UNIT-I
INTRODUCTION TO SOFTWARE ENGINEERING

Software: Software is

(1) Instructions (computer programs) that provide desired features, function, and performance, whenexecuted.
(2) Data structures that enable the programs to adequately manipulate information,
(3) Documents that describe the operation and use of the programs.

Characteristics of Software:

(1) Software is developed or engineered; it is not manufactured in the classical sense.
(2) Software does not "wear out"
(3) Although the industry is moving toward component-based construction, most software continues tobe custom built.

Software Engineering:

(1) The systematic, disciplined quantifiable approach to the development, operation and maintenance ofsoftware; that is, the application of engineering to software.
(2) The study of approaches as in (1)

EVOLVING ROLE OF SOFTWARE:

Software takes dual role. It is both a **product** and a **vehicle** for delivering a product.
As a **product**: It delivers the computing potential embodied by computer Hardware or by a networkof computers.
As a **vehicle**: It is information transformer-producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as single bit or as complex as a multimedia presentation. Software delivers the most important product of our time- information.

- It transforms personal data
- It manages business information to enhance competitiveness
- It provides a gateway to worldwide information networks
- It provides the means for acquiring information

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

- Dramatic Improvements in hardware performance
- Vast increases in memory and storage capacity
- A wide variety of exotic input and output options

THE CHANGING NATURE OF SOFTWARE:

The 7 broad categories of computer software present continuing challenges for software engineers:

1) System software
2) Application software
3) Engineering/scientific software
4) Embedded software
5) Product-line software
6) Web-applications
7) Artificial intelligence software.

System software:

1. It is the computer software that is designed to operate the computer hardware manage the functioning of the application software running on it.
2. Examples of system software are device drivers, boot program, operating systems, servers, utilities, and so on. Such software reduces the burden of application programmers.

Application software:

1. Application Software is designed to accomplish certain specific needs of the end user.
2. Sale transaction software, educational software, video editing software, word processing software, database software are some examples of application software

Engineering/Scientific software:

3. Engineering problems and quantitative analysis are carried out using automated tools.
4. Scientific software is typically used to solve mathematical functions and calculations.
5. Computer-aided design and computer-aided manufacturing software (CAD/CAM) , electronic design automation (EDA), embedded system software (ESS), statistical process control software (SPCS), civil engineering and architectural software, math calculation software, modeling and simulation software, etc.., are the examples of engineering scientific software.

Embedded software:

1. Embedded software is a type of software that is built into hardware systems.
2. Controllers, real-time operating systems, communication protocols are some examples of embedded software.

Product-line software:

1. Product-line software is a set of software intensive systems that share a common, managed set of features to satisfy the specific needs of a particular market segment or mission.
2. Some common applications are multimedia, database software, word processing software, etc,.

Web-applications:

1. Web applications are base on client server architecture, where the client request information and the server stores and retrieves information from the web software.
2. Examples include HTML 5.0,JSp,ASP,PHP etc

Artificial intelligence software:

1. AI software is made to think like human beings and therefore it is useful in solving complex problems automatically.
2. Game playing, speech recognition, understanding natural language. Computer vision, expert systems, robotics are some applications of AI software

The following are the **new challenges** on the horizon:

1. **Ubiquitous computing.Net sourcing**
2. **Open source**
3. **The "new economy"**

Ubiquitous computing: The **challenge** for software engineers will be to develop systems and application software that will allow small devices, personal computers and enterprise system to communicate across vast networks.

Net sourcing: The **challenge** for software engineers is to architect simple and sophisticated applications that provide benefit to **targeted end-user market** worldwide.

Open Source: The **challenge** for software engineers is to build source that is self descriptive but more importantly to develop techniques that will enable both **customers and developers to know what changes have been made and how those changes manifest themselves** within the software.

The "new economy": The **challenge** for software engineers is to build applications that will facilitate masscommunication and **mass product distribution**.

SOFTWARE MYTHS

Beliefs about software and the process used to build it- can be traced to the earliest days of computing myths have a number of attributes that have made them insidious.

Management myths: Manages with software responsibility, like managers in most disciplines, are often under pressure to maintain budgets, keep schedules from slipping, and improve quality.

Myth: We already have a book that's full of standards and procedures for building software - Wont that provide my people with everything they need to know?

Reality: The book of standards may very well exist but, is it used? Are software practitioners aware of its existence? Does it reflect modern software engineering practice?

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

Reality: Software development is not a mechanistic process like manufacturing. As new people are added, people who were working must spend time educating the new comers, thereby reducing the amount of timespend 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 built 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: The customer believes myths about software because software managers and practitioners do little to correct misinformation. Myths lead to false expectations and ultimately, dissatisfaction with the developer.

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

Reality: Although a comprehensive and stable statement of requirements is not always possible, anambiguous statement of objectives is recipe for disaster.

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 and change can cause upheaval that requires additional resources and major design modification.

Practitioner's myths: Myths that are still believed by software practitioners: during the early days of software, programming was viewed as an art from old ways and attitudes die hard.

Myth: Once we write the program and get it to work, our jobs are 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: The only deliverable work product for a successful project is the working program.

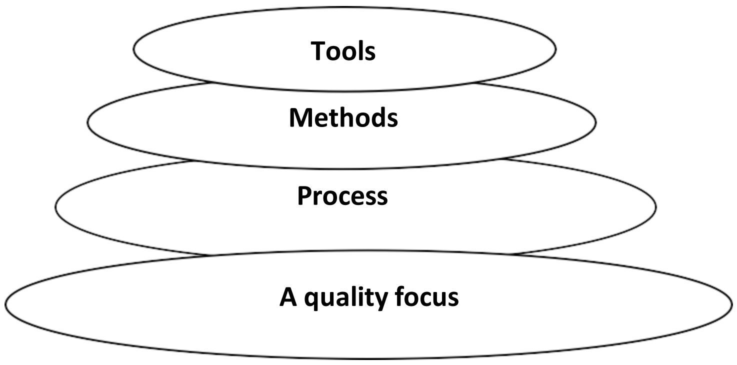
Reality: A working program is only one part of a software configuration that includes many elements. Documentation provides guidance for software support.

Myth: software engineering will make us create voluminous and unnecessary documentation and will invariably slows 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.

A GENERIC VIEW OF PROCESS:

SOFTWARE ENGINEERING - A LAYERED TECHNOLOGY:



Software Engineering Layers

Software engineering is a layered technology. Any engineering approach must rest on an organizationalcommitment to quality. **The bedrock that supports software engineering is a quality focus.**

The foundation for software engineering is the process layer. Software engineering process is the glue that holds the technology layers. **Process defines a framework that must be established for effective delivery of software engineering technology**.

The software forms the basis for management control of software projects and establishes the contextin which

- technical methods are applied,
- work products are produced,
- milestones are established,
- quality is ensured,
- And change is properly managed.

Software engineering methods rely on a set of basic principles that govern area of the technology andinclude modeling activities.

Methods encompass a broad array of tasks that include

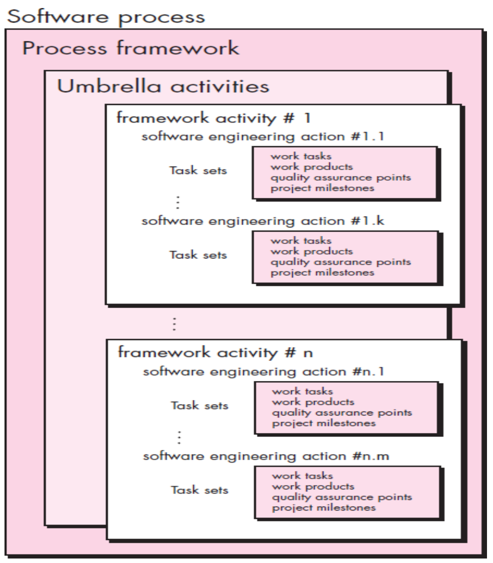
- ✓ communication,
- ✓ requirements analysis,
- ✓ design modeling,
- ✓ program construction,
- ✓ Testing and support.

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

1) **Communication**: This framework activity involves heavy communication and collaboration with the customer and encompasses requirements gathering and other related activities.
2) **Planning**: This activity establishes a plan for the software engineering work that follows. It describes the technical tasks to be conducted, the risks that are likely, 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 that allow the developer and customer to better understand software requirements and the design that will achieve those requirements. The modeling activity is composed of 2 software engineering actions- analysis and design.
    - ✓ Analysis encompasses a set of work tasks.
    - ✓ Design encompasses work tasks that create a design model.
4) **Construction:** This activity combines core generation and the testing that is required to uncoverthe errors in the code.
5) **Deployment:** The software is delivered to the customer who evaluates the delivered product andprovides feedback based on the evolution.

These 5 generic framework activities can be used during the development of small programs, the creationof large web applications, and for the engineering of large, complex computer-based systems.

A PROCESS FRAMEWORK:



- **Software process** must be established for effective delivery of software engineering technology.
- A **process framework** establishes the foundation for a complete software process by identifying a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.
- The process framework encompasses a **set of umbrella activities** that are applicable across theentire software process.
- Each **framework activity** is populated by a set of software engineering actions
- Each **software engineering action** is represented by a number of different task sets- each a collection of software engineering work tasks, related work products, quality assurance points, andproject milestones.

In brief

"A **process** defines who is doing what, when, and how to reach a certain goal."

A **Process Framework**

- establishes the foundation for a complete software process
- identifies a small number of **framework activities**
    - applies to all s/w projects, regardless of size/complexity.
- also, set of **umbrella activities**
    - applicable across entire s/w process.

- Each **framework activity** has
    - set of **s/w engineering actions**.
- Each **s/w engineering action** (e.g., design) has
    - collection of related **tasks (**called

        task sets): work tasks
        work products
        (deliverables) quality
        assurance points
        project milestones.

The following are the set of **Umbrella Activities.**

1) **Software project tracking and control –** allows the software team to assess 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 -** assesses software engineering work products in an effort to uncover and remove errors before they are propagated to the next action or activity.
5) **Measurement -** define and collects process, project and product measures that assist the team in delivering software that needs customer’s needs, can be used in conjunction with all other framework and umbrella activities.
6) **Software configuration management -** manages the effects of change throughout the software process.
7) **Reusability management** - defines criteria for work product reuse and establishes mechanisms to achieve reusable components.
8) **Work Product preparation and production -** encompasses the activities required to create work products such as models, document, logs, forms and lists.

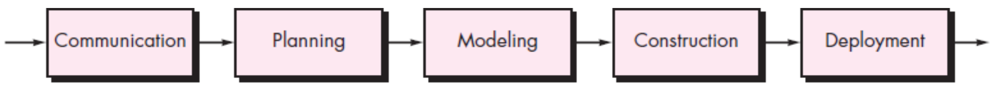
SOFTWARE PROCESS:

A **linear process flow** execute search of the five frame work activities in sequence, beginning with communication and culminating with deployment.

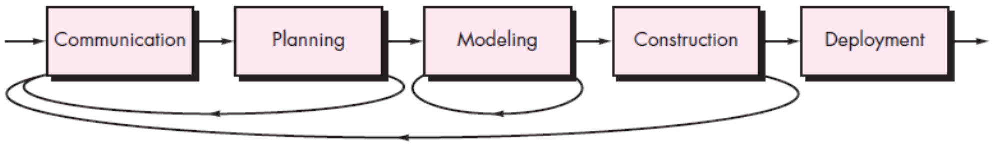
An **Iterative process flow** repeats one or more of the activities before proceeding to the next.

An **Evolutionary process** flow executes the activities in a“ circular” manner. Each circuit through the five activities leads
to a more complete version of the software.

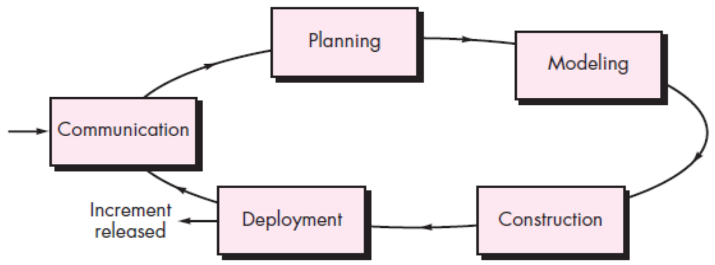
A parallel process flow executes one or more activities in parallel with other activities (e.g, modeling for one
Aspect of the software might be executed in parallel with construction of another aspect of the software).



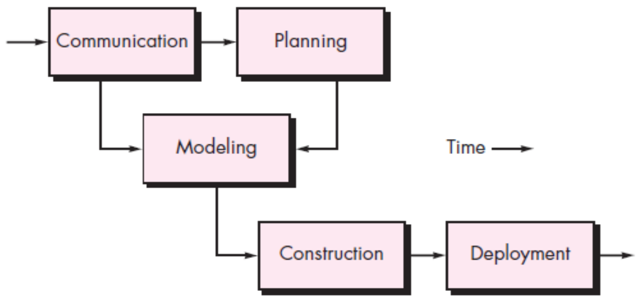
(a) Linear process flow



(b) Iterative process flow



(c) Evolutionary process flow



(d) Parallel process flow

THE CAPABILITY MATURITY MODEL INTEGRATION (CMMI):

The CMMI represents a process meta-model in two different ways:

- As a continuous model
- As a staged model.

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

Level 0: Incomplete. The process area is either not performed or does not achieve all goals and objectives defined by CMMI for level 1 capability.

Level 1: Performed. All of the specific goals of the process area have been satisfied. Work tasks required to produce defined work products are being conducted.

Level 2: Managed. All level 1 criteria have been satisfied. In addition, all work associated with the processarea conforms to an organizationally defined policy; all people doing the work have access to adequate resources to get the job done; stakeholders are actively involved in the process area as required; all work tasks and work products are "monitored, controlled, and reviewed;

Level 3: Defined. All level 2 criteria have been achieved. In addition, the process is "tailored from the organizations set of standard processes according to the organizations tailoring guidelines, and contributes and work products, measures and other process-improvement information to the organizational process assets".

Level 4: Quantitatively managed. All level 3 criteria have been achieved. In addition, the process area is controlled and improved using measurement and quantitative assessment."Quantitative objectives for quality and process performance are established and used as criteria in managing the process"

Level 5: Optimized. All level 4 criteria have been achieved. In addition, the process area is adapted and optimized using quantitative means to meet changing customer needs and to continually improve the efficacy of the process area under consideration"

SOFTWARE DEVELOPMENT PROCESS MODELS

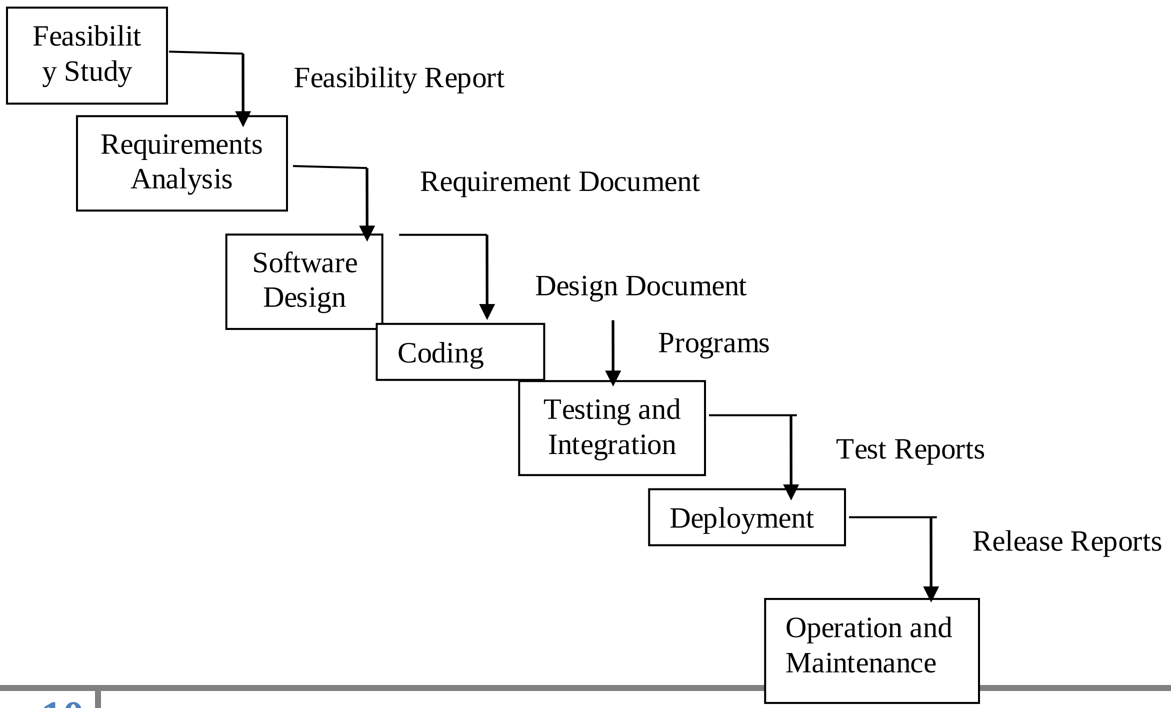
- ✓ Software development organizations follow some development process models when developing a software product.
- ✓ The general activities of the software life cycle models are feasibility study, analysis, design, coding, testing, deployment, and maintenance.

We will discuss the following development process models

1. **Waterfall Model/Classical/Traditional/SDLC**
2. **V-Model**
3. **Incremental Model**
4. **Evolutionary Process Model**
   a. **Prototype model**
   b. **Spiral Model**
   c. **Concurrent Process Model**
5. **Specialized Process Model**
6. **Unified Process**

1. CLASSICAL WATERFALL MODEL

- ✓ The waterfall model is a classical development process model proposed by R.W Royce in 1970.
- ✓ In this model, software development proceeds through an orderly sequence of transitions from one phase to the next in order (like a waterfall).
- ✓ There is no concept **of Backtracking** in this model.



Advantages of Classical waterfall model

1. Simple and easy to understand and use.
2. Easy to manage due to the rigidity of the model.
3. Works well for smaller projects where requirements are very well understood.
4. The amount of resources required to implement this model are minimal.
5. Development processed in sequential manner so very less chance to rework.
6. Due to straightforward organization of phases, it is fit for other engineering process models, such as civil, mechanical etc.
7. It is a document-driven process that can help new people to transfer knowledge.

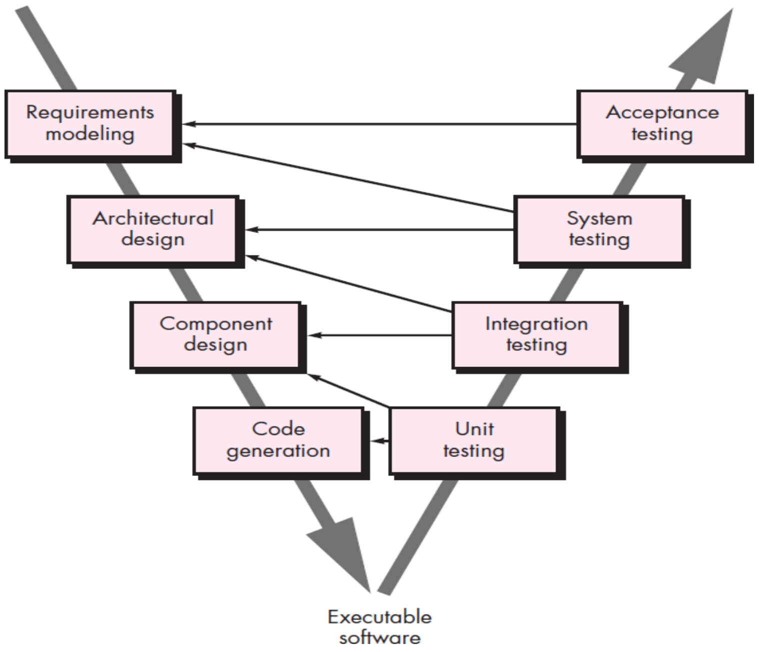
Disadvantages of Classical waterfall model

1. The model assumes that the requirements will not change during the project.
2. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-constructed in the earlier stages.
3. No working software is produced until late during the life cycle.
4. It is very difficult to estimate time and cost in the waterfall model.
5. Not a good model for complex and object-oriented projects.
6. Poor model for long and ongoing projects.
7. Not suitable for the projects where requirements are at a moderate to high risk of changing.
8. Less effective if requirements are not very clear at the beginning.

Some situations where the use of Classical Waterfall model is most appropriate are (*Applicability*)

- Requirements are very well documented, clear and fixed.
- Product definition is stable or when changes in the project are stable.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- Ample resources with required expertise are available to support the product.
- The project is short or small.
- For low budget projects.

2. V – Model :

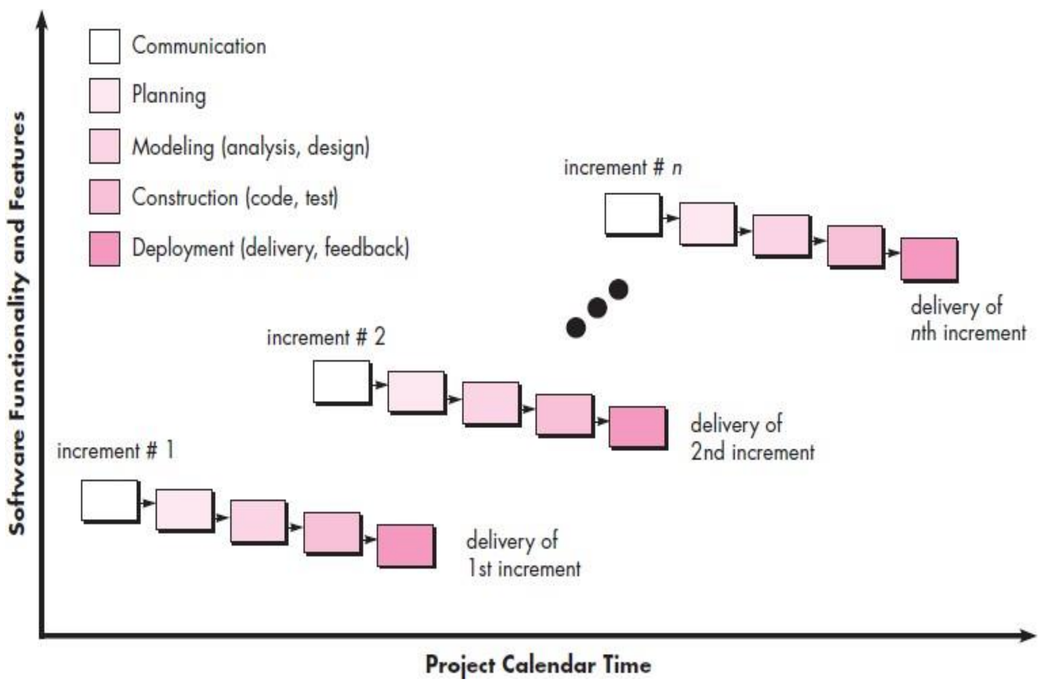


As a software team moves down the left side of the **V**, basic problem requirements are refined into progressively more detailed and technical representations of the problem and its solution. Once code has been generated, the team moves up the right side of the **V**, essentially performing a series of tests that validate each of the models created as the team moved down the left side. **The V-model provides a way of visualizing how verification and validation actions are applied to earlier engineering work**.

3. INCREMENTAL MODEL

- ✓ In incremental model the whole requirement is divided into various builds or Modules.
- ✓ Multiple development cycles take place here, making the life cycle a“multi-waterfall” cycle.
- ✓ Cycles are divided up into smaller, more easily managed modules.
- ✓ Each module passes through the requirements, design, implementation and testing phases.
- ✓ A working version of software is produced during the first module, so you have working software early on during the software life cycle.
- ✓ Each subsequent release of the module adds function to the previous release.
- ✓ The process continues till the complete system is achieved.

The process of Incremental model is shown in the figure given below


Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
increment # n
delivery of nth increment
increment # 2
delivery of 2nd increment
increment # 1
delivery of 1st increment
Software Functionality and Features
Project Calendar Time

Advantages of Incremental model:

1. Generates working software quickly and early during the software life cycle.
2. This model is more flexible – less costly to change scope and requirements.
3. It is easier to test and debug during a smaller iteration.
4. In this model customer can respond to each built.
5. Lowers initial delivery cost.
6. Easier to manage risk because risky pieces are identified and handled during it’d iteration.

Disadvantages of Incremental model:

1. Needs good planning and design.
2. Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
3. Total cost is higher than waterfall.

When to use incremental model:

- This model can be used when the requirements of the complete system are clearly defined and understood.

1. Used when major requirements must be defined; however, some details can evolve with time.
2. Used when there is a need to get a product to the market early.
3. Used when a new technology is being used.
4. Used when resources with needed skill set are not available.
5. Used when there are some high risk features and goals.

4. **EVOLUTIONARY PROCESS MODELS** :

a. PROTOTYPE MODEL

- ✓ A **prototype model** is a toy/demo implementation of the actual product or system.
- ✓ A prototype model usually exhibits limited functional capabilities, low reliability, and inefficient performance as compared to the actual software.

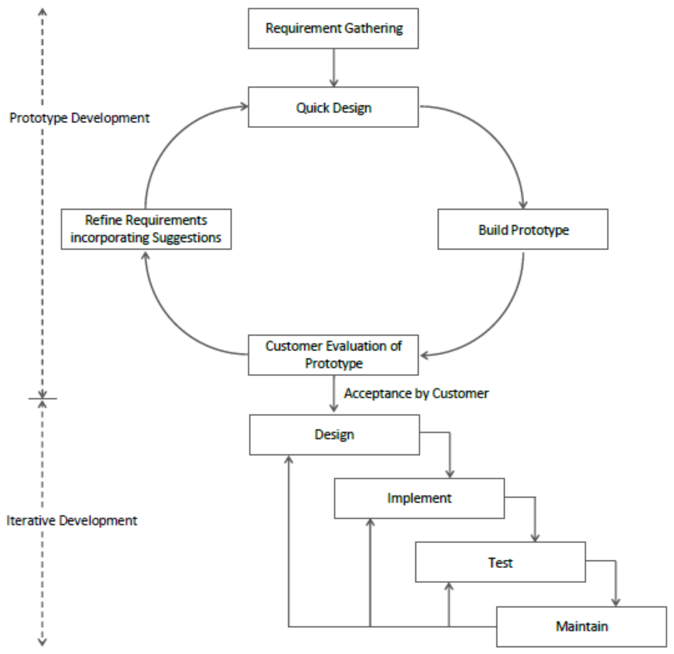


Figure 3: Prototype Model of Software Development

In the **prototype model**, prototyping starts with **initial requirements gathering** phase. Quick design is carried out and a prototype is built. The developed prototype is submitted to the customer for his assessment.

Based on the customer feedback, the requirements are refined and the prototype is suitably modified. This cycle of obtaining customer feedback and modifying the prototype continues until the customer approves the prototype. Once the *customer approves* the prototype, the actual system is developed using the iterative waterfall approach.

Need for a Prototype Model in Software Development:

1. To illustrate the input data formats, messages, reports, and interactive dialogues to the customer.
2. To gain a better understanding of the customer's needs: how the screen might look like, how the user interface would behave, and how the system would produce the output?
3. To examine the technical issues associated with product development.
4. It is not possible to get the perfect product in the first attempt. Many engineers and researchers advocate that if you want to develop a good product you must plan to throw away the first version. The experience acquired in developing the prototype can be used to develop the final product.

Advantages of Prototype Model

1. **Demo working model**: Customer get demo working model of actual product which help them to give a better understanding and attain a high level of satisfaction.
2. **New requirement**: Based on the customer feedback, the requirements are redefined and the prototype is suitably modified till final approval.
3. **Missing functionality**: can be easily established.
4. **Easy error detection**: It saves time and cost in developing the prototype and enhance the quality of the final product.
5. **Flexibility**: in the development phase.

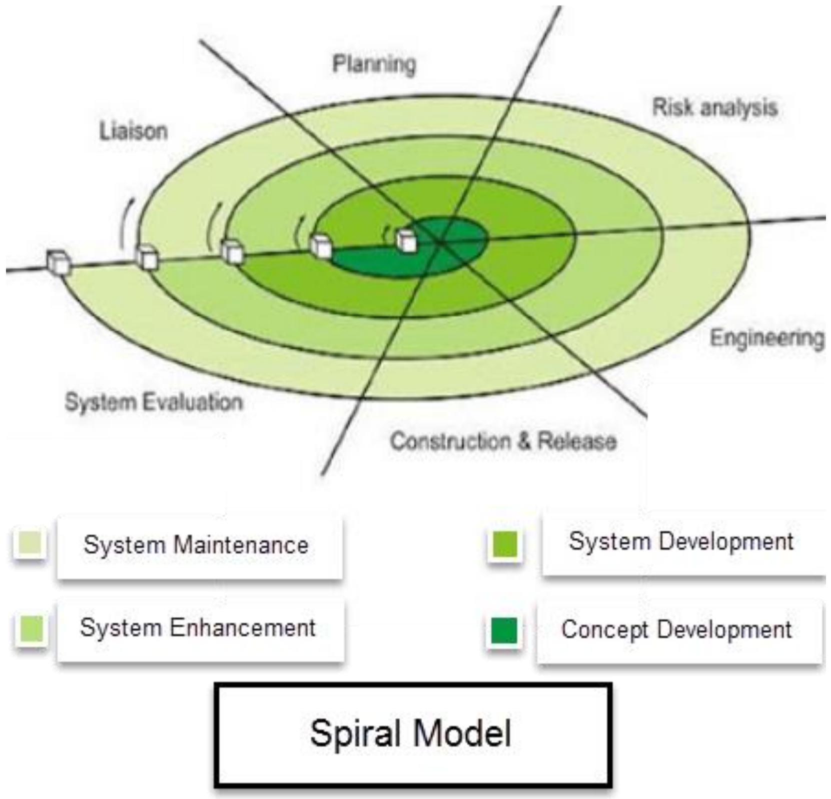
Disadvantages of Prototype Model

1. **Time-consuming:** As the prototype is being modified time to time according to customer requirement which usually increases the time of completion of the product.
2. **Complexity**: Change in the requirement usually expand the scope of the product beyond its original plan and thus increase the complexity.
3. **Poor Documentation**: Continuous changing of requirement can lead to poor documentation.

4. **Unpredictability of no of iteration**: It is difficult to determine the no of iteration required before the prototype is finally accepted by the customer.
5. **Confusion**: Customer can confuse between the actual product and prototype

B. SPIRAL PROCESS MODEL

- ✓ The spiral model is an iterative software development approach which was proposed by Boehm in 1988.
- ✓ The spiral model is shown in the figure given below



- ✓ **Spiral Model** is a risk-driven software development process model.
- ✓ It is a **combination of waterfall model and iterative model**.
- ✓ Each phase of spiral model in software engineering begins with a **design goal** and **ends with the client reviewing** the progress.

ACTIVITIES PERFORMED DURING PHASE:

- **Planning:** It includes estimating the cost, schedule and resources for the iteration. It also involves understanding the system requirements for continuous communication between the system analyst and the customer.
- **Risk Analysis:** Identification of potential risk is done while risk mitigation strategy is planned and finalized.

- **Engineering:** It includes testing, coding and deploying software at the customer site.
- **Evaluation:** Evaluation of software by the customer. Also, includes identifying and monitoring risks such as schedule slippage and cost overrun.

Advantages:

1. Additional functionality or changes can be done at a later stage
2. Cost estimation becomes easy as the prototype building is done in small fragments.
3. Continuous or repeated development helps in risk management.
4. Development is fast and features are added in a systematic way in Spiral development
5. There is always a space for customer feedback.

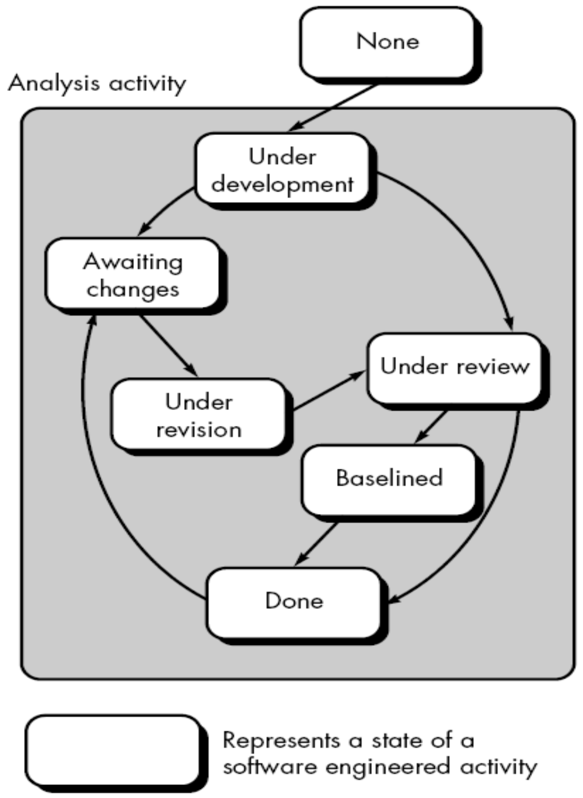
Disadvantages:

1. Spiral software development is not advisable for smaller project, it might cost them a lot.
2. Documentation is more as it has intermediate phases

When to use Spiral Model:

- A Spiral model in software engineering is used when project is large
- When releases are required to be frequent, spiral methodology is used
- When creation of a prototype is applicable
- When risk and costs evaluation is important
- Spiral methodology is useful for medium to high-risk projects
- When requirements are unclear and complex, Spiral model in SDLC is useful
- When changes may require at any time
- When long term project commitment is not feasible due to changes in economic priorities

C. CONCURRENT PROCESS MODEL:



- The concurrent development model is called **Concurrent Engineering**.
- The communication activity has completed in the first iteration and exits in the awaiting changes state.
- The modelling activity completed its initial communication and then go to the under development state.
- if the customer specifies the change in the requirement then the modeling activity moves from the under development state into the awaiting change state.
- The concurrent Process model activities moving from one state to another state.

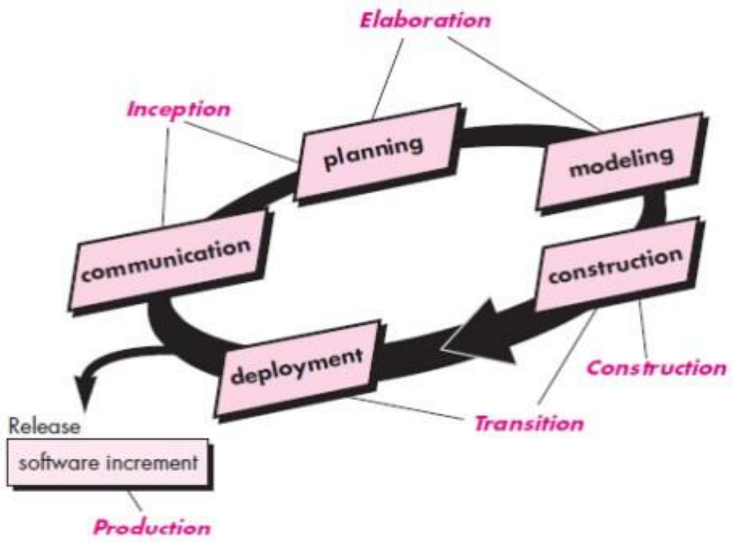
Advantages of Concurrent Model:

- This model applicable to all types of software development process.
- It is easy to understand and use.
- It gives immediate feedback from testing.
- It provides an accurate picture of the current state of the project.

Disadvantages:

- It needs better communication between the team members. This may not be achieved all the time.
- It requires to remember the status of the different activities.

UNIFIED PROCESS MODEL:



Rational Unified Process (RUP) is a software development process for object-oriented models. It is also known as the Unified Process Model. It is created by Rational corporation and is designed and documented using UML (Unified Modeling Language).

RUP is proposed by Ivar Jacobson, Grady Bootch, and James Rambaugh.

Phases of RUP:

- **A. Inception**
- **B. Elaboration**
- **C. Construction**
- **D. Transition**
- **E. Production**

A. Inception

1. Communication and planning are main.
2. Identifies Scope of the project using use-case model allowing managers to estimate costs and time required.
3. Customers' requirements are identified and then it becomes easy to make a plan of the project.
4. Project plan, Project goal, risks, use-case model, Project description, are made.
5. Project is checked against the milestone criteria and if it couldn't pass these criteria then project can be either cancelled or redesigned.

B.Elaboration

1. Planning and modelling are main.
2. Detailed evaluation, development plan is carried out and diminish the risks.
3. Revise or redefine use-case model (approx. 80%), business case, risks.
4. Again, checked against milestone criteria and if it couldn't pass these criteria then again project can be cancelled or redesigned.
5. Executable architecture baseline.

C.Construction –

1. Project is developed and completed.
2. System or source code is created and then testing is done.
3. Coding takes place.

D.Transition –

1. Final project is released to public.
2. Transit the project from development into production.
3. Update project documentation.
4. Beta testing is conducted.
5. Defects are removed from project based on feedback from public.

E.Production –

1. Final phase of the model.
2. Project is maintained and updated accordingly.