectural model as input, this phase of the process develops or acquires the software components that will make each use-case operational for end-users. As these components are being developed, unit tests are designed and executed for each. The different components are also integrated and tested.
Transition Phase	At this stage, the software is given to users for testing with the intent of uncovering defects and deficiencies. A defect/deficiency reporting scheme is established, and the software team assesses the feedback. Also at this stage, user manuals, trouble-shooting guides, and installation procedures are produced.
Production Phase	During this phase, the on-going use of the software is monitored, support is provided and defect reports and requests for changes are submitted and evaluated.



Unified Process Phases



Personal and Team Process Models:

Watts Humphrey suggests that it is possible to create a personal software process and a team software process. Both require hard work, training and coordination, but both are achievable.

Personal Software Process (PSP):

1. Personal software process(PSP) emphasizes personal measurement of both the work product that is produced and the resultant quality of the work product.
2. PSP makes the practitioner responsible for project planning and controlling the quality of all software work products that are developed.
3. PSP process model defines five framework activities:
   - Planning
   - High-Level Design
   - High-Level design review
   - Development
   - Postmortem

Planning: This activity isolates requirements. Based on these, size and resource estimates, defect estimates etc. are made. All metrics are recorded, development tasks are identified and a project schedule is crated.

High-Level Design: External Specifications for each component are developed and a component design is created.

High-Level Design review: Formal verification methods are applied to uncover errors in the design.

Development: The component level design is refined and reviewed. Code is generated, reviewed, complied and tested.

Postmortem: using measures and metrics the effectiveness of the process is determined. Measures and metrics should provide guidance for modifying the process to improve its effectiveness.

PSP has not been widely adopted throughout the industry. The reason is PSP is intellectually challenging and demands a level of commitment by practitioners and manager that is not always possible to obtain.

Team Software Process (TSP):

The goal of TSP is to build “a self-directed” project team the organizes itself to produce high-quality software.

The following are the objectives for TSP:

- Build self-directed teams that plan and track their work, establish goals and own their processes and plans.
- Show managers how to coach and motivate their teams and how to help them.
- Accelerate software process improvement
- Provide improvement guidance to organizations.
- Facilitate University teaching.

TSP defines the following framework activities:

- Launch
- High-Level Design
- Implementation
- Integration and Test
- Postmortem.

These activities enable the tem to plan, design and construct software in a disciplined manner. The postmortem sets the stage for process improvements.

TSP makes use of a wide variety of scripts, forms and standards that serve to guide team members in their work.

Scripts define specific process activities i.e., project launch, design, implementation, integration & testing and postmortem and other more detailed work functions like development planning, requirements development, software configuration management and unit test.

****