UNIT I

SOFTWARE PROCESS MODELS

The Evolving role of Software – Software – The changing Nature of Software – Legacy software — A generic view of process– A layered Technology – A Process Framework – The Capability maturity Model Integration (CMMI) – Process Assessment – Personal and Team Process Models – Product and Process – Process Models – The Waterfall Model – Incremental Process Models – Incremental Model – The RAD Model – Evolutionary Process Models – Prototyping – The Spiral Model – The Concurrent Development Model – Specialized Process Models – the Unified Process.

THE EVOLVING ROLE OF SOFTWARE

Nowadays, software plays a major role with dual activity. It is a product like a vehicle. As a product, it delivers the computing potential embodied by computer hardware or a network of computers that are accessible by local hardware. Whether the product or software resides within a mobile phone or operates inside a mainframe computer, software is an information transformer likely to produce, manage, acquire, modify, display or transmit the information.

Dual role of Software

- A Product
  - Information transformer- producing, managing and displaying
- A Vehicle for delivering a product
  - Control of computer (operating system),the communication of information (networks) and the creation of other programs

The software

- provides good product with useful information
- transforms the data so that it can be more useful in a local context
- manages business information to enhance competitiveness
- provides a gateway to worldwide networks like internet

The role of computer software has undergone significant change over a time span of little more than 50 years.

The software has seen many changes since its inception. After all, it has evolved over the period of time against all odds and adverse circumstances. Computer industry has also progressed at a break-neck speed through the computer revolution, and recently, the network revolution

triggered and/or accelerated by the explosive spread of the internet and most recently the web. Computer industry has been delivering exponential improvement in price performance, but the problems with software have not been decreasing. Software still come late, exceed budget and are full of residual faults. As per the latest IBM report, "31% of the projects get cancelled before they are completed, 53% over-run their cost estimates by an average of 189% and for every 100 projects, there are 94 restarts" .

SOFTWARE

Software is defined as

Instructions

- Programs that when executed provide desired function

Data structures

-Enable the programs to adequately manipulate information

Documents

-Describe the operation and use of the programs.

Definition of Engineering

-Application of science, tools and methods to find cost effective solution to problems

Definition of SOFTWARE ENGINEERING

- SE is defined as systematic, disciplined and quantifiable approach for the development, operation and maintenance of software

Software is the set of instructions encompasses programs that execute within a computer of any size and architecture, documents that encompass hard-copy and virtual forms, and data that combine numbers and text. It also includes representations of pictorial, video, and audio information. Software engineers can build the software and virtually everyone in the industrialized world uses it either directly or indirectly. It is so important because it affects nearly every aspect of our lives and has become pervasive in our commerce, our culture, and our everyday activities. The steps to build the computer software is as the user would like to build any successful product, by applying a process that leads to a high-quality result that meets the needs of the people who will use the product. From the software engineer's view, the product is may be the programs, documents, and data that are computer software. But from the user's viewpoint, the product is the resultant information that somehow makes the user's world better. Software's impact on the society and culture continues to be profound. As its importance grows, the software community continually attempts to develop technologies that will make it easier, faster, and less expensive to build high-quality computer programs. Some of

these technologies are targeted at a specific application domain like web-site design and implementation; others focus on a technology domain such as object oriented systems and still others are broad-based like operating systems such as LINUX. However, a software technology has to develop useful information. The technology encompasses a process, a set of methods, and an array of tools called as software engineering.

THE CHANGING NATURE OF SOFTWARE

Seven Broad Categories of software are challenges for software engineers

- System software
- Application software
- Engineering and scientific software
- Embedded software
- Product-line software
- Web-applications
- Artificial intelligence software

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

- Embedded software
  -- resides in read-only memory
  --is used to control products and systems for the consumer and industrial markets.
- 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
- Engineering and scientific software. Engineering and scientific software have been characterized by "number crunching" algorithms

LEGACY SOFTWARE

Legacy software are older programs that are developed decades ago. The quality of legacy software is poor because it has inextensible design, convoluted code, poor and nonexistent documentation, test cases and results that are not achieved.

- Support core business functions
- Have longevity and business criticality

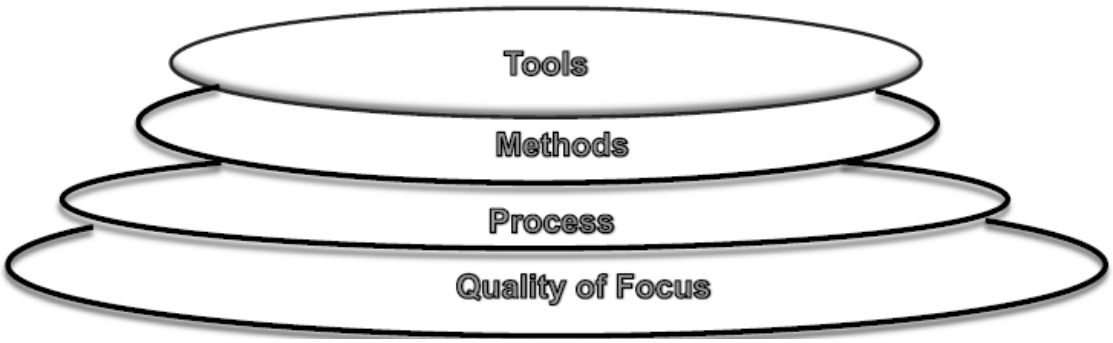
- Exhibit poor quality
  - Convoluted code, poor documentation, poor testing, poor change management

As time passes legacy systems evolve due to following reasons:

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

A GENERIC VIEW OF PROCESS– A LAYERED TECHNOLOGY

- Software engineering encompasses a process, the management of activities, technical methods, and use of tools to develop software products.
- Fritz Bauer defined Software engineering as the "establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines. "
- IEEE definition of software engineering (1) the application of a 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 as in (1).
- We need discipline but we also need adaptability and agility.
- Software Engineering is a layered technology as shown below. Any engineering approach must rest on an organizational commitment to quality.
- The bedrock that supports software engineering is a quality focus.



The foundation for S/W eng is the process layer. It is the glue that holds the technology layers together and enables rational and timely development of computer S/W.

- Process defines a framework that must be established for effective delivery of S/W eng technology.

- The software process forms the basis for management control of software projects and establishes the context in which technical methods are applied, work products (models, documents, data, reports, etc.) are produced, milestones are established, quality is ensured, and change is properly managed.

- S/W eng methods provide the technical "how to's" for building S/W. Methods encompass a broad array of tasks that include communication, req. analysis, design, coding, testing and support.

- S/W eng tools provide automated or semi-automated support for the process and the methods.
- When tools are integrated so that info. Created by one tool can be used by another, a system for the support of S/W development called computer-aided software engineering is established.

A PROCESS FRAMEWORK

Software process models can be prescriptive or agile, complex or simple, all-encompassing or targeted, but in every case, five key activities must occur. The framework activities are applicable to all projects and all application domains, and they are a template for every process model.

- **Software process**
    - **Process framework**
        - **Umbrella activities**
            - **Framework activity #1**
                - **Software Engineering action**

Each framework activity is populated by a set of S/W eng actions – a collection of related tasks that produces a major S/W eng work product (design is a S/W eng action). Each action is populated with individual work tasks that accomplish some part of the work implied by the action.

The following generic process framework is applicable to the vast majority of S/W projects.

- **Communication:** involves heavy communication with the customer (and other stakeholders) and encompasses requirements gathering.
- **Planning:** Describes the technical tasks to be conducted, the risks that are likely, resources that will be required, the work products to be produced and a work schedule.
- **Modeling:** encompasses the creation of models that allow the developer and customer to better understand S/W req. and the design that will achieve those req.
- **Construction:** combines code generation and the testing required uncovering errors in the code.
- **Deployment:** deliver the product to the customer who evaluates the delivered product and provides feedback.

Each S/W eng action is represented by a number of different task sets – each a collection of S/W eng work tasks, related work products, quality assurance points, and project milestones. The task set that best accommodates the needs of the project and the characteristics of the team is chosen.

The framework described in the generic view of S/W eng is complemented by a number of **umbrella activities**. Typical activities include:

- **S/W project tracking and control:** allows the team to assess progress against the project plan and take necessary action to maintain schedule.
- Risk Management: Assesses the risks that may affect the outcome of the project or the quality.
- **Software quality assurance:** defines and conducts the activities required to ensure software quality.
- **Formal Technical Review:** uncover and remove errors before they propagate to the next action.
- **Measurement:** defines and collects process, project, and product measures that assist the team in delivering S/W that meets customers' needs.

- **Software configuration management:** Manages the effect of change throughout the S/W process
- **Reusability management:** defines criteria for work product reuse.
- **Work product preparation and production:** encompasses the activities required to create work products such as models, documents, etc.

THE CAPABILITY MATURITY MODEL INTEGRATION

- The Software Engineering Institute (SEI) has developed a comprehensive process meta-model that is predicated on a set of system and software eng capabilities that should be present as organizations reach different levels of process capability and maturity.
- The CMMI defines each process area in terms of "specific goals" and the "specific practices" required to achieve these goals.
- Specific goals establish the characteristics that must exist if the activities implied by a process area are to be effective.
    - Specific practices refine a goal into a set of process-related activities.

The CMMI is from the Software Engineering Institute (SEI)

• Level 0: Incomplete (process is not performed or does not achieve all goals defined for

• Level 1: Performed (work tasks required to produce required work products are being conducted)

• Level 2: Managed (people doing work have access to adequate resources to get job done, stakeholders are actively involved, work tasks and products are monitored, reviewed, and evaluated for conformance to process description)

• Level 3: Defined (management and engineering processes documented, standardized, and integrated into organization-wide software process)

• Level 4: Quantitatively Managed (software process and products are quantitatively understood and controlled using detailed measures)

• Level 5: Optimizing (continuous process improvement is enabled by quantitative feedback from the process and testing innovative ideas)

PROCESS ASSESSMENT

The process should be assessed to ensure that it meets a set of basic process criteria that have been shown to be essential for a successful software engineering. This is used by industry professionals.

The PSP model is good from the perspective that an individual software engineer can use it to improve his or her personal productivity and work product quality. Both models are largely iterative or evolutionary in their approach to software development. PSP and TSP are interesting, but are not pivotal to an understanding of process issues. A key point of this section is that individuals and teams should measure their work and the errors they make and act to improve their approach so that the causes of errors are eliminated.

PERSONAL AND TEAM SOFTWARE PROCESS

The Personal Software Process (PSP) model is good from the perspective that an individual software engineer can use it to improve his or her personal productivity and work product quality.

PSP process model defines five framework activities: planning, high-level design, high-level design review, development, and postmortem. It stresses the need to identify errors early and to understand the types of errors.

Planning: it isolates reqs. And a project schedule is created.

High-level design: Prototypes are built when uncertainty exists.

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

Development: Code is generated, reviewed, compiled, and tested.

Postmortem: using the measures and metrics collected, the effectiveness of the process is determined.

TEAM SOFTWARE PROCESS (TSP):

The goal of TSP is to build a “self-directed” project team that organizes itself to produce high-quality s/w.

Each project is “launched” using a “script” that defines the tasks to be accomplished.

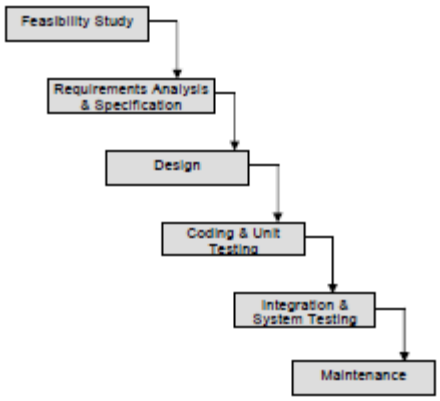
Teams are self-directed.

Measurement is encouraged.

Measures are analyzed with the intent of improving the team process.

PROCESS MODELS

- **The Waterfall Model**



The classical waterfall model is intuitively the most obvious way to develop software. Though the classical waterfall model is elegant and intuitively obvious, it is not a practical model in the sense that it can not be used in actual software development projects. Thus, this model can be considered to be a *theoretical way of developing software*. But all other life cycle models are essentially derived from the classical waterfall model. So, in order to be able to appreciate other life cycle models it is necessary to learn the classical waterfall model.

Classical waterfall model divides the life cycle into the following phases as shown in fig.

Feasibility Study

The main aim of feasibility study is to determine whether it would be financially and technically feasible to develop the product.

 At first project managers or team leaders try to have a rough understanding of what is required to be done by visiting the client side. They study different input data to the system and output data to be produced by the system. They study what kind of processing is needed to be done on these data and they look at the various constraints on the behavior of the system.

 After they have an overall understanding of the problem they investigate the different solutions that are possible. Then they examine each of the solutions in terms of what kind of resources required, what would be the cost of development and what would be the development time for each solution.

 Based on this analysis they pick the best solution and determine whether the solution is feasible financially and technically. They check whether the customer budget would meet the cost of the product and whether they have sufficient technical expertise in the area of development.

Requirements Analysis and Specification

The aim of the requirements analysis and specification phase is to understand the exact requirements of the customer and to document them properly. This phase consists of two distinct activities, namely

 Requirements gathering and analysis, and

 Requirements specification

The goal of the requirements gathering activity is to collect all relevant information from the customer regarding the product to be developed. This is done to clearly understand the customer requirements so that incompleteness and inconsistencies are removed.

The requirements analysis activity is begun by collecting all relevant data regarding the product to be developed from the users of the product and from the customer through interviews and discussions. After all ambiguities, inconsistencies, and incompleteness have been resolved and all the requirements properly understood, the requirements specification activity can start. During this activity, the user requirements are systematically organized into a Software Requirements Specification (SRS) document.

The customer requirements identified during the requirements gathering and analysis activity are organized into a SRS document. The important

components of this document are functional requirements, the nonfunctional requirements, and the goals of implementation.

Design

The goal of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the software architecture is derived from the SRS document. Two distinctly different approaches are available: the traditional design approach and the object-oriented design approach.

 Traditional design approach

Traditional design consists of two different activities; first a structured analysis of the requirements specification is carried out where the detailed structure of the problem is examined. This is followed by a structured design activity. During structured design, the results of structured analysis are transformed into the software design.

 Object-oriented design approach

In this technique, various objects that occur in the problem domain and the solution domain are first identified, and the different relationships that exist among these objects are identified. The object structure is further refined to obtain the detailed design.

Coding and Unit Testing

The purpose of the coding and unit testing phase (sometimes called the implementation phase) of software development is to translate the software design into source code. Each component of the design is implemented as a program module. The end-product of this phase is a set of program modules that have been individually tested.

During this phase, each module is unit tested to determine the correct working of all the individual modules. It involves testing each module in isolation as this is the most efficient way to debug the errors identified at this stage.

Integration and system testing: -

Integration of different modules is undertaken once they have been coded and unit tested. During the integration and system testing phase, the modules are integrated in a planned manner. The different modules making up a software product are almost never integrated in one shot. Integration is normally carried out incrementally over a number of steps. During each integration step, the partially integrated system is tested and a set of previously planned modules are added to it. Finally, when all the modules have been successfully integrated and tested, system testing is carried out. The goal of system testing is to ensure that the developed system conforms to its requirements laid out in the SRS document. System testing usually consists of three different kinds of testing activities:

- α – testing: It is the system testing performed by the development team.
- β – testing: It is the system testing performed by a friendly set of customers.
- acceptance testing: It is the system testing performed by the customer himself after the product delivery to determine whether to accept or reject the delivered product.

Maintenance

Maintenance of a typical software product requires much more than the effort necessary to develop the product itself. Many studies carried out in the past confirm this and indicate that the relative effort of development of a typical software product to its maintenance effort is roughly in the 40:60 ratio.

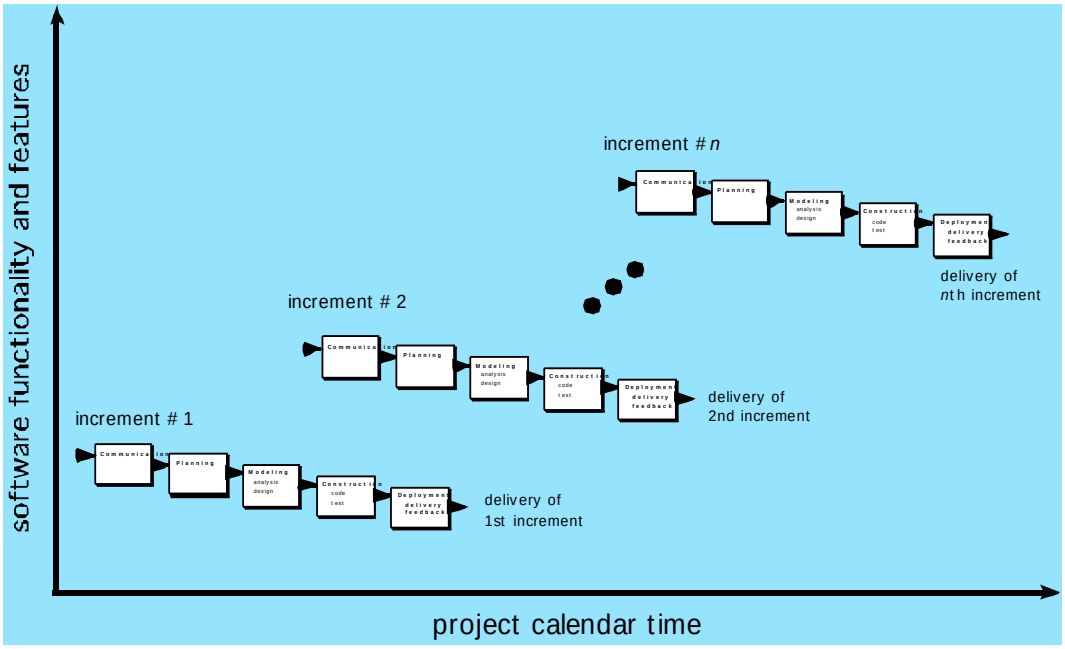
Maintenance involves performing any one or more of the following three kinds of activities:

- Correcting errors that were not discovered during the product development phase. This is called **corrective maintenance.**
- Improving the implementation of the system, and enhancing the functionalities of the system according to the customer's requirements. This is called **perfective maintenance**.
- Porting the software to work in a new environment. For example, porting may be required to get the software to work on a new computer platform or with a new operating system. This is called **adaptive maintenance**.

Incremental Process Models

The process models in this category tend to be among the most widely used (and effective) in the industry.

a. The Incremental Model



- The *incremental model* combines elements of the *waterfall* model applied in an iterative fashion. The model applies linear sequences in a staggered fashion as calendar time progresses.

- Each linear sequence produces deliverable "increments" of the software. (Ex: a Word Processor delivers basic file mgmt., editing, in the first increment; more sophisticated editing, document production capabilities in the 2nd increment; spelling and grammar checking in the 3rd increment.

- When an increment model is used, the 1st increment is often a *core product*. The core product is used by the customer.

- As a result of use and / or evaluation, a plan is developed for the next increment.

- The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality.

- The process is repeated following the delivery of each increment, until the complete product is produced.

- If the customer demands delivery by a date that is impossible to meet, suggest delivering one or more increments by that date and the rest of the Software later.

b. The RAD Model

Rapid Application Development (*RAD*) is an incremental software process model that emphasizes a short development cycle.

RAD is a "high-speed" adaptation of the waterfall model, in which rapid development is achieved by using a component based construction approach.

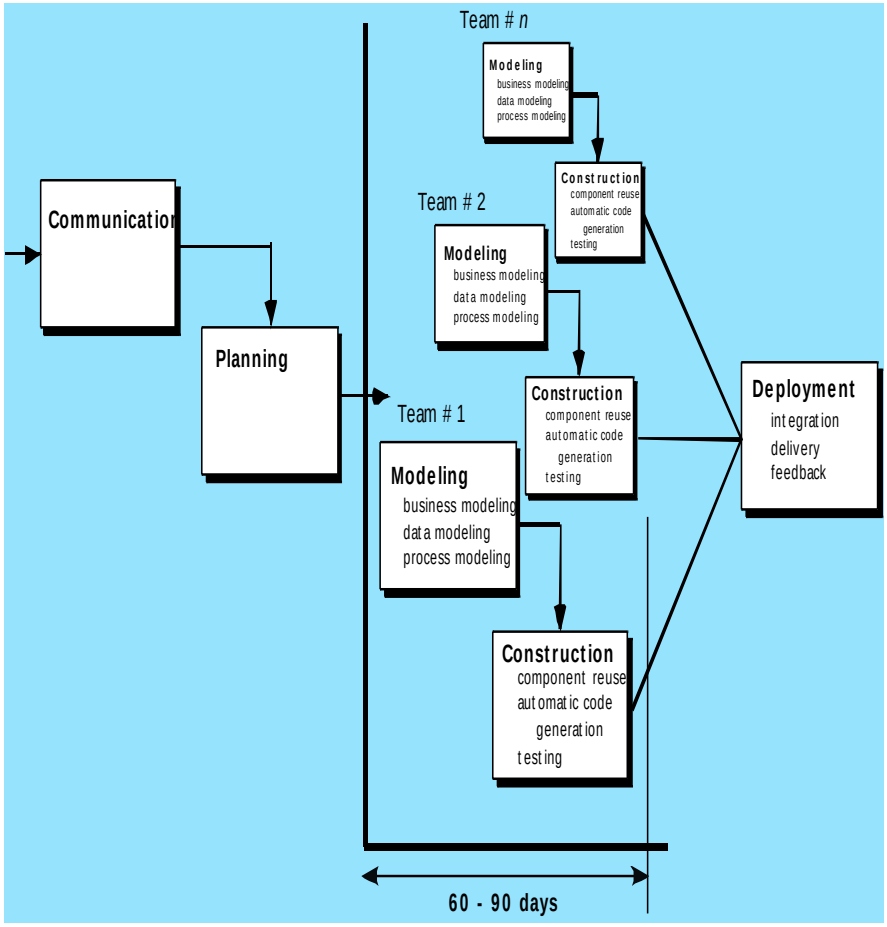
If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a fully functional system within a short period of time.

What are the drawbacks of the RAD model?

1. For large, but scalable projects, RAD requires sufficient human resources to create the right number of RAD teams.

2. If developers and customers are not committed to the rapid-fire activities necessary to complete the system in a much abbreviated time frame, RAD project will fail.

3. If a system cannot properly be modularized, building the components necessary for RAD will be problematic.

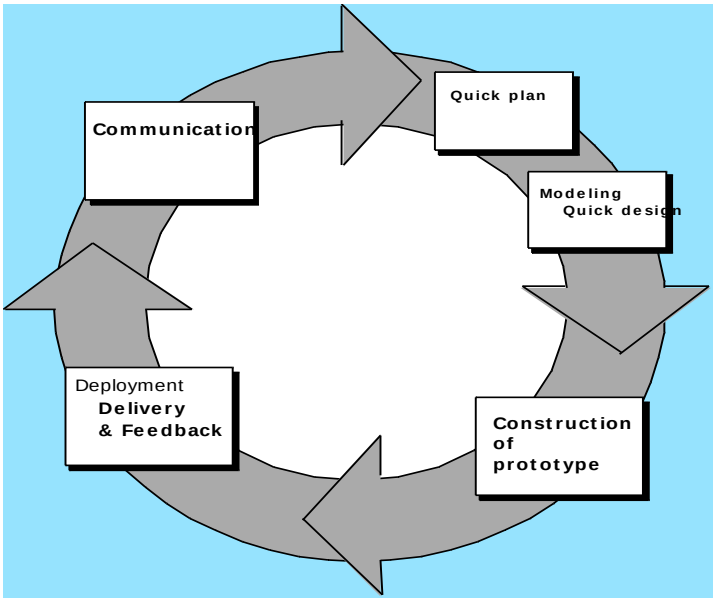


EVOLUTIONARY PROCESS MODELS

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. Software Engineering needs a process model that has been explicitly designed to accommodate a product that evolves over time. Evolutionary process models are iterative. They produce increasingly more complete versions of the Software with each iteration.

a. **Prototyping**

Customers often define a set of general objectives for Software, but doesn't identify detailed input, processing, or input requirements. Prototyping paradigm assists the Software engineering and the customer to better understand what is to be built when requirements are fuzzy.



The prototyping paradigm begins with *communication* where requirements and goals of Software are defined. Prototyping iteration is *planned* quickly and modeling in the form of quick design occurs. The *quick design* focuses on a representation of those aspects of the Software that will be visible to the customer "Human interface". The quick design leads to the *Construction of the Prototype*.

The prototype is *deployed* and then *evaluated* by the customer. *Feedback* is used to refine requirements for the Software. Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while enabling the developer to better understand what needs to be done. The prototype can serve as the "first system". Both customers and developers like the prototyping paradigm as users get a feel for the actual system, and developers get to build Software immediately. Yet, prototyping can be problematic:

1. The customer sees what appears to be a working version of the Software, unaware that the prototype is held together "with chewing gum. "Quality, long-term maintainability." When informed that the product is a prototype, the customer cries foul and demands that few fixes be applied to make it a working product. Too often, Software development management relents.

2. The developer makes implementation compromises in order to get a prototype working quickly. An inappropriate O/S or programming language used simply b/c it's available and known. After a time, the developer may become comfortable 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. The customer and the developer must both agree that the prototype is built to serve as a mechanism for defining requirements.

b. **The Spiral Model**

The spiral model is an evolutionary Software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model.

It has two distinguishing features:

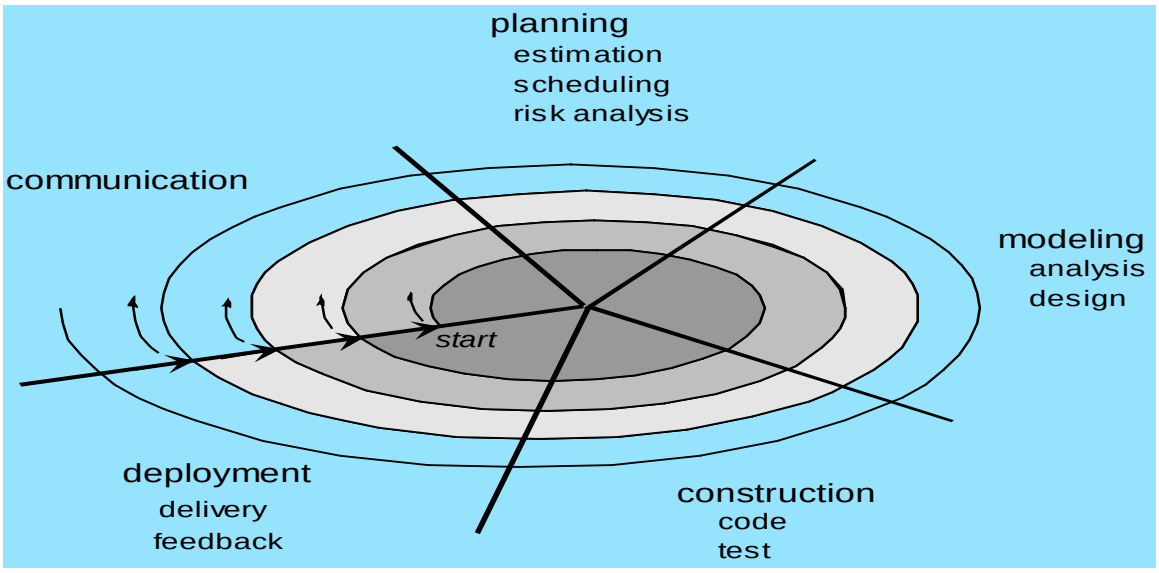
a. A cyclic approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk.

b. A set of *anchor point milestones* for ensuring stakeholder commitment to feasible and mutually satisfactory solutions.

Using the spiral model, Software is developed in a series of evolutionary releases. During early stages, the release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced. A spiral model is divided into a set of framework activities divided by the Software engineering team. As this evolutionary process begins, the Software team performs activities that are implied by a circuit around the spiral in a clockwise direction, beginning at the center. Risk is considered as each revolution is made.

Anchor-point milestones – a combination of work products and conditions that are attained along the path of the spiral- are noted for each evolutionary pass.

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 the customer after delivery. Unlike other process models that end when Software is delivered, the spiral model can be adapted to apply throughout the life of the Software.



THE CONCURRENT DEVELOPMENT MODEL

The *concurrent development model*, sometimes called *concurrent engineering*, can be represented schematically as a series of framework activities, Software engineering actions of tasks, and their associated states. The concurrent model is often more appropriate for system engineering projects where different engineering teams are involved.

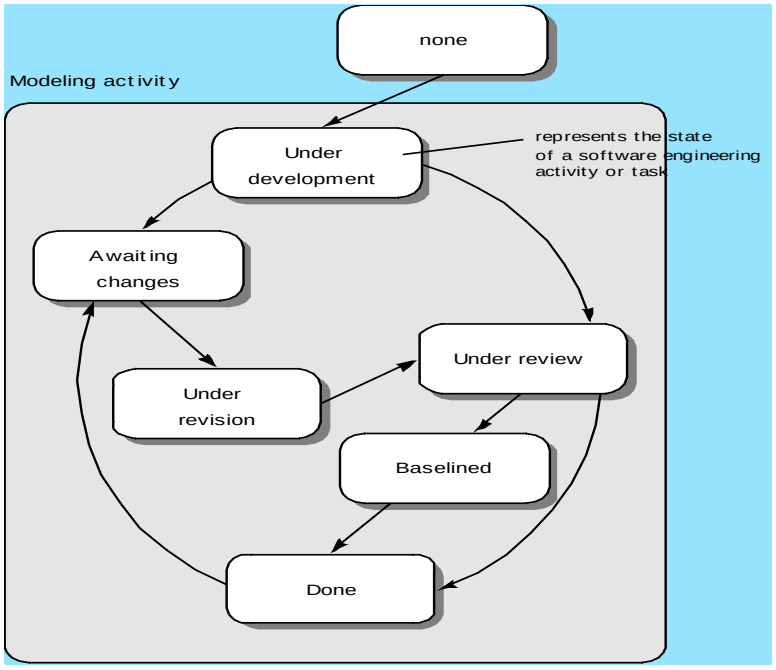


Figure above provides a schematic representation of one Software engineering task within the modeling activity for the concurrent process model. The activity – modeling- may be in any one of the states noted at any given time. All activities exist concurrently but reside in different states. For example, early in the project the *communication* activity has completed its first iteration and exists in the **awaiting changes** state. The *modeling* activity which existed in the **none** state while initial communication was completed now makes a transition into **underdevelopment** state. If, however, the customer indicates the changes in requirements must be made, the *modeling* 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, actions, or tasks.

SPECIALIZED PROCESS MODELS

a. Component Based Development

Commercial off-the-shelf (COTS) Software components, developed by vendors who offer them as products, can be used when Software is to be built. These components provide targeted functionality with well-defined interfaces that enable the component to be integrated into the Software. The *component-based development* model incorporates many of the characteristics of the spiral model.

The *component-based development* model incorporates the following steps:

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

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 specifications of Software. Formal methods enable a Software engineer to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation. A variation of this approach, called *clean-room* Software *engineering* is currently applied by some software development organizations.

Although not a mainstream approach, the formal methods model offers the promise of defect-free Software. Yet, concern about its applicability in a business environment has been voiced:

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

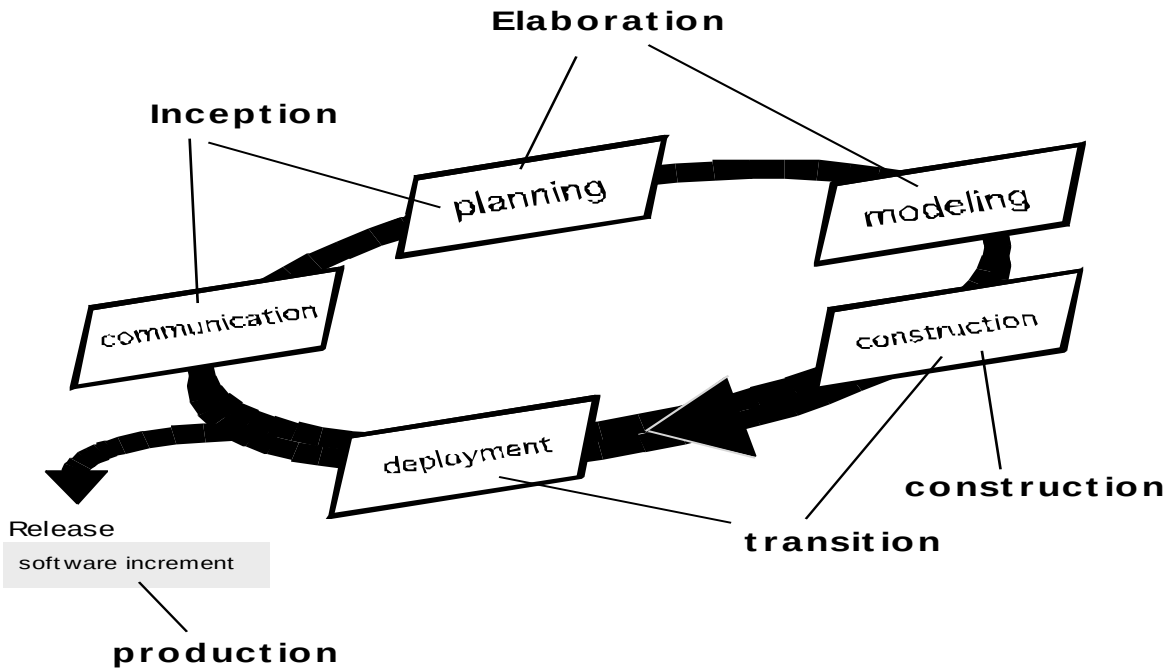
THE UNIFIED PROCESS

A "use-case driven, architecture-centric, iterative and incremental" software process closely aligned with the Unified Modeling Language (UML). The UP is an attempt to draw on the best features and characteristics of conventional software process models, but characterize them in a way that implements many of the best principles of agile software development. The UP recognizes the importance of customer communication and streamlined methods for describing the customer's view of a system. It emphasizes the important role of software architecture and "helps the architect focus on the right goals, such as understandability, reliance to future changes, and reuse." UML provides the necessary technology to support Object Oriented Software Engineering practice, but it doesn't provide the process framework to guide project teams in their application of the technology. The UML developers developed the *Unified Process*, a framework Object Oriented Software Engineering using UML.

Phases of the Unified Process

The figure below depicts the phases of the UP and relates them to the generic activities.

The *Inception* phase of the UP encompasses both customer communication and planning activities. By collaborating with the customer and end-users, business requirements for the software are identified, a rough architecture for the system is proposed, and a plan for the iterative, incremental nature of the ensuing project is developed.



A use-case describes a sequence of actions that are performed by an *actor* (person, machine, another system) as the actor interacts with the Software. The *elaboration* phase encompasses the customer communication and modeling activities of the generic process model. Elaboration refines and expands the preliminary use-cases that were developed as part of the inception phase and expands the architectural representation to include five different views of the software - the use-case model, the analysis model, the design model, the implementation model, and the deployment model.

The *construction* phase of the UP is identical to the construction activity defined for the generic software process. Using the architectural model as input, the construction phase develops or acquires the software components that will make each use-case operational for end-users.

The *transition* phase of the UP encompasses the latter stages of the generic construction activity and the first part of the generic deployment activity. Software is given to end-users for beta testing, and user feedback reports both defects and necessary changes. At the conclusion of the transition phase, the software increment becomes a usable software release "user manuals, trouble-shooting guides, and installation procedures.) The *production* phase of the UP coincides with the development activity of the generic process. The on-going use of the software is monitored, support for the operating environment is provided and defect reports and requests for changes are submitted and evaluated.

A Software Engineering workflow is distributed across all UP phases.