MODULE 1

INTRODUCTION TO SOFTWARE ENGINEERING

Software Engineering is a new technological discipline, but based on the foundations of computer science, management science, economics, communication skills, and the engineering approach to problem solving.

Software Engineering activities occur within an organizational context, and a high degree of communication is required among customers, managers, software engineers, hardware engineers, and other technologists. Good oral, written, and interpersonal communication skills are crucial for software engineer.

A fundamental principle of software engineering is to design software products that minimize the intellectual distance between problem and solution.

Software is intangible; it has no mass, no volume, no color, no odor – no physical properties.

In order to develop a software product, user needs and constraints must be determined and explicitly stated; the product must be designed to accommodate implementers, users, and maintainers; the source code must be carefully implemented and thoroughly tested; and supporting documents such as principles of operation, the user's manual, installation instructions, training aids, and maintenance documents must be prepared. Software maintenance tasks include analysis of change requests, redesign and modification of the source code, thorough testing of the modified code, updating of documents and documentation to reflect the changes, and distribution of the modified work products to appropriate user sites.

The primary goals of software engineering are to improve the quality of software products and to increase the productivity and job satisfaction of software engineers.

THE EVOLVING ROLE OF SOFTWARE

Software takes on a dual role. It is a **product** and, the **vehicle** for delivering a product. As a product, it delivers the computing potential embodied by computer hardware or, a network of computers that are accessible by local hardware. Whether it resides within a cellular phone or operates inside a mainframe computer, software is information transformer— producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as a single bit or as complex as a multimedia presentation.

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 delivers the most important product of our time—information. Software transforms personal data; it manages business information to enhance competitiveness; it provides a gateway to worldwide information networks (e.g., Internet) and provides the means for acquiring information in all of its forms.

The role of computer software has undergone significant change over more than 50 years. Dramatic improvements in hardware performance, profound changes in computing architectures, vast increases in memory and storage capacity, and a wide variety of input and output options have all precipitated more sophisticated and complex computer-based systems. Today a huge software industry has become a dominant factor in the economies of the industrialized world.

DEFINING SOFTWARE ENGINEERING

Software Engineering is defined as systematic, disciplined and quantifiable approach for the development, operation and maintenance of software.

Software Engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

According to Boehm, Software Engineering involves “the practical application of scientific knowledge to the design and construction of computer programs and the associated documentation required to develop, operate, and maintain them”.

The IEEE Standard Glossary of Software Engineering terminology defines software engineering as: “The systematic approach to the development, operation, maintenance, and retirement of software”.

Software Engineering is the technological and managerial discipline concerned with systematic production and maintenance of software products that are developed and modified on time and with cost estimates.

CHANGING NATURE OF SOFTWARE

System Software

System software is a collection of programs which are written to service other programs. Some system software processes complex but determinate, information structures. Other system application process largely indeterminate data. Sometimes when, the system software area is characterized by the heavy interaction with computer hardware that requires scheduling, resource sharing, and sophisticated process management.

Application Software

Application software is defined as programs that solve a specific business need. Application in this area process business or technical data in a way that facilitates business operation or management technical decision making. In addition to convention data processing application, application software is used to control business function in real time.

Engineering and Scientific Software

This software is used to facilitate the engineering function and task. However modern application within the engineering and scientific area are moving away from the conventional numerical algorithms. Computer-aided design, system simulation, and other interactive applications have begun to take real-time and even system software characteristic.

Embedded Software

Embedded software resides within the system or product and is used to implement and control feature and function for the end-user and for the system itself. Embedded software can perform the limited and esoteric function or provided significant function and control capability.

Product-line Software

Designed to provide a specific capability for use by many different customers, product line software can focus on the limited and esoteric marketplace or address the mass consumer market.

Web Application

It is a client-server computer program which the client runs on the web browser. In their simplest form, Web apps can be little more than a set of linked hypertext files that present information using text and limited graphics. However, as e-commerce and B2B application grow in importance. Web apps are evolving into a sophisticate computing environment that not only provides a standalone feature, computing function, and content to the end user.

Artificial Intelligence Software

Artificial intelligence software makes use of a non-numerical algorithm to solve a complex problem that is not amenable to computation or straightforward analysis. Application within this area includes robotics, expert system, pattern recognition, artificial neural network, theorem proving and game playing.

SOFTWARE MYTHS

Many causes of a software affliction can be traced to a mythology that arose during the early history of software development.

Management myths

Managers with software responsibility, are often under pressure to maintain budgets, keep schedules from slipping, and improve quality. Like a drowning person who grasps at a straw, a software manager often grasps at belief in a software myth, if that belief will lessen the pressure.

Myth*:* We already have a book that's full of standards and procedures for building software; won't 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? Does it reflect modern software engineering practice? Is it complete? Is it streamlined to improve time to delivery while still maintaining a focus on quality? In many cases, the answer to all of these questions is "no."

Myth*:* My people have state-of-the-art software development tools, after all, we buy them the newest computers.

Reality*:* It takes much more than the latest model mainframe, workstation, or PC to do high-quality software development. Computer-aided software engineering (CASE) tools are more important than hardware for achieving good quality and productivity, yet the majority of software developers still do not use them effectively.

Myth*:* If we get behind schedule, we can add more programmers and catch up.

Reality*:* Software development is not a mechanistic process like manufacturing. In the words of Brooks: "adding people to a late software project makes it later." At first, this statement may seem counterintuitive. However, as new people are added, people who were working must spend time educating the newcomers, thereby reducing the amount of time spent on productive development effort. People can be added but only in a planned and well-coordinated manner.

Myth: If I decide to outsource the software project to a third party, I can just relax and let that firm build it.
Reality: If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsources software projects.

Customer myths

A customer who requests computer software may be a person at the next desk, a technical group down the hall, the marketing/sales department, or an outside company that has requested software under contract.
In many cases, the customer believes myths about software because software managers and practitioners do little to correct misinformation. Myths lead to false expectations (by the customer) and ultimately, dissatisfaction with the developer.

Myth: A general statement of objectives is sufficient to begin writing programs—we can fill in the details later.

Reality: A poor up-front definition is the major cause of failed software efforts. A formal and detailed description of the information domain, function, behaviour, performance, interfaces, design constraints, and validation criteria is essential. These characteristics can be determined only after thorough communication between customer and developer.

Myth: Project requirements continually change, but change can be easily accommodated because software is flexible.

Reality: It is true that software requirements change, but the impact of change varies with the time at which it is introduced. If serious attention is given to up-front definition, early requests for change can be accommodated easily. The customer can review requirements and recommend modifications with relatively little impact on cost. When changes are requested during software design, the cost impact grows rapidly. Resources have been committed and a design framework has been established. Change can cause upheaval that requires additional resources and major design modification, that is, additional cost. Changes in function, performance, interface, or other characteristics during implementation (code and test) have a severe impact on cost. Change, when requested after software is in production, can be over an order of magnitude more expensive than the same change requested earlier.

Practitioner's myths

Myths that are still believed by software practitioners have been fostered by 50 years of programming culture. During the early days of software, programming was viewed as an art form. Old ways and attitudes die hard.

Myth: Once we write the program and get it to work, our job is done.

Reality: Someone once said that "the sooner you begin 'writing code', the longer it'll take you to get done." Industry data indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.

Myth: Until I get the program "running" I have no way of assessing its quality.

Reality: One of the most effective software quality assurance mechanisms can be applied from the inception of a project—the formal technical review. Software reviews are a "quality filter" that have been found to be more effective than testing for finding certain classes of software defects.

Myth: The only deliverable work product for a successful project is the working program.

Reality: A working program is only one part of a software configuration that includes many elements. Documentation provides a foundation for successful engineering and, more important, guidance for software support.

Myth: Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.

Reality: Software engineering is not about creating documents. It is about creating quality. Better quality leads to reduced rework. And reduced rework results in faster delivery times.

Many software professionals recognize the fallacy of the myths just described. Regrettably, habitual attitudes and methods foster poor management and technical practices, even when reality dictates a better approach. Recognition of software realities is the first step toward formulation of practical solutions for software engineering.

TERMINOLOGIES

Software engineering differs from traditional computer programming in that engineering-like techniques are used to specify, design, implement, validate, and maintain software products within the time and budget constraints established for the project. In addition, software engineering is concerned with managerial issues.

Programmer is used to denote an individual who is concerned with the details of implementing, packaging, and modifying algorithms and data structures written in particular programming languages. **Software engineers** are additionally concerned with issues of analysis, design, verification and testing, documentation, software maintenance, and project management. The terms “developer” and “software engineer” are used interchangeably.

Computer software is synonymous with “computer program” or “source code” or “software product”. Software is not just the programs but also all associated documentation and configuration data which is needed to make these programs operate correctly. A software system usually consists of a number of separate programs, configuration files which are used to set up these programs, system documentation which describes the structure of the system and user documentation which explains how to use the system and, for software products, web sites for users to download recent product information.

Documentation explains the characteristics of a document. Internal documentation of source code describes the characteristics of the code, and external documentation explains the characteristics of the documents associated with the code.

The term “**Customer**” is used to denote an individual or organization that initiates procurement or modification of a software product.

Fundamental **quality attributes**:

1. Usefulness: The most important quality attribute ie., the software product must satisfy user needs. Careful planning, analysis, and customer involvement are required to develop useful software products.
2. Reliability: The ability of a program to perform a required function under stated conditions for a stated period of time.
3. Clarity: Software products must be clearly written and easy to understand.
4. Efficiency: Software product should be efficient. It is a fundamental quality attribute.

ROLE OF SOFTWARE DEVELOPMENT

The Software Development Life Cycle (SDLC) consists of the following phases: Analysis, Design, Implementation, System Testing, and Maintenance.

Analysis

Analysis consists of two sub phases: *Planning* and *Requirements Definition*.

Planning include understanding the customer’s problem, performing a feasibility study, developing a recommended solution strategy, determining the acceptance criteria, and planning the development process. The products of planning are a *System Definition* and a *Project Plan*.

The System Definition is typically expressed in English or natural language, and may incorporate charts, figures, graphs, tables, and equations.

The Project Plan contains the life-cycle model to be used, the organizational structure for the project, the preliminary development schedule, preliminary cost and resource esti8mates, preliminary staffing requirements, tools and techniques to be used, and standard practices to be followed.

Requirements Definition is concerned with identifying the basic functions of software component in a hardware / software / people system. The product of requirements definition is a specification that describes the processing environment, the required software functions, performance constraints on the software (size, speed, machine configuration), exception handling, subsets and implementation priorities, probable changes and likely modifications, and the acceptance criteria for the software.

Design

Design is concerned with identifying software components (functions, data streams, and data stores), specifying relationships among components, specifying software structure, maintaining a record of design decisions, and providing a blueprint for the implementation phase. Detailed design consists of *Architectural Design* and *Detailed Design.*

Architectural design involves identifying the software components, decoupling and decomposing them into processing modules and conceptual data structures, and specifying the interconnections among components.

Detailed design is concerned with the details of "how to". It involves adaptation of existing code, modification of standard algorithms, invention of new algorithms, design of data representations, and packaging of the software product.

Implementation

It involves translation of design specifications into source code, and debugging, documentation, and unit testing of the source code. Modern programming languages provide many features to enhance the quality of source code. These include structured control constructs, built-in and user-defined data types, secure type checking, flexible scope rules, exception handling mechanisms, concurrency constructs, and separate compilation of modules.

System Testing

It involves two kinds of activities: *Integration Testing and Acceptance Testing.* Developing a strategy for **integrating** the components of a software system into a functioning whole requires careful planning so that modules are available for integration when needed. **Acceptance testing** involves planning and execution of various types of tests in order to demonstrate that the implemented software system satisfies the requirements stated in the requirements document.

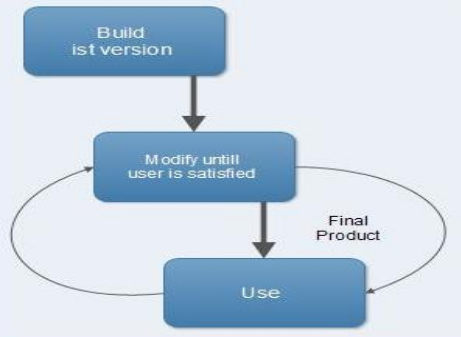
Maintenance

Following acceptance by the customer, the software system is released for production work and enters the maintenance phase of the phased life-cycle model. Maintenance activities include enhancement of capabilities, adaptation of the software to new processing environments, and correction of software bugs.

SOFTWARE LIFE CYCLE MODELS

Planning the software development process is important. The software life cycle encompasses all activities required to define, develop, test, deliver, operate, and maintain a software product.

BUILD & FIX MODEL



In the build and fix model (also referred to as an **ad hoc model**), the software is developed without any specification or design. An initial product is built, which is then repeatedly modified until it satisfies the user. That is, the software is developed and delivered to the user. The user checks whether the desired functions 'are present. If not, then the software is changed according to the needs by adding, modifying or deleting functions. This process goes on until the user feels that the software can be used productively. However, the lack of design requirements and repeated modifications result in loss of acceptability of software. Thus, software engineers are strongly discouraged from using this development approach.

This model includes the following two phases:

1. **Build**: In this phase, the software code is developed and passed on to the next phase.
2. **Fix**: In this phase, the code developed in the build phase is made error free. Also, in addition to the corrections to the code, the code is modified according to the user's requirements.

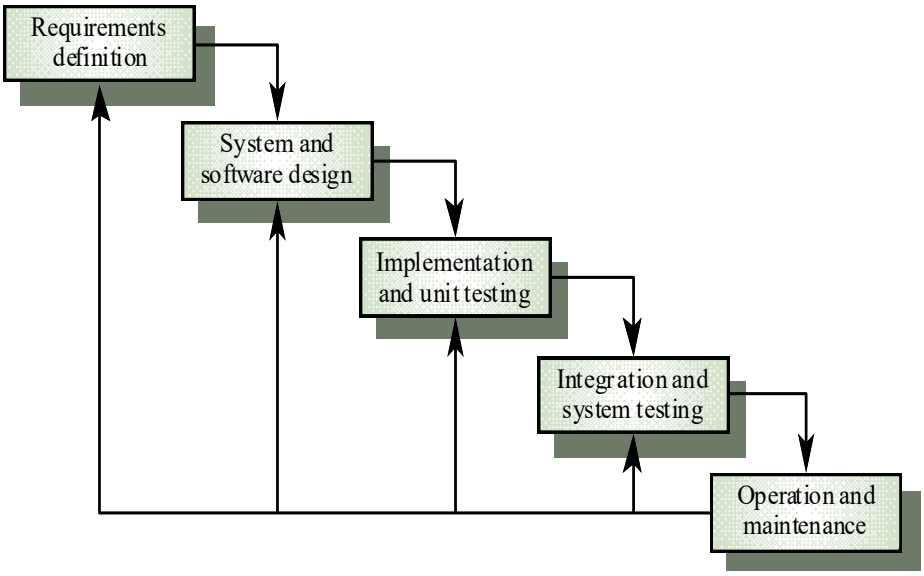
Advantages

1. Requires less experience to execute or manage other than the ability to program.
2. Suitable for smaller software.
3. Requires less project planning.

Disadvantages

1. No real means is available of assessing the progress, quality, and risks.
2. Cost of using this process model is high as it requires rework until user's requirements are accomplished.
3. Informal design of the software as it involves unplanned procedure.
4. Maintenance of these models is problematic.

WATERFALL MODEL



Waterfall model is the simplest model of software development paradigm. It says the all the phases of Software Development Life Cycle (SDLC) will function one after another in linear manner. That is, when the first phase is finished then only the second phase will start and so on. This model is also called Phased Life Cycle Model or Waterfall Chart or Linear Sequential Model or Classical Life Cycle Model.

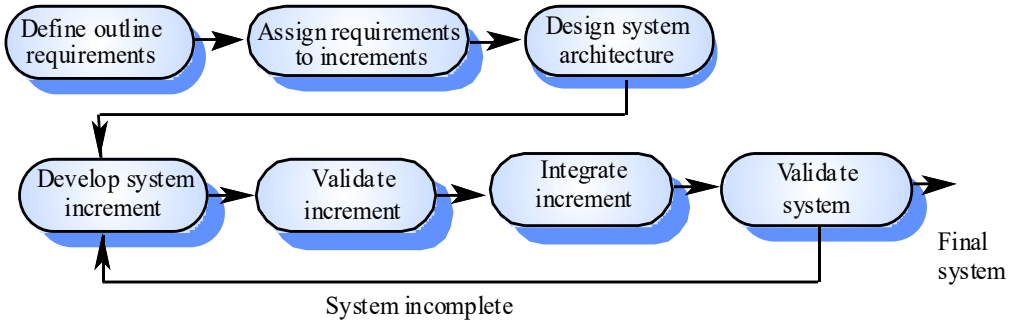
The waterfall model suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through design, implementation, testing and maintenance.

Drawbacks

- Real projects rarely follow the sequential flow
- Accommodates iteration indirectly
- Changes can cause confusion
- It is often difficult for the customer to state all requirements explicitly
- Has difficulty accommodating the natural uncertainty that exists at the beginning of many projects
- The customer must have patience
- A working version of the program(s) will not be available until late in the project time-span
- A major blunder, if undetected until the working program is reviewed, can be disastrous
- Leads to “blocking states

INCREMENTAL PROCESS MODELS

- 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.
- 1st increment is often a core product
- 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.
- For example, word-processing software developed using the incremental paradigm might deliver basic file management, editing, and document production functions in the first increment; more sophisticated editing and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advanced page layout capability in the fourth increment.



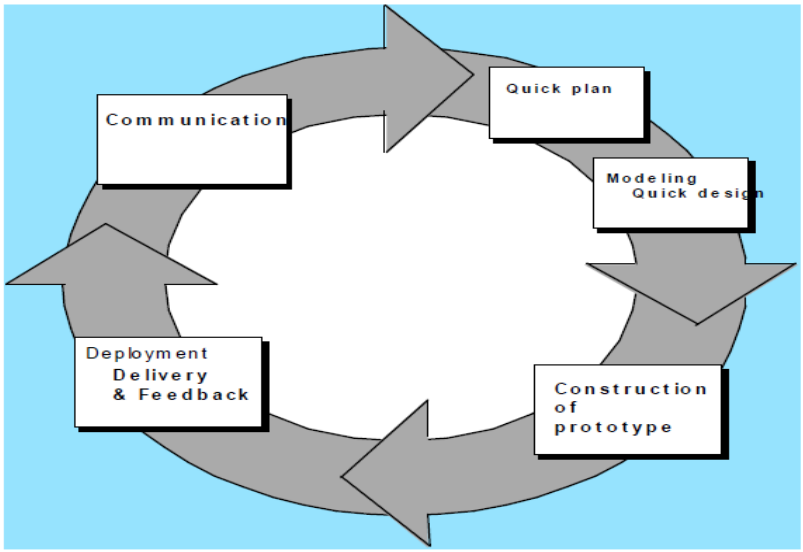
Advantages

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

EVOLUTIONARY PROCESS MODELS

Complex software system 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; tight market deadlines make completion of a comprehensive software product impossible, but a limited version must be introduced to meet competitive or business pressure. Evolutionary models are iterative.

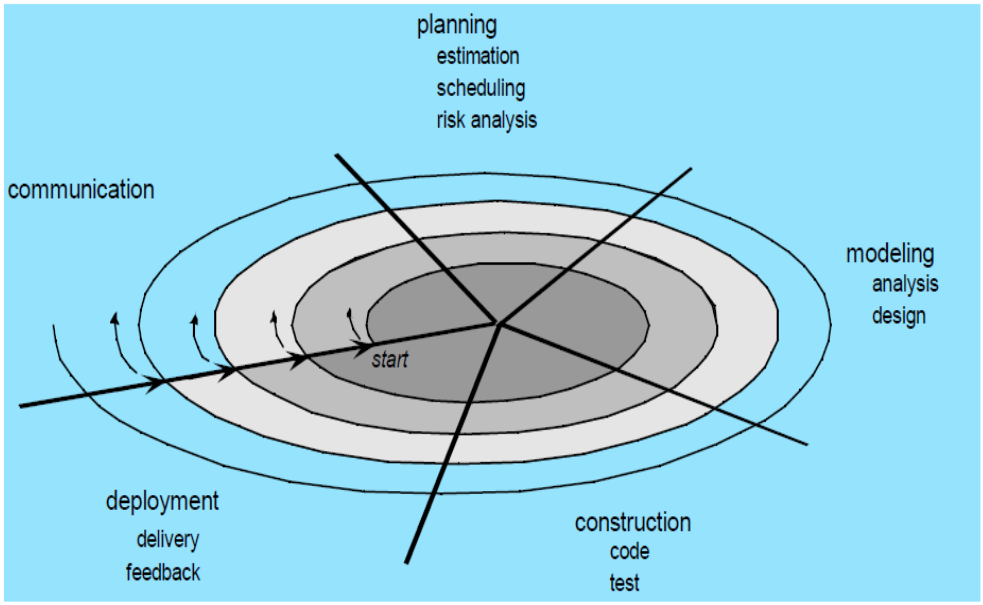
PROTOTYPING



- The prototyping paradigm begins with communication where requirements and goals of Software are defined.
- Prototyping iteration is planned quickly and modelling in the form of quick design occurs.
- The quick design leads to the Construction of the Prototype.
- Communication or feedback is used to refine requirements for the software.

THE SPIRAL MODEL

- A cyclic approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk.
- Set of anchor point milestones for ensuring Stake holder commitment of easy and mutually satisfactory solutions



Spiral model sectors

- Objective setting
  - Specific objectives for the phase are identified.
- Risk assessment and reduction
  - Risks are assessed and activities put in place to reduce the key risks.
- Development and validation
  - A development model for the system is chosen which can be any of the generic models.
- Planning
  - The project is reviewed and the next phase of the spiral is planned**.**

Advantages

- Focuses attention on reuse options.
- Focuses attention on early error elimination.
- Puts quality objectives up front.
- Integrates development and maintenance.
- Provides a framework for hardware/software development.

Disadvantages

- Contractual development often specifies process model and deliverables in advance.
- Requires risk assessment expertise**.**

THE UNIFIED PROCESS (UP)

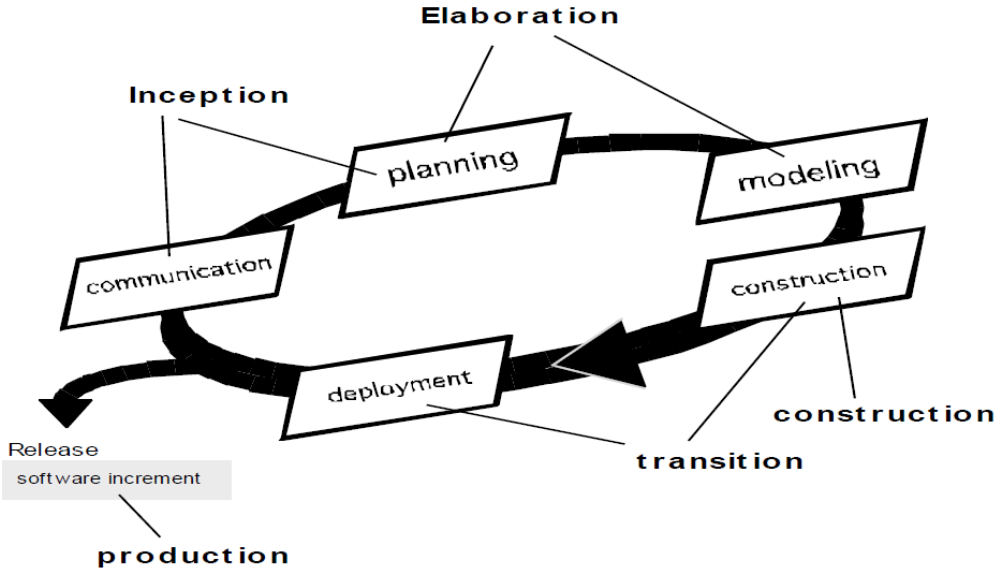
- It is a use-case driven, architecture-centric, iterative and incremental software process
- UP is an attempt to draw on the best features and characteristics of conventional software process models
- Also implements many of the best principles of agile software development

- UP is a framework for object-oriented software engineering using UML (Unified Modeling Language)

Recognize

- Importance of customer communication
- Methods for describing the customer's view of a system
- Easy to identify goals, such as understandability, add future changes, and reuse

The UML developers developed the Unified Process, a framework Object Oriented Software Engineering using UML.



Five different views of the software – the use-case model, the analysis model, the design model, the implementation model and the deployment model.

SELECTION OF A LIFE CYCLE MODEL

Selection of proper lifecycle model is the most important task. It can be selected by keeping the advantages and disadvantages of various models in mind. The different issues that are analyzed before selecting a suitable life cycle model are:

- Characteristics of the software to be developed
- Characteristics of the development team
- Risk associated with the project
- Characteristics of the customer