SOFTWARE ENGINEERING - CS22502

KAVIYA P

CSE/AP

UNIT I - SOFTWARE PROCESS AND PROJECT MANAGEMENT

Software Engineering Process Paradigms
Phases and models of Software Development lifecycle
Project management
Process and Project Metrics
Software estimation
Empirical estimation models
planning Risk analysis
Software project scheduling and Tracking.

What is Software Engineering?

- **Software:** It is a collection of integrated programs. Means it carefully-organized instructions and code written by developers on any of various particular computer languages.
- **Engineering:** It is the application of scientific and practical knowledge to invent, design, build, maintain and improve frameworks, processes, etc.

- **Software Engineering:** It is an engineering branch related to the evolution of software product using well-defined scientific principles, techniques and procedures.

  The result of software engineering is an effective and reliable software product.

The term is made of two words, software and engineering.
Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **software product.**
Engineering is all about developing products, using well-defined, scientific principles and methods.

Definitions:

IEEE defines software engineering as:
The application of a systematic,disciplined,quantifiable approach to the development,operation and maintenance of software; that is, the application of engineering to software.

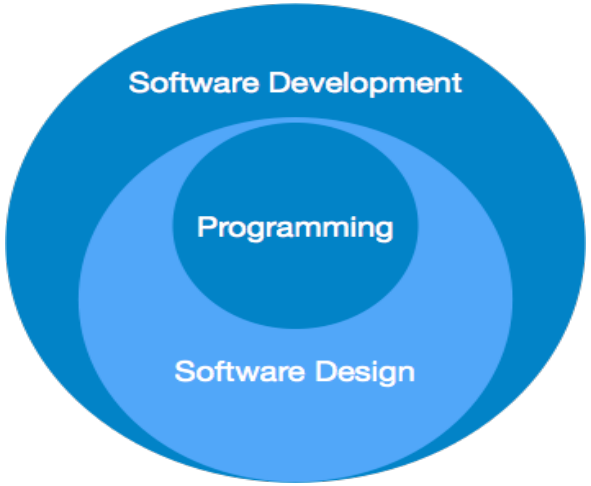
Fritz Bauer, a German computer scientist, defines software engineering as:
Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and work efficiently on real machines.

Software Evolution

- The process of developing a software product using software engineering **principles** and **methods** is referred to as **software evolution.** This includes the **initial development** of software and its **maintenance** and **updates**, till desired software product is developed, which satisfies the expected requirements.
- Evolution starts from the **requirement gathering** process. After which developers create a prototype of the intended software and show it to the users to get their feedback at the early stage of software product development.
- The users suggest changes, on which several consecutive updates and maintenance keep on changing too. This process changes to the original software, till the desired software is accomplished.
- Even after the user has desired software in hand, the advancing technology and the changing requirements force the software product to change accordingly. Re-creating software from scratch and to go one-on-one with requirement is not feasible. The only feasible and economical solution is to update the existing software so that it matches the latest requirements

Software Paradigms

- Software paradigms refer to the methods and steps, which are taken while designing the software.
- There are many methods proposed and are in work today, but we need to see where in the software engineering these paradigms stand. These can be combined into various categories, though each of them is contained in one another:
- Programming paradigm is a subset of Software design paradigm which is further a subset of Software development paradigm.



1. Software Development Paradigm:

- This Paradigm is known as software engineering paradigms where all the engineering concepts pertaining to the development of software are applied. It includes various **researches and requirement gathering which helps the software product to build**. It consists of –
- Requirement gathering
- Software design
- Programming

2. Software Design Paradigm:

This paradigm is a part of Software Development and includes –

- Design
- Maintenance
- Programming

3. Programming Paradigm:

- This paradigm is related closely to programming aspect of software development. This includes –
- Coding
- Testing
- Integration

Need of Software Engineering

1. **Handling Big Projects:** Corporation must use SE to handle large projects without any issues.
2. **To manage the cost:** Software engineering programmers plan everything and reduce all those things that are not required.
3. **To decrease time:** It will save a lot of time if you are developing software using a software engineering technique.
4. **Reliable software:** It is the company's responsibility to deliver software products on schedule and to address any defects that may exist.
5. **Effectiveness:** Effectiveness results from things being created in accordance with the software standards.

Characteristics of Good Software

Operational	Transitional	Maintenance
Budget	Interoperability	Flexibility
Efficiency	Reusability	Maintainability
Usability	Portability	Modularity
Dependability	Adaptability	Scalability
Correctness		
Functionality		
Safety		
Security		

Nature of Software

1. System Software:

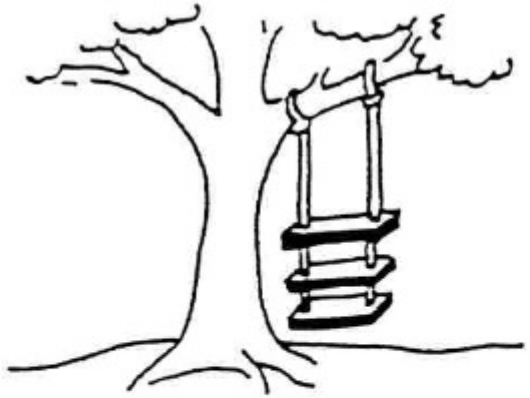
- System software is software designed to provide a platform for other software's.
- It is a interaction between hardware & application software.
- **Examples :** Operating Systems like macOS, Linux, Android and Microsoft Windows.

2. Application Software:

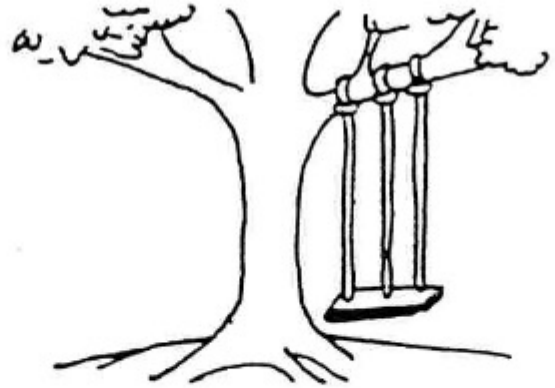
- Application Software is a computer program designed to carry out a specific task as per user & business need.
- **Examples:** Social medias apps, Gaming apps, Word processing apps, Multimedia apps, Banking apps, Shopping apps, Booking apps etc.

Scenerio

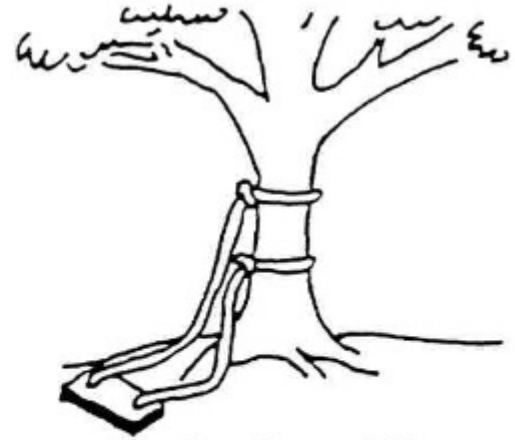
Design a swing under the tree…….



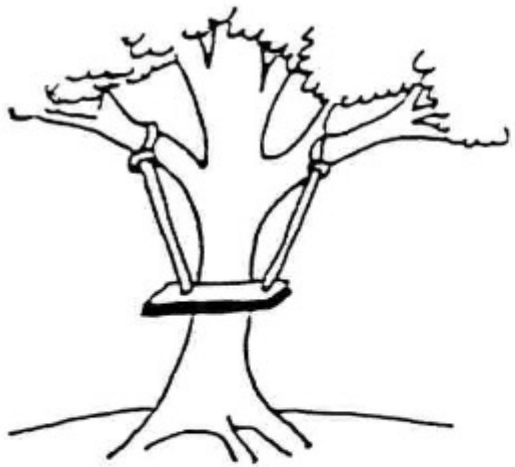
As proposed by the project sponsors



As specified in the project request

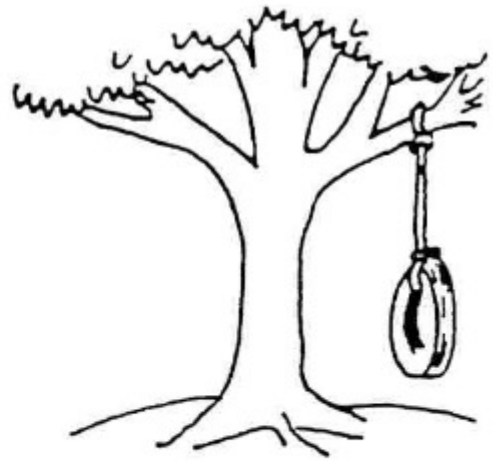


As designed by the senior analyst



As produced by the programmers

As installed at the user's site



What the user wanted

Software Development Life Cycle

Software Development Life Cycle, SDLC for short, is a well-defined, structured sequence of stages in software engineering to develop the intended software product.

SDLC Activities

- SDLC provides a series of steps to be followed to design and develop a software product efficiently. SDLC framework includes the following steps:

Phases Involved in SDLC

- In the software development cycle, the stakeholders, team leads, developers, and other resources are defined with the allocated task with the resources to meet the deadlines of any project. Although for every SDLC several steps are involved, the list composes of 6 phases, which are:
- Planning
- Defining Requirements
- Designing Architecture
- Development of Product
- Testing and Integration
- Deployment and Maintenance

Communication:

- This is the first step where the user initiates the request for a desired software product. User contacts the service provider and tries to negotiate the terms. User submits request to the service providing organization in writing.

Requirement Gathering:

- This step onwards the software development team works to carry on the project. The team holds discussions with various stakeholders from problem domain and tries to bring out as much information as possible on their requirements. The requirements are contemplated and segregated into **user requirements, system requirements and functional requirements**. The requirements are collected using a number of practices as given –
- studying the existing or obsolete system and software,
- conducting interviews of users and developers,
- referring to the database
- collecting answers from the questionnaires.

Feasibility Study

- After requirement gathering, the team comes up with a rough plan of software process.
- At this step the team analyzes if a software can be made to fulfill all requirements of the user and if there is any possibility of software being no more useful.
- It is found out, if the project is financially, practically and technologically feasible for the organization to take up.
- There are many algorithms available, which help the developers to conclude the feasibility of a software project.

System Analysis

- At this step the developers decide a roadmap of their plan and try to bring up the best software model suitable for the project.
- System analysis includes Understanding of software product limitations, learning system related problems or changes to be done in existing systems beforehand, identifying and addressing the impact of project on organization and personnel etc.
- The project team analyzes the scope of the project and plans the schedule and resources accordingly.

Software Design

- Next step is to bring down whole knowledge of requirements and analysis on the desk and design the software product.
- The inputs from users and information gathered in requirement gathering phase are the inputs of this step.
- The output of this step comes in the form of two designs; logical design and physical design. Engineers produce meta-data and data dictionaries, logical diagrams, data-flow diagrams and in some cases pseudo codes.

Coding

- This step is also known as programming phase.
- The implementation of software design starts in terms of writing program code in the suitable programming language and developing error-free executable programs efficiently.

Testing:

- An estimate says that 50% of whole software development process should be tested.
- Errors may ruin the software from critical level to its own removal.
- Software testing is done while coding by the developers and thorough testing is conducted by testing experts at various levels of code such as module testing, program testing, product testing, in-house testing and testing the product at user's end.
- Early discovery of errors and their remedy is the key to reliable software.

Integration:

- Software may need to be integrated with the libraries, databases and other programs. This stage of SDLC is involved in the integration of software with outer world entities.

Implementation:

- This means installing the software on user machines.
- At times, software needs post-installation configurations at user end. Software is tested for portability and adaptability and integration related issues are solved during implementation.

Operation and Maintenance:

- This phase confirms the software operation in terms of more efficiency and less errors.
- If required, the users are trained on, or aided with the documentation on how to operate the software and how to keep the software operational.
- The software is maintained timely by updating the code according to the changes taking place in user end environment or technology.
- This phase may face challenges from hidden bugs and real-world unidentified problems.

Disposition:

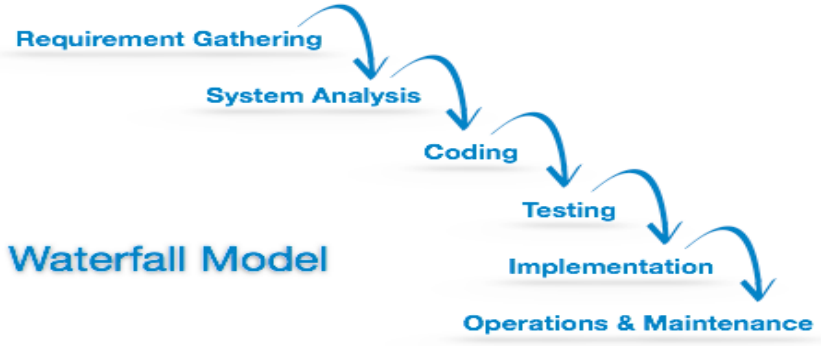
- As time elapses, the software may decline on the performance front.
- It may go completely obsolete or may need intense upgradation.
- This phase includes archiving data and required software components, closing down the system, planning disposition activity and terminating system at appropriate end-of-system time.

Software Development Paradigm

- The software development paradigm helps developer to **select a strategy to develop the software.**
- A software development paradigm has its own **set of tools, methods and procedures,** which are expressed clearly and defines software development life cycle.
- A few of software development paradigms or process models are defined as follows:
- **Waterfall Model**
- **Agile**
- **Iterative Model**
- **Spiral Model**
- **V – model**
- **Big Bang Model**

Waterfall Model

- Waterfall model is the simplest model of software development paradigm. It says the **all the phases of 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 assumes that everything is carried out and taken place perfectly as planned in the previous stage and **there is no need to think about the past issues that may arise in the next phase.**
- This model **does not work smoothly** if there are some issues left at the previous step. The sequential nature of **model does not allow us go back and undo or redo our actions.**
- This model is best suited when developers already have designed and developed similar software in the past and are aware of all its domains.

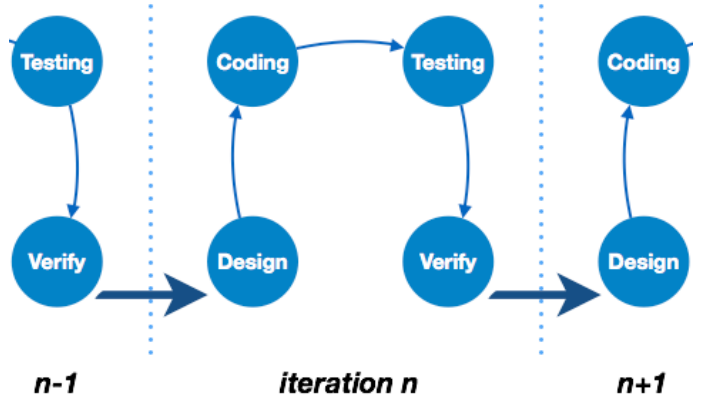


- **Requirement Gathering and analysis** − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
- **System Design** − The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
- **Implementation** − With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.
- **Integration and Testing** − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- **Deployment of system** − Once the testing is done; the product is deployed in the customer environment or released into the market.
- **Maintenance** − There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.
- All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases.
- The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

Functional Testing	Non-functional Testing
It verifies the operations and actions of an application.	It verifies the behavior of an application.
It is based on requirements of customer.	It is based on expectations of customer.
It helps to enhance the behavior of the application.	It helps to improve the performance of the application.
Functional testing is easy to execute manually.	It is hard to execute non-functional testing manually.
It tests what the product does.	It describes how the product does.
Functional testing is based on the business requirement.	Non-functional testing is based on the performance requirement.
Examples: **1.** Unit Testing **2.** Smoke Testing **3.** Integration Testing **4.** Regression Testing	**Examples:** **1.** Performance Testing **2.** Load Testing **3.** Stress Testing **4.** Scalability Testin

Iterative Model

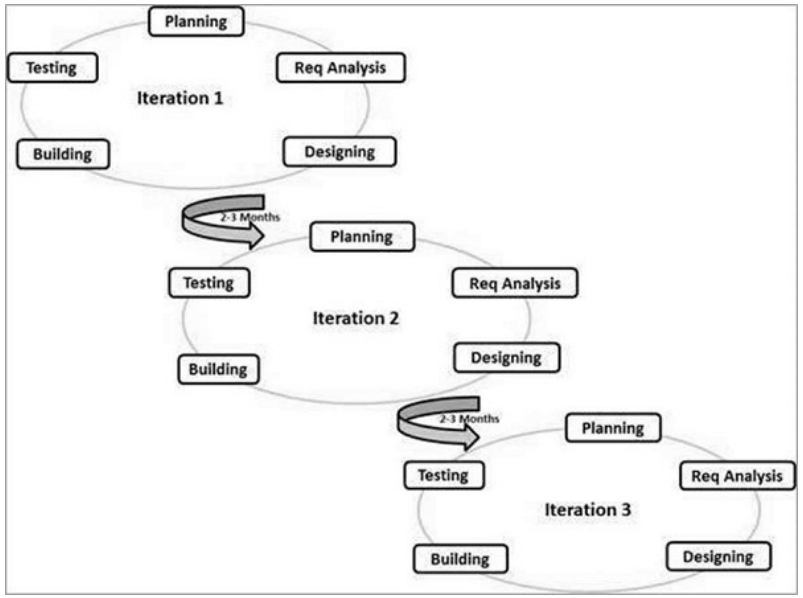
- This model leads the software development process in **iterations**. It projects the process of development in **cyclic manner repeating every step after every cycle of SDLC process**.
- The software is first developed on very small scale and all the steps are followed which are taken into consideration.
- Then, on every next iteration, more features and modules are designed, coded, tested and added to the software.
- Every cycle produces a software, which is complete in itself and has more features and capabilities than that of the previous one.
- After each iteration, the management team can do work on **risk management** (process of recognizing, evaluating, and handling threats or risks that have an effect overall operations)and prepare for the next iteration.
- Because a cycle includes small portion of whole software process, it is easier to manage the development process but it consumes more resources.



Agile

- Agile SDLC model is a combination of **iterative and incremental process models** with focus on process adaptability and customer satisfaction by rapid delivery of working software product.
- Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three Months. Every iteration involves cross functional teams working simultaneously on various areas like −

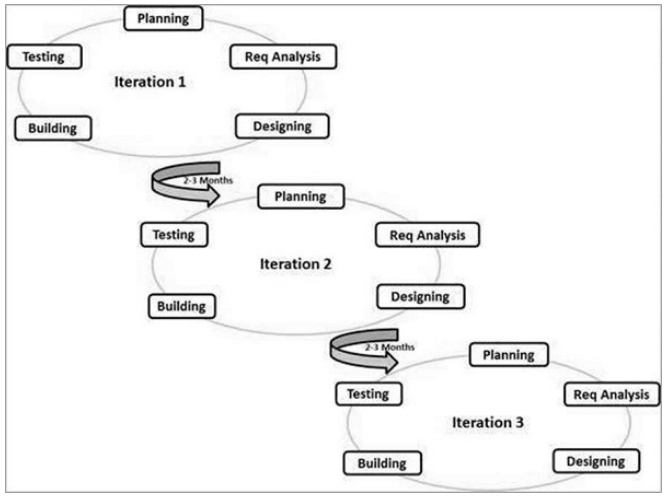
1. Planning
2. Requirements Analysis
3. Design
4. Coding
5. Unit Testing
6. Acceptance Testing.



At the end of the iteration, a working product is displayed to the customer and important stakeholders.

What is Agile?

- Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements.
- In Agile, the tasks are divided to time boxes (small time frames) to deliver specific features for a release.
- Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

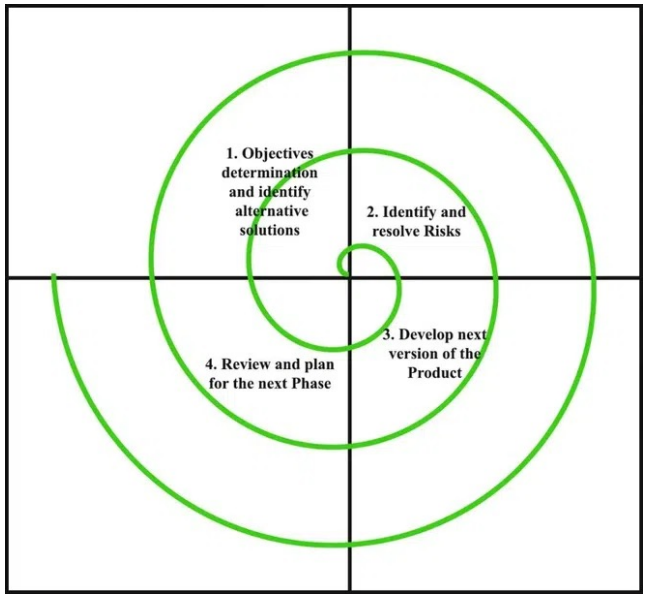


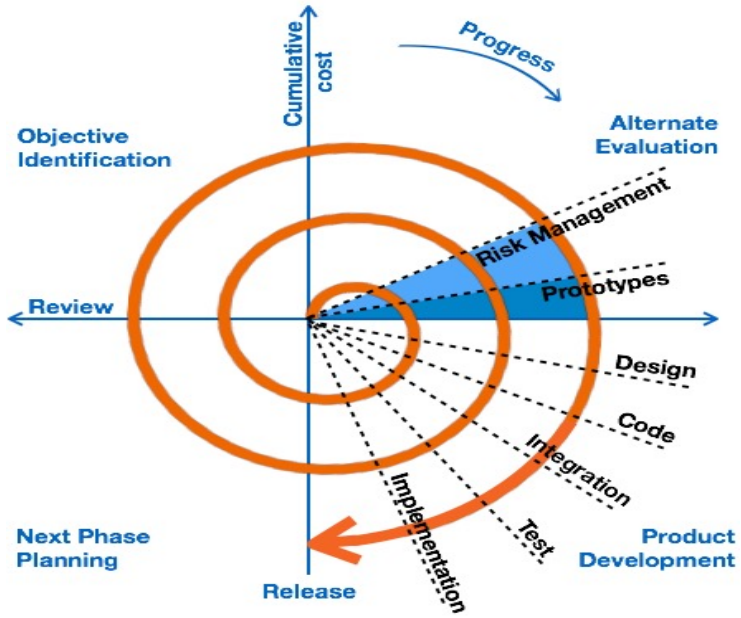
Spiral Model

- The spiral model combines the idea of **iterative development** with the systematic, controlled aspects of the **waterfall model**.
- This Spiral model is a combination of iterative development process model and sequential linear development model i.e. the waterfall model with a very high emphasis on risk analysis.

Spiral Model - Design

- The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.





Identification

- This phase starts with gathering the business requirements in the baseline spiral.
- In the subsequent spirals as the **product matures, identification of system requirements, subsystem requirements and unit requirements** are all done in this phase.
- This phase also includes understanding the system requirements by **continuous communication** between the customer and the system analyst.
- At the end of the spiral, the product is deployed in the identified market.

Design

- The Design phase starts with the conceptual design in the baseline spiral and involves **architectural design, logical design of modules, physical product design and the final design** in the subsequent spirals.

Construct or Build

- The Construct phase refers to production of the actual software product at every spiral.
- In the baseline spiral, when the product is just thought of and the design is being developed a **POC (Proof of Concept)** is developed in this phase to get **customer feedback**.
- Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number.
- These builds are sent to the customer for feedback.

Evaluation and Risk Analysis

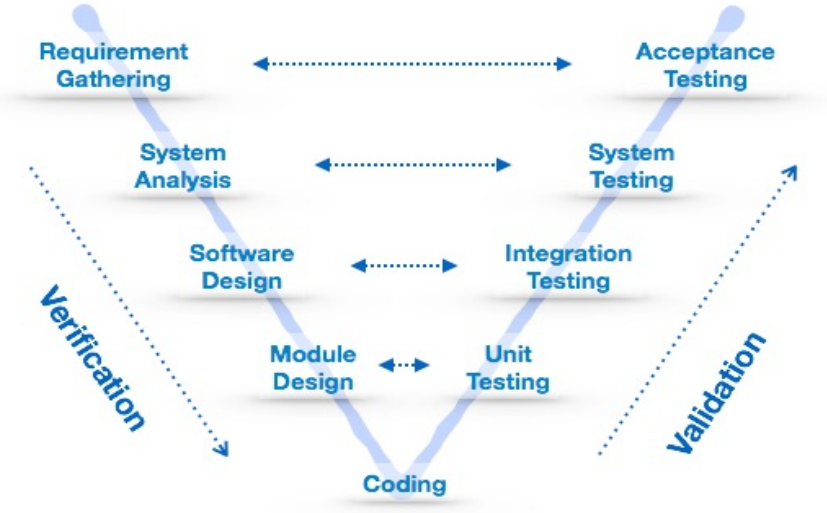
- Risk Analysis includes **identifying, estimating and monitoring the technical feasibility and management risks**, such as schedule slippage and cost overrun.
- After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.
- Based on the **customer evaluation,** the software development process enters the next iteration and subsequently follows the linear approach to implement the feedback suggested by the customer.
- The process of iterations along the spiral continues throughout the life of the software.

V – model

- The major drawback of waterfall model is we move to the next stage only when the previous one is finished and there was no chance to go back if something is found wrong in later stages.
- V-Model provides means of testing of software at each stage in reverse manner.
- At every stage, test plans and test cases are created to **verify and validate** the product according to the requirement of that stage.
- For example, in requirement gathering stage the test team prepares all the test cases in correspondence to the requirements.
- Later, when the product is developed and is ready for testing, test cases of this stage verify the software against its validity towards requirements at this stage.
- This makes both **verification and validation** go in parallel. This model is also known as verification and validation model.
- **Verification and Validation** is the process of investigating whether a software system satisfies specifications and standards and fulfills the required purpose.
- ***Verification:*** *Are we building the product right?*
- ***Validation:*** *Are we building the right product?*

V-Model - Design

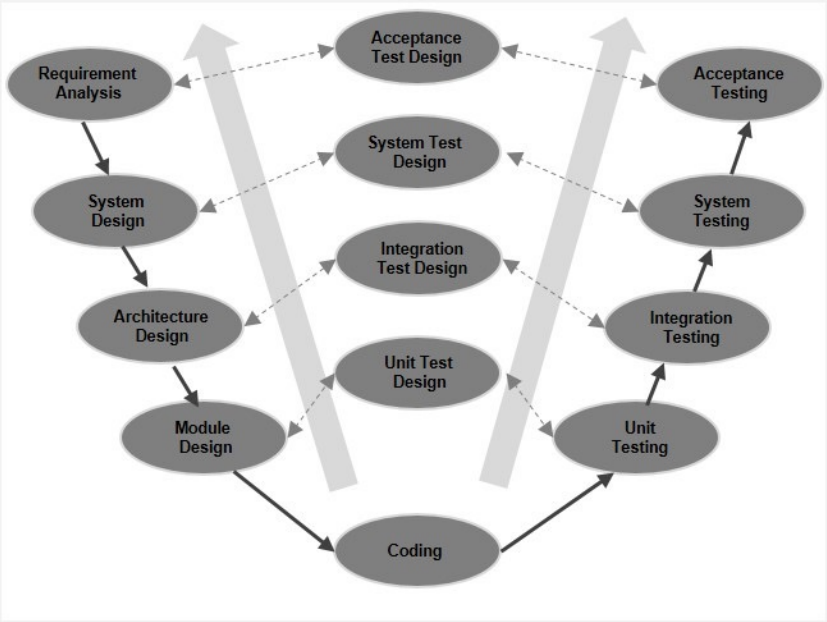
So, there are Verification phases on one side of the 'V' and Validation phases on the other side. The Coding Phase joins the two sides of the V-Model.



V-Model - Verification Phases

Business Requirement Analysis

- This is the first phase in the development cycle where the product requirements are understood from the customer's perspective.
- This phase involves detailed communication with the customer to understand his expectations and exact requirement.
- This is a very important activity and needs to be managed well, as most of the customers are not sure about what exactly they need.
- The **acceptance test design planning** is done at this stage as business requirements can be used as an input for acceptance testing.



System Design

- Once you have the clear and detailed product requirements, it is time to design the complete system. The system design will have the understanding and detailing the complete hardware and communication setup for the product under development.
- The system test plan is developed based on the system design. Doing this at an earlier stage leaves more time for the actual test execution later.

Architectural Design

- Architectural specifications are understood and designed in this phase. Usually more than one technical approach is proposed and based on the technical and financial feasibility the final decision is taken.
- The system design is broken down further into modules taking up different functionality. This is also referred to as **High Level Design (HLD)**.
- The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood and defined in this stage. With this information, integration tests can be designed and documented during this stage.

Module Design

- In this phase, the detailed internal design for all the system modules is specified, referred to as **Low Level Design (LLD)**.
- It is important that the design is compatible with the other modules in the system architecture and the other external systems.
- The unit tests are an essential part of any development process and helps eliminate the maximum faults and errors at a very early stage. These unit tests can be designed at this stage based on the internal module designs.

Coding Phase

- The actual coding of the system modules designed in the design phase is taken up in the Coding phase.
- The best suitable programming language is decided based on the system and architectural requirements.
- The coding is performed based on the coding guidelines and standards.
- The code goes through numerous code reviews and is optimized for best performance before the final build is checked into the repository.

Validation Phases

Unit Testing

- Unit tests designed in the module design phase are executed on the code during this validation phase. Unit testing is the testing at code level and **helps eliminate bugs** at an early stage, though all defects cannot be uncovered by unit testing.

Integration Testing

- Integration testing is associated with the architectural design phase. Integration tests are performed to **test the coexistence and communication of the internal modules** within the system.

System Testing

- System testing is directly associated with the system design phase. System tests check the entire system functionality and the communication of the system under development with external systems. Most of the **software and hardware compatibility issues** can be uncovered during this system test execution.

Acceptance Testing

- Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests **uncover the compatibility issues with the other systems available** in the user environment. It also discovers the non-functional issues such as **load and performance defects** in the actual user environment.

Big Bang Model

- This model is the simplest model in its form. It **requires little planning, lots of programming and lots of funds**.
- As scientists say that after big bang lots of galaxies, planets and stars evolved just as an event. Likewise, if we put together **lots of programming and funds**, you may achieve the best software product.
- The Big Bang model is an SDLC model where we **do not follow any specific process**.
- The development just starts with the required **money and efforts as the input**, and the **output is the software** developed which **may or may not be as per customer requirement**.
- This Big Bang Model **does not follow a process/procedure** and there is a **very little planning** required. Even the customer is not sure about what exactly he wants and the requirements are implemented on the fly without much analysis.

Big Bang Model – Design

- The Big Bang Model comprises of focusing all the possible resources in the software development and coding, with very little or no planning. The requirements are understood and implemented as they come. Any changes required may or may not need to change the complete software.
- This model is ideal for small projects with one or two developers working together and is also useful for academic or practice projects. It is an ideal model for the product where **requirements are not well understood and the final release date is not given**.

Software Project Management and Project Metrics

- The job pattern of an IT company engaged in software development can be seen split in two parts:

1. **Software Creation**
2. **Software Project Management**

- A project is well-defined task, which is a collection of several operations done in order to achieve a goal (for example, **software development and delivery**).

A Project can be characterized as:

- Every project may has a unique and distinct goal.
- Project is not routine activity or day-to-day operations.
- Project comes with a start time and end time.
- Project ends when its goal is achieved hence it is a temporary phase in the lifetime of an organization.
- Project needs adequate resources in terms of time, manpower, finance, material and knowledge-bank

Software Project

- A Software Project is the complete procedure of software development from requirement gathering to testing and maintenance, carried out according to the execution methodologies, in a specified period of time to achieve intended software product.

Need of software project management

- Software is said to be an intangible product.
- Software development is a kind of all new stream in world business and there's very little experience in building software products. Most software products are tailor made to fit client's requirements.
- The most important is that the underlying technology changes and advances so frequently and rapidly that experience of one product may not be applied to the other one.
- All such business and environmental constraints bring risk in software development hence it is essential to manage software projects efficiently.
- The image shows triple constraints for software projects. It is an essential part of software organization to deliver quality product, keeping the cost within client's budget constrain and deliver the project as per scheduled.
- There are several factors, both internal and external, which may impact this triple constrain triangle. Any of three factor can severely impact the other two.
- Therefore, software project management is essential to incorporate user requirements along with budget and time constraints.

Software Project Manager

- A software project manager is a person who undertakes the responsibility of executing the software project.
- Software project manager is thoroughly aware of all the phases of SDLC that the software would go through.
- Project manager may never directly involve in producing the end product but he controls and manages the activities involved in production.
- A project manager closely monitors the development process, prepares and executes various plans, arranges necessary and adequate resources, maintains communication among all team members in order to address issues of cost, budget, resources, time, quality and customer satisfaction.

Let us see few responsibilities that a project manager shoulders -

Managing People

- Act as project leader
- Liaison (someone who helps groups to work effectively with each other) with stakeholders
- Managing human resources
- Setting up reporting hierarchy etc.

Managing Project

- Defining and setting up project scope
- Managing project management activities
- Monitoring progress and performance
- Risk analysis at every phase
- Take necessary step to avoid or come out of problems
- Act as project spokesperson

Software Management Activities

- Software project management comprises of a number of activities, which contains planning of project, deciding scope of software product, estimation of cost in various terms, scheduling of tasks and events, and resource management.
- Project management activities may include:

1. **Project Planning**
2. **Scope Management**
3. **Project Estimation**

Project Planning

- Software project planning is task, which is performed before the production of software actually starts.
- It is there for the software production but involves no concrete activity that has any direction connection with software production; rather it is a set of multiple processes, which facilitates software production.

Scope Management

- It defines the scope of project; this includes all the activities, process need to be done in order to make a deliverable software product.
- Scope management is essential because it creates boundaries of the project by clearly defining what would be done in the project and what would not be done.
- This makes project to contain limited and quantifiable tasks, which can easily be documented and in turn avoids cost and time overrun.

- Define the scope
- Decide its verification and control
- Divide the project into various smaller parts for ease of management.
- Verify the scope
- Control the scope by incorporating changes to the scope

Project Estimation

- For an effective management accurate estimation of various measures is a must. With correct estimation managers can manage and control the project more efficiently and effectively.

Effort estimation

- The managers estimate efforts in terms of personnel requirement and man-hour required to produce the software.
- For effort estimation software size should be known. This can either be derived by managers' experience, organization's historical data or software size can be converted into efforts by using some standard formulae.
- **Time estimation** : Once size and efforts are estimated, the time required to produce the software can be estimated.
- Efforts required is segregated into sub categories as per the requirement specifications and interdependency of various components of software.

Cost estimation

- This might be considered as the most difficult of all because it depends on more elements than any of the previous ones. For estimating project cost, it is required to consider –

    - Size of software
    - Software quality
    - Hardware
    - Additional software or tools, licenses etc.
    - Skilled personnel with task-specific skills
    - Travel involved
    - Communication
    - Training and support

3. Effective software project management

Focuses on the four P's: people, product, process, and project.

1) The People

- The people management defines the following key practice areas for software people

- RECRUITING
- SELECTION
- PERFORMANCE MANAGEMENT
- TRAINING
- COMPENSATION
- CAREER DEVELOPMENT
- ORGANIZATION AND WORK DESIGN
- TEAM/CULTURE DEVELOPMENT.

❖ The Players(Stack holders):

The software process (and every software project) is populated by players who can be categorized into one of five constituencies:

1. **Senior managers:** who define the business issues that influence on the project.
2. **Project** (technical) **managers** :who must plan, motivate, organize, and control the practitioners who do software work.
3. **Practitioners** who deliver the technical skills that are helps to develop a product or application.
4. **Customers** who specify the requirements for the software to be engineered(developed)
5. **End-users** who interact with the software once it is released for Production use.

❖ Team Leaders:

- ✓ Project management is a people-intensive activity
- ✓ MOI model of leadership:

- ✓ **Motivation.** The ability to encourage (by "push or pull") technical people to produce to their best ability.
- ✓ **Organization.** The ability to mold existing processes (or invent new ones) that will enable the initial concept to be translated into a final product.
- ✓ **Ideas or innovation**. The ability to encourage people for development of a particular software product or application

The Software Team:

Many human organizational structures for software development as there are organizations that develop software

Project factors that should be considered when planning the structure of software engineering teams:

- The difficulty of the problem to be solved.
- The size of the resultant program(s) in lines of code or function points
- The time that the team will stay together (team lifetime).
- The degree to which the problem can be modularized.
- The required quality and reliability of the system to be built
- The rigidity of the delivery date.

- **Agile Team:**
  - An Agile team is a self organizing team that has anatomy to plan and make technical decisions.
- **Coordination and Communication Issues:**
- There are many reasons that software projects get into trouble.
- Interoperability has become a key characteristic of many systems.
- New software must communicate with existing software and conform to predefined constraints imposed by the system or product.
- To deal with them effectively, a software engineering team must establish effective methods for coordinating the people who do the work.
- Formal communication is accomplished through —writing, structured meetings, and impersonal communication channels (emails, public announcements).
- Electronic communication encompasses electronic mail, electronic bulletin boards, and by extension, video-based conferencing systems.

2)The Product: Before a project can be planned

✓product objectives

✓scope should be established,

✓alternative solutions should be considered,

✓technical and management constraints should be identified. Without this information, it is impossible to estimates the cost.

Software Scope:

- The first software project management activity is the determination of software scope.
- Scope is defined by answering the following questions:
- **Context.** How does the software to be built fit into a larger system, product, or business context and what constraints are imposed as a result of the context?
- **Information objectives.** What customer-visible data objects are produced as output from the software? What data objects are required for input?
- **Function and performance.** What function does the software perform to transform input data into output? Are any special performance characteristics to be addressed?

Problem Decomposition:

Problem decomposition, sometimes called partitioning or problem elaboration, is an activity that sits at the core of software requirements analysis .

- Decomposition is applied in two major areas:

  (1) The functionality that must be delivered and

  (2) The process that will be used to deliver it.

- Software functions, described in the statement of scope, are evaluated and refined to provide more detail prior to the beginning of estimation

- 3) The Process
- A software process provides the framework from which a comprehensive plan for software development can be established..
- umbrella activities—such as software quality assurance, software configuration management, and measurement—overlay the process model
- A wide array of software engineering paradigms include:
- The linear sequential model
- The prototyping model
- The RAD model
- The incremental model
- The spiral model

The project manager must decide which process model is most appropriate for
(1)The customers who have requested the product and the people who will do the work.

(2) The characteristics of the product itself.

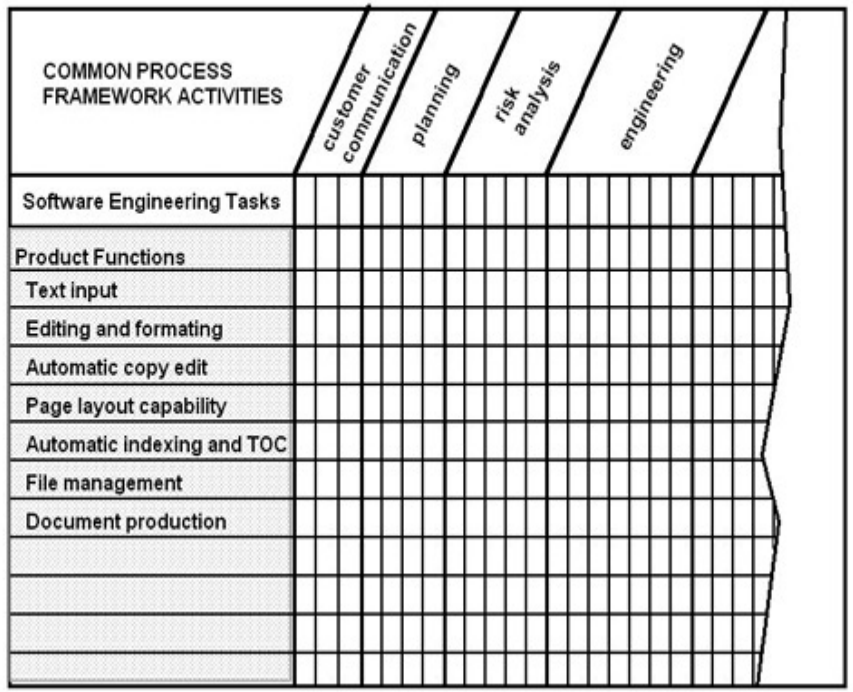
(3) The project environment in which the software team works.

Process Decomposition

- Once the process model has been chosen, the common process framework (CPF) is adapted to it.
- How do we accomplish this CPF activity?
- For example, a small ,relatively simple project might require the following work tasks for the customer communication activity:

1. Develop list of clarification issues.
2. Meet with customer to address clarification issues.
3. Jointly develop a statement of scope.
4. Review the statement of scope with all concerned.
5. Modify the statement of scope as required.

Melding Problem and Process



4)The Project

- ✓ In order to avoid project failure, a software project manager and the software engineers who build the product must avoid a set of common warning signs that lead to good project management.
- ✓ In order to manage a successful software project, we must understand what can go wrong and how to do it right.
- ✓ In a software projects one defines ten signs that indicate that an information systems project is in jeopardy:

1. Software people don't understand their customer's needs.
2. The product scope is poorly defined.
3. Changes are managed poorly.
4. The chosen technology changes.
5. Business needs change [or are ill-defined].
6. Deadlines are unrealistic.
7. Users are resistant.
8. Sponsorship is lost [or was never properly obtained].
9. The project team lacks people with appropriate skills.
10. Managers [and practitioners] avoid best practices and lessons learned.

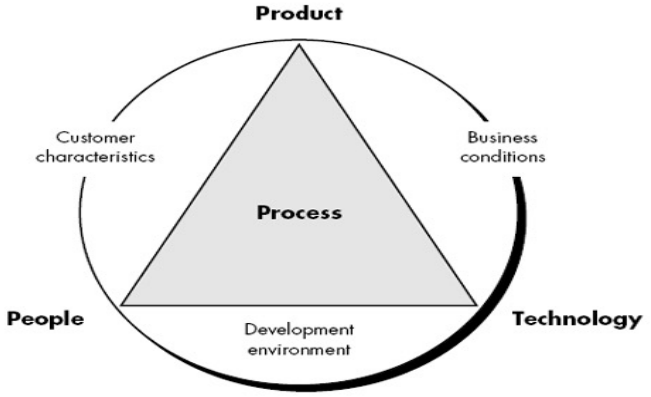
PROCESS AND PROJECT METRICS

- What are Process and Project Metrics
- What are the Categories of Software Measurement
- Metrics for Software Quality Correctness
- Integrating Metrics within the Software Process

PROCESS AND PROJECT METRICS

Metrics

- Software process and project metrics are quantitative measures
- They are a management tool
- They offer the effectiveness of the software process and the projects that are conducted using the process as a framework.
- Basic **quality** and **productivity data** are collected
- These data are **analyzed, compared against past averages, and assessed**
- The **goal** is to determine whether quality and productivity improvements have occurred
- The data can also be used to pinpoint problem areas
- Remedies can then be developed and the software process can be improved



Software Metrics

- A software metric is a measure of software characteristics which are measurable or countable.
- Software metrics are valuable for many reasons, including **measuring software performance, planning work items, measuring productivity**, and many other uses.
- Within the software development process, many metrics are that are all connected. Software metrics are similar to the four functions of management**: Planning, Organization, Control, or Improvement**.

Classification of Software Metrics

- **Software metrics can be classified into two types as follows:**

1. Product Metrics: These are **the measures of various characteristics of the software product**. The two important software characteristics are:

1.Size and **complexity** of software.

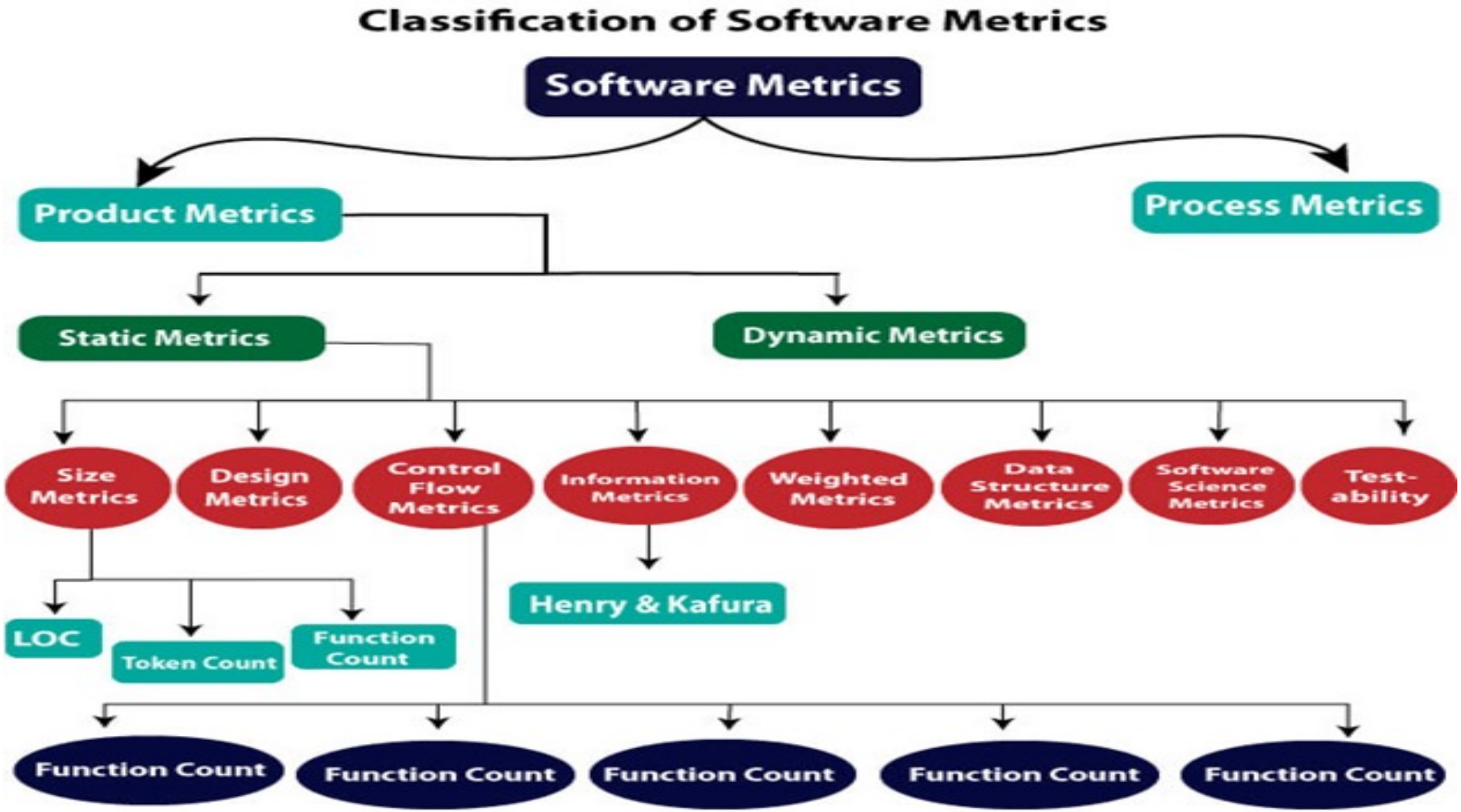
2.Quality and **reliability** of software.

- These metrics can be computed for different stages of SDLC.

2. Process Metrics: These are the measures of various characteristics of the software development process.

For example, the efficiency of fault detection.

They are used to measure the **characteristics of methods, techniques, and tools** that are used for developing software.

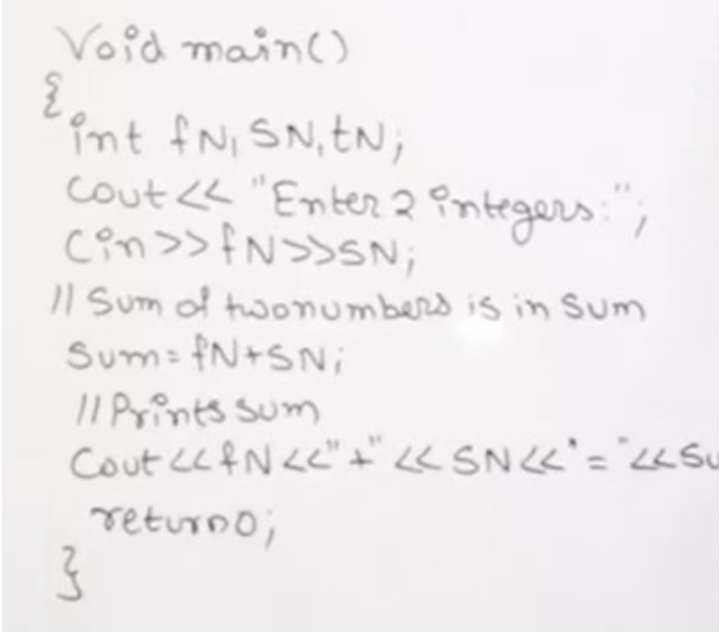

Classification of Software Metrics
Software Metrics
Product Metrics
Process Metrics
Static Metrics
Dynamic Metrics
Size Metrics
Design Metrics
Control Flow Metrics
Information Metrics
Weighted Metrics
Data Structure Metrics
Software Science Metrics
Test-ability
LOC
Token Count
Function Count
Henry & Kafura
Function Count
Function Count
Function Count
Function Count
Function Count

Types of Metrics:

- **Internal metrics:** Internal metrics are the metrics used for **measuring properties that are viewed to be of greater importance to a software developer**. For example, Lines of Code (LOC) measure.
- **External metrics:** External metrics are the metrics used for measuring properties that are viewed to be of greater importance to the user, e.g., portability, reliability, functionality, usability, etc.
- **Hybrid metrics:** Hybrid metrics are the metrics that combine product, process, and resource metrics. For example, cost per FP where FP stands for Function Point Metric.
- **Project metrics:** Project metrics are the metrics used by the project manager to check the project's progress.
- Data from the past projects are used to collect various metrics, like time and cost; these estimates are used as a base of new software.
- Note that as the project proceeds, the project manager will check its progress from time-to-time and will compare the effort, cost, and time with the original effort, cost and time.
- Also understand that these metrics are used to decrease the development costs, time efforts and risks. The project quality can also be improved. As quality improves, the number of errors and time, as well as cost required, is also reduced.

Line of Code(LoC)

- Introduction
- What do we count in LoC and what do we don't?
- Why blank spaces and comments are included in code?
- Why we don't count blank spaces and comments?
- Advantages and Disadvantages



Metrics in the **Process Domain**

Process metrics are collected across all projects and over long periods of time.

- They are used for making strategic decisions.
- The only way to know how/where to improve any process is to

– Measure specific attributes of the process.

– Develop a set of meaningful metrics based on these attributes.

– Use the metrics to provide indicators that will lead to a strategy for improvement.

- We measure the effectiveness of a process by deriving a set of metrics based on outcomes of the process such as

– Errors uncovered before release of the software

– Defects delivered to and reported by the end users

– Work products delivered

– Human effort expended

– Calendar time expended

– Conformance to the schedule

– Time and effort to complete each generic activity

Etiquette of Process Metrics

- Use common sense and organizational sensitivity when interpreting metrics data
- Provide regular feedback to the individuals and teams who collect measures and metrics
- Don’t use metrics to evaluate individuals
- Work with practitioners and teams to set clear goals and metrics that will be used to achieve them
- Never use metrics to threaten individuals or teams

Metrics in the Project Domain

- Project metrics enable a software project manager to
  - Assess the status of an ongoing project
  - Track potential risks
  - Uncover problem areas before their status becomes critical
  - Adjust work flow or tasks
  - Evaluate the project team’s ability to control quality of software work products
- Many of the same metrics are used in both the process and project domain
- Project metrics are used for making tactical decisions
- – They are used to adapt project workflow and technical activities

Use of Project Metrics

- The first application of project metrics occurs **during estimation**

– Metrics from past projects are used as a basis for **estimating time and effort**

– As a project proceeds, the amount of time and effort expended are compared to original estimates

- Project metrics are used to

– Minimize the development schedule by making the adjustments necessary to avoid delays and mitigate potential problems and risks

– Assess product quality on an ongoing basis and, when necessary, to modify the technical approach to improve quality

Categories of Software Measurement

- Standard of measure that contains many activities which involve some degree of measurement
  - Direct Metrics: Immediately measurable Eg: Line of Code
  - Indirect Metrics: Not immediately quantifiable Eg: Functionality

- **Product Metrics**
- Describes the characteristics of the product
  - size
  - Complexity
  - Design features
  - Performance
  - Quality level
  - Reliability
  - Functionalities

- **Process Metrics**
- Used to improve the development process and maintenance activities of the software
  - Effort required
  - Time to produce the product
  - No of defects found
  - Tools and tech
  - Quality
  - Efficiency

- **Project Metrics**
- Describes the project characteristics and execution
  - No of s/w developers
  - Staffing patterns
  - Cost
  - Schedule
  - Productivity
  - Quality
  - Assess status of ongoing project

2. Categories of Software Measurement

- There are the two Categories of Software Measurement

– **Direct measures of the**

- Software process (cost, effort, etc.)
- Software product (lines of code produced, execution speed, defects reported over time, etc.)

– Indirect measures of the

- Software product (functionality, quality, complexity, efficiency, reliability, maintainability, etc.)

Project metrics can be consolidated to create process metrics for an organization by using **Software Measurement**

Size Oriented Metrics (KLOC Measurements)

Metrics for Software Cost and Effort Estimation

Size-oriented Metrics

• Derived by normalizing quality and/or productivity measures by considering the size of the software produced.

• Thousand lines of code (KLOC) are often chosen as the normalization value

• Metrics include

– Errors per KLOC

- Errors per person-month

– Defects per KLOC

- KLOC per person-month

– Dollars per KLOC

- Dollars per page of documentation

– Pages of documentation per KLOC

• Size-oriented metrics are not universally accepted as the best way to measure the software process

• Opponents argue that KLOC measurements

– Are dependent on the programming language

– Penalize well-designed but short programs

– Cannot easily accommodate nonprocedural languages

– Require a level of detail that may be difficult to achieve

Advantages:

- easy to count or calculate from developed code

Issues in Estimation using Size Oriented Metrics

- Programming Language Dependent
- Penalize well-designed but short programs
- Not universally accepted as a best way to measure software process

Size oriented metrics

Project	LOC	Effort	$(000)	Pp. doc.	Errors	Defects	People
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6
•	•	•	•	•	•		
•	•	•	•	•	•		
•	•	•	•	•	•		

Function Oriented Metrics

Metrics for Software Cost and Effort Estimation

- Measure of the **functionality** delivered by the application
- Most widely used metric of this type is the **Function Point**
- Using historical data it can:
  - Estimate the cost or effort required to design, code, and test the software
  - Predict the number of errors that will be encountered during testing
  - Forecast the number of components and/or the number of projected source lines in the implemented system

Advantages over Size Oriented Metrics:

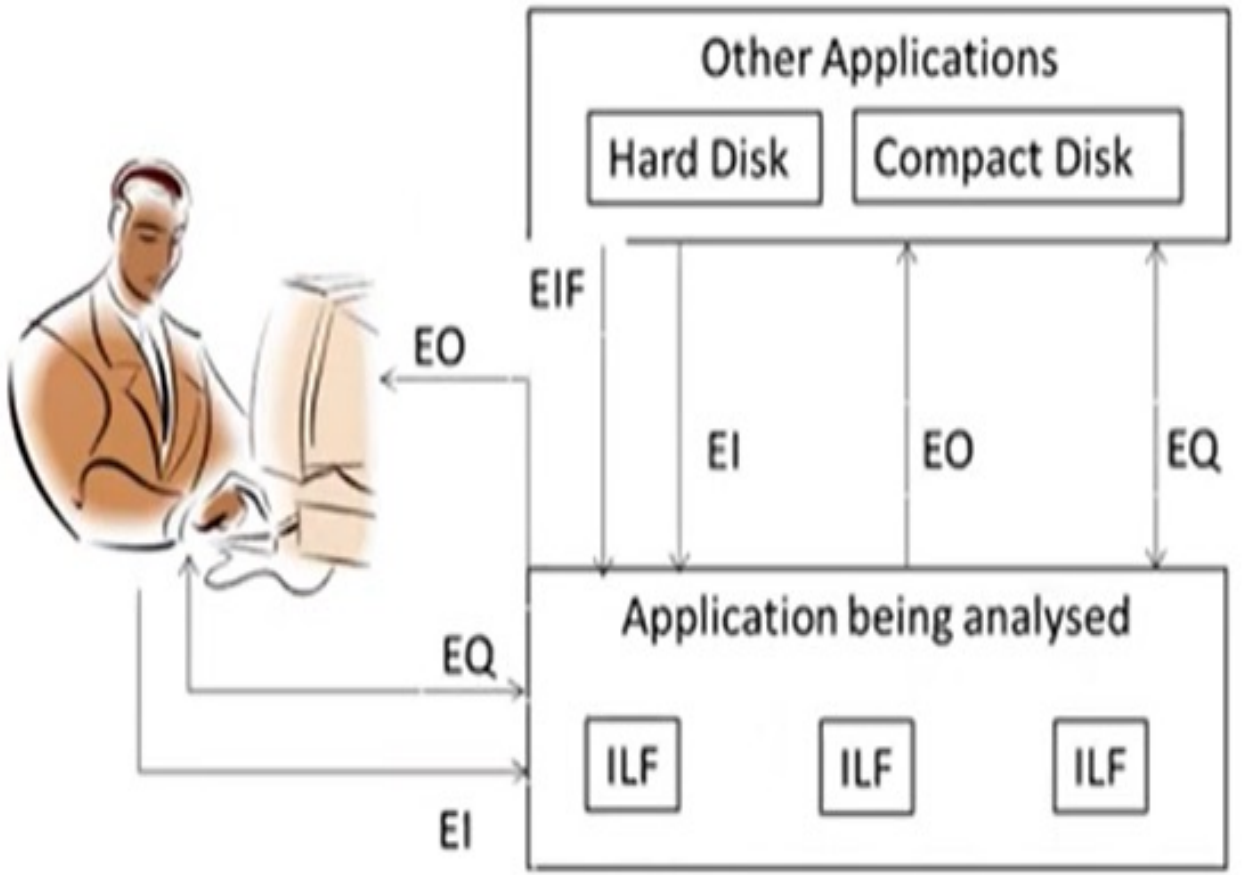
- Programming language independent
- Based on data that are more likely to be known in the early stages of a project

Disadvantages

- Computation is based on subjective data
- Counts of the information domain can be difficult to collect
- Has no direct physical meaning, it's just a number

Principle of FPA

- System is decomposed into five functional units
  - External Inputs (EI)
  - External Outputs (EO)
  - External Enquiries (EQ)
  - Internal Logical Files (ILF)
  - External Interface Files (EIF)



Users view of the system

Function Oriented Metrics

Metrics for Software Cost and Effort Estimation

Calculations

- FPA = UFP*CAF
- UFP = $\sum_{i=1\ j=1}^{i=5,j=3} W_{ij} * C_{ij}$
- CAF=0.65+[0.01*$\sum F_i$]

Function Point Analysis (FPA)
Unadjusted Function Point (UFP)
Complexity Adjustment Factor (CAF)
Weighting Factor (W)
Count (C)
Total Complexity Adjustment Value (F)

FPA Matrix

Information Domain Value	Weighting factor		
	Simple	Average	Complex
External Inputs (EIs)	3	4	6
External Outputs (EOs)	4	5	7
External Inquiries (EQs)	3	4	6
Internal Logical Files (ILFs)	7	10	15
External Interface Files (EIFs)	5	7	10

- ***Calculate FPA if all CAF and WAF are average for the following count values EI=10, EO=30, EQ=50, EIF=10, ILF=20***

Step1: Calculate UFP = $\sum_{i=1\ j=1}^{i=5,j=3} W_{ij} * Ci_j$

UFP=(10*4)+(30*5)+(50*4)+(10*7)+(20*10)

UFP=40+150+200+70+200

UFP=660

- **Step 2: Calculate** CAF=0.65+[0.01*$\sum F_i$]

CAF=0.65+[0.01*14*3]

CAF=1.07

Step 3: Calculate FPA=UFP*CAF

FPA=660*1.07

FPA=706.2

Questionnaires for software development

1. Does the system require reliable backup and recovery? (3)
2. Are specialized data communications required to transfer information to or from the application?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require online data entry?
7. Does the online data entry require the input transaction to be built over multiple screens or operations?
8. Are the ILFs updated online?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

Significance value

- **0- Not present/ No influence**
- **1- Incidental**
- **2- Moderate**
- **3- Average**
- **4- Significant**
- **5- Essential**

Web Engineering Project Metrics :

- **Measures that can be collected are**
- **Number of static Web pages** the end-user has no control over the content displayed on the page
- **Number of dynamic Web** pages end-user actions result in customized content displayed on the page
- **Number of internal page links** (internal page links are pointers that provide a hyperlink to some other Web page within the WebApp
- **Number of persistent data** objects As the number of persistent data objects grows the complexity of webapps also grows
- **Number of external systems interfaced** As the requirement for interfacing grows, system complexity and development effort also increases
- **Number of static content objects** They encompass graphical,audio,video information incorporated into the webapp
- **Number of dynamic content objects** Generated based on end-user actions
- **Number of executable functions** An executable function provides some computational service to end-user. As number of functions grows construction effort increases
- We can define a metric that reflects degree of end-user customization for webapp to effort expended on web-project

  Nsp=number of static pages

  Ndp=number of dynamic pages

  Customization index C=Ndp/(Ndp+Nsp)

Object-oriented Metrics

The set of metrics for oo projects are

Number of scenario scripts (i.e., use cases)

- This number is directly related to the size of an application and to the number of test cases required to test the system

Number of key classes (the highly independent components)

- Key classes are defined early in object-oriented analysis and are central to the problem domain
- This number indicates the amount of effort required to develop the software

Number of support classes

- This number indicates the amount of effort and potential reuse

Average number of support classes per key class

- Estimation of the number of support classes can be made from the number of key classes
- GUI applications have between two and three times more support classes as key classes
- Non-GUI applications have between one and two times more support classes as key classes

Number of subsystems

- A subsystem is an aggregation of classes that support a function that is visible to the end user of a system

3.Metrics for Software Quality Correctness

These are the measures of software quality

Correctness. A program must operate correctly or it provides little value to its users. Correctness is the degree to which the software performs its required function.

Maintainability. Software maintenance and support accounts for more effort than any other software engineering activity. Maintainability is the ease with which a program can be corrected if an error is encountered, adapted if its environment changes, or enhanced if the customer desires a change in requirements.

A simple time-oriented metric is **mean-time-to-change (MTTC),** the time it takes to analyze the change request

Integrity. Software integrity has become increasingly important in the age of cyber terrorists and hackers. This attribute measures a system's ability to withstand attacks (both accidental and intentional) to its security.

To measure integrity, two additional attributes must be defined: threat and security.

Threat is the probability (which can be estimated or derived from empirical evidence) that an attack of a specific type will occur within a given time. Security is the probability (which can be estimated or derived from empirical evidence) that the attack of a specific type will be repelled.

The integrity of a system can then be defined as:

$$\text{Integrity} = \Sigma\,[1 - (\text{threat} \times (1 - \text{security}))]$$

Defect Removal Efficiency

- Defect removal efficiency provides benefits at both the project and process level.
- It indicates the percentage of software errors found before software release
- It is defined as **DRE = E / (E + D)**
  - is the number of errors found before delivery of the software to the end user
  - is the number of defects found after delivery

4 .Integrating Metrics within the Software Process

To be an effective aid in process improvement and/or cost and effort estimation, baseline data must have the **following attributes:**

(1) Data must be reasonably accurate—guestimates about past projects are to be avoided,

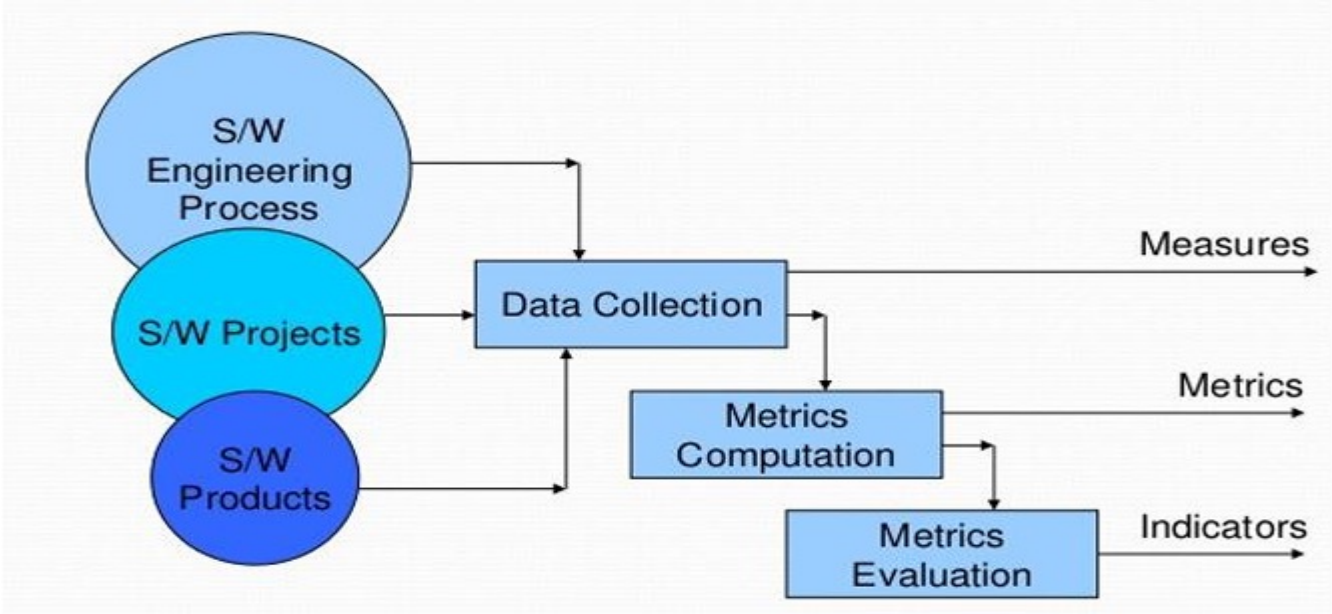
(2) Data should be collected for as many projects as possible

(3) Measures must be consistent (for example,a line of code must be interpreted consistently across all projects for which data are collected)

(4) Applications should be similar to work that is to be estimated—it makes little sense to use a baseline for batch information systems work to estimate a real-time, embedded application.

Metrics Collection, Computation, and Evaluation:

The process for establishing a metrics baseline is illustrated in below diagram. Ideally, data needed to establish a baseline have been collected in an ongoing manner.



EMPIRICAL ESTIMATION MODELS/Project Estimation- COCOMO Model

1. **Project Estimation** in software engineering?
2. **COCOMO Model** and its hierarchies.
3. **Basic COCOMO** Model and its **Modes** of development.
4. How to calculate **Effort**, **Development Time**, **Staff Size**, and **Productivity** of software development using the basic COCOMO Model in three different modes of development.

1. **Project Estimation** in software engineering?

Controlling

Tracking

Monitoring

Result gets done in time, with quality.

The estimation carries inherent risk and this risk leads to uncertainty.

- Project scope must be understood
- Elaboration or decomposition is necessary
- Historical metrics are very helpful
- At least two different techniques should be used

- Project scope must be understood
- Elaboration or decomposition is necessary
- Historical metrics are very helpful
- At least two different techniques should be used

- Past or similar project experience
- Task breakdown and effort estimates
- Empirical Models
- Automated Tools
- Function Point
- Lines of Code

COCOMO MODEL

CO — Constructive

CO — Cost

MO — Model

It is based on LOC (Lines of Code) where project estimation is done base on the total lines of codes required to develop the system. i.e. Size of the system define the cost of the project.

Barry W. Boehm in 1981

Used to estimate the **effort, cost, development time, average staff size, productivity,**

- BASIC COCOMO MODEL
- INTERMEDIATE COCOMO MODEL
- DETAILED COCOMO MODEL

EMPIRICAL ESTIMATION MODELS/Project Estimation-COCOMO Model

- An *estimation model* for computer software uses empirically derived formulas to predict effort as a function of LOC or FP.

The Structure of Estimation Models

A typical estimation model is derived using regression analysis on data collected from past software projects. The overall structure of such models takes the form .

$$E = A + B \times (e_v)^C$$

where A, B, and C are empirically derived constants,

E is effort in person-months,

e_v is the estimation variable (either LOC or FP).

Among the many LOC-oriented estimation models proposed in the literature are

$E = 5.2 \times (\text{KLOC})^{0.91}$	Walston-Felix model
$E = 5.5 + 0.73 \times (\text{KLOC})^{1.16}$	Bailey-Basili model
$E = 3.2 \times (\text{KLOC})^{1.05}$	Boehm simple model
$E = 5.288 \times (\text{KLOC})^{1.047}$	Doty model for KLOC > 9

FP-oriented models have also been proposed. These include

$E = -91.4 + 0.355\ \text{FP}$	Albrecht and Gaffney model
$E = -37 + 0.96\ \text{FP}$	Kemerer model
$E = -12.88 + 0.405\ \text{FP}$	Small project regression model

EMPIRICAL ESTIMATION MODELS

Constructive Cost Model

- **Used** to calculate:
  - Effort (manpower)
  - Development Time
  - Average Staff Size
  - Productivity
- **Types** of Models
  - Basic COCOMO
  - Intermediate COCOMO
  - Complete/Detailed COCOMO
- **Modes** to apply COCOMO
  - Organic
  - Semi-detached
  - Embedded

Type	Use
Basic	Small teams
Intermediate	Intermediate deadlines with innovations in between
Complete	Very large projects

Types of COCOMO Models and their applications

Mode/Details	Project Size	Project Nature	Innovation	Deadline
Organic	2-50 KLOC	Small sized and experienced developers	Little	Flexible (not tight)
Semi-detached	50-300 KLOC	Medium size project and team	Medium	Medium
Embedded	Over 300 KLOC	Large projects	Significant	Tight

Modes of COCOMO and their applications

Invontry mgt s/w
Lib.mgt
Payroll s/w

Database type of appln

ATM system
Flight Simulation

In COCOMO, projects are categorized into three types:

1.Organic

2.Semidetached

3.Embedded

1.Organic: A development project can be treated of the organic type, if the project deals with developing a well-understood application program, the size of the development team is reasonably small, and the **team members are experienced** in developing similar methods of projects. **Examples of this type of projects are simple business systems, simple inventory management systems, and data processing systems.**

2. Semidetached: A development project can be treated with semidetached type if the development consists of a **mixture of experienced and inexperienced staff**. Team members may have finite experience in related systems but may be unfamiliar with some aspects of the order being developed. **Example of Semidetached system includes developing a new operating system (OS), a Database Management System (DBMS), and complex inventory management system.**

3. Embedded: A development project is treated to be of an embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational method exist. **For Example:** ATM, Air Traffic control.

For three product categories, Bohem provides a different set of expression to predict effort (in a unit of person month)and development time from the size of estimation in KLOC(Kilo Line of code) efforts estimation takes into account the productivity loss due to holidays, weekly off, coffee breaks, etc.

According to Boehm, software cost estimation should be done through three stages:

1.Basic Model
2.Intermediate Model
3.Detailed Model

1. Basic COCOMO Model: The basic COCOMO model provide an accurate size of the project parameters. The following expressions give the basic COCOMO estimation model:

$$\textbf{Effort} = a_1 * (\text{KLOC})^{a_2}\ \textbf{PM}$$

$$\textbf{Tdev} = b_1 * (\text{efforts})^{b_2}\ \textbf{Months}$$

Where

KLOC is the estimated size of the software product indicate in Kilo Lines of Code,

a_1, a_2, b_1, b_2 are constants for each group of software products,

Tdev is the estimated time to develop the software, expressed in months,

Effort is the total effort required to develop the software product, expressed in **person months (PMs)**.

- $E = a_b (KLOC)^{b_b}$ *Person Month*
- $D = c_b (E)^{d_b}$ *Months*
- $P = \frac{E}{D}$ *Persons*
- $Prod = \frac{KLOC}{Effort}$ *KLOC/Person Month*

Mode/Coefficients	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Coefficients of Basic COCOMO Model

Eg: Suppose project estimated to be of 400 KLOC. Calculate Effort, Dev Time, Staff Size and Productivity for each of the modes in Basic COCOMO Model

Organic

$E = a_b (KLOC)^{b_b}$

$E=2.4\ (400)^{1.05}$

$E=1295.31$ Person Month

$D = c_b (E)^{d_b}$

$D=2.5(1295.31)^{0.38}$

D=38.07 Months

$P = \frac{E}{D}$

P=1295.31/38.07

P=34.02 ~ 34 Persons

Prod=KLOC/Effort

Prod=400/1295.31

Prod=0.308 KLOC/PM

Intermediate COCOMO

- Extension of the Basic COCOMO Model
- Has 15 Additional predictors (cost drivers)
- Predictors are used to adjust the nominal cost of project to actual project environment
- Each Cost Driver is rated in terms of ***VERY LOW, LOW, NORMAL, HIGH, VERY HIGH, EXTRA HIGH***

COST DRIVERS

Product Attributes

- Required s/w reliability (RELY)
- Database size (DATA)
- Product complexity (CPLX)

Computer Attributes

- Execution Time Constraint (TIME)
- Storage Constraints (STOR)
- Virtual Machine Volatility (VIRT)
- Turn Around Time (TURN)

Personal Attributes

- Analyst Capability (ACAP)
- Application Experience (AEXP)
- Programmer Capability (PCAP)
- Programming Language Experience (LEXP)
- Virtual Machine Experience (VEXP)

Project Attributes

- Modern Programming Practices (MODP)
- Use of S/W tools (TOOL)
- Required development Schedule (SCED)

Cost Drivers	Very Low	Low	Nominal	High	Very High	Extra High
RELY	0.75	0.88	1.00	1.15	1.40	-
DATA	-	0.94	1.00	1.08	1.16	-
CPLX	0.70	0.85	1.00	1.15	1.30	1.65
TIME	-	-	1.00	1.11	1.30	1.66
STOR	-	-	1.00	1.06	1.21	1.56
VIRT	-	0.87	1.00	1.15	1.30	-
TURN	-	0.87	1.00	1.07	1.15	-
ACAP	1.46	1.19	1.00	0.86	0.71	-
AEXP	1.29	1.13	1.00	0.91	0.83	-
PCAP	1.42	1.17	1.00	0.86	0.70	-
VEXP	1.21	1.10	1.00	0.90	-	-
LEXP	1.14	1.07	1.00	0.95	-	-

Cost Driver Effort Adjustment Factor

- $E = a_b (KLOC)^{b_b} * EAF$ *Person Month*

Where EAF is Effort Adjustment Factor

- $D = c_b (E)^{d_b}$ *Months*

Cost Drivers	Very Low	Low	Nominal	High	Very High	Extra High
MODP	1.24	1.10	1.00	0.91	0.82	-
TOOL	1.24	1.10	1.00	0.91	0.83	-
SCED	1.23	1.08	1.00	1.04	1.10	-

Cost Driver Effort Adjustment Factor contd.

Mode/Coefficients	a_b	b_b	c_b	d_b
Organic	3.2	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

Coefficients of Intermediate COCOMO Model

A project size of 200 KLOC is to be developed. Software development team has nominal experience on similar type of projects and the project schedule is on high priority. Calculate the Effort and development time of the project.

Soln:

- Semidetached mode works best here as estimated size is 200 KLOC
- Software Dev Team Avg Experience: LEXP is Nominal, hence 1.00
- Project Schedule is high priority: SCED is high, hence 1.04
- Effort Adjustment Factor Calculation

  EAF=1.00*1.04=1.04

- Effort Calculation

  $E = a_b\ (KLOC)^{b_b} * EAF\ _{Person\ Month}$

  $E=(3.0)(200)^{1.12} * 1.04$

  $E=1178.44\ _{Person\ Month}$

- Development Time Calculation

  $D = c_b\ (E)^{d_b}\ _{Months}$

  $D=(2.5)(1178.44)^{0.35}$

  $D=29.70\ _{Months}$

Detailed COCOMO

- Phase Sensitive
  - Planning And Requirement Gathering (PR)
  - System Design (SD)
  - Document Design (DD)
  - Code and Test (C)
  - Integration and Test (I)
- Cost Driver effect is calculated on each Phase to determine effort to complete each phase
- Ratings are given to drivers at that level where cost driver is affected the most

- Adjustment Factor (A) is calculated as reusability will affect the cost of development

  $A=0.4(DD)+0.3(C)+0.3(I)$
- The Cost of the software is calculated as:

  $Effort_d=\mu_p * E_i$

  $Development\ Time_d= \tau_p * D_i$

Given a project with the following main components : Screen Edit, CLI, File I/P and O/P, Cursor Movement, Screen Movement. The size for these are estimated to be 4KLOC, 2KLOC, 1KLOC, 2KLOC and 3KLOC. Using detailed COCOMO determine

1. Overall cost and schedule estimates
2. Cost and schedule estimates for different phases

Soln:

1. $E_i=a_i(KLOC)b_i$ * EAF
   - Total KLOC = 4+2+1+2+3=12 KLOC

     Organic Mode is useful is this size
   - EAF Cost Drivers:

     Reliability – High – 1.15

     Language Experience – Low – 1.07

     Product Complexity – High – 1.15

     Analyst Capability – High – 0.86

     Hence EAF=1.15*1.07*1.15*0.86=1.216

$E_i=a_i(KLOC)b_i$ * EAF

$E_i=(3.2)(12)^{(1.05)}$ * 1.216

E_i=52.9 Person Month

$D_i=c_i(E)d_i$

$D_i=(2.5)(52.9)^{(0.38)}$

D_i=11.29 Months

Mode and Code Size	Plan and Reqmts	System Design	Document Design	Code and Test	Integrate and Test
Life Cycle Phase Value of μ_p					
Organic Small (2)	0.06	0.16	0.26	0.42	0.16
Organic Medium (32)	0.06	0.16	0.24	0.38	0.22
Semi Detached Medium (32)	0.07	0.17	0.25	0.33	0.25
Semi Detached Large (128)	0.07	0.17	0.24	0.31	0.28
Embedded Large (128)	0.08	0.18	0.25	0.26	0.31
Embedded XL (320)	0.08	0.18	0.24	0.24	0.34

2. $Effort_d = \mu_p * E_i$

$Effort_d$ in Person Month in all phases individually

$=0.06*52.9=3.174$

$=0.16*52.9=8.464$

$=0.26*52.9=13.754$

$=0.42*52.9=22.218$

$=0.16*52.9=8.464$

Mode and Code Size	Plan and Reqmts	System Design	Document Design	Code and Test	Integrate and Test
Life Cycle Phase Value of τ_p					
Organic Small (2)	0.10	0.19	0.24	0.39	0.18
Organic Medium (32)	0.12	0.19	0.21	0.34	0.26
Semi Detached Medium (32)	0.20	0.26	0.21	0.27	0.26
Semi Detached Large (128)	0.22	0.27	0.19	0.25	0.29
Embedded Large (128)	0.36	0.36	0.18	0.18	0.28
Embedded XL (320)	0.40	0.38	0.16	0.16	0.30

2. Development $Time_d = \tau_p * D_i$

Development $Time_d$ in Month in all phases individually

=0.10*11.29=1.129
=0.19*11.29=2.145
=0.24*11.29=2.709
=0.39*11.29=4.403
=0.18*11.29=2.032

Problems

Suppose that a project was estimated to be 4 million LOC. Calculate effort & time for each of 3 modes of development.

SOLUTION

As we know that the 3 modes of development are

1. *Organic*
2. *Semi-detached*
3. *Embeded*

Also we know that

Effort $= a\ (KLOC)^{b}$ *Person-Month*

Development Time $= c\ (Effort)^{d}$ *Months*

Here, we are given LOC = 400000

Therefore, KLOC = 400

1. Organic

KLOC = 400 | a = 2.4 b = 1.05 c = 2.5 d = 0.38

Effort

$= a\ (KLOC)^{b}$ *Person-Month*

$= 2.4\ (400)^{1.05}$ *Person-Month*

≈ 1295 *Person-Month*

Development Time

$= c\ (Effort)^{d}$ *Months*

$= 2.5\ (1295)^{0.38}$ *Months*

≈ 38 *Months*

2. Semi-detacheed

KLOC = 400 | a = 3 b = 1.12 c = 2.5 d = 0.35

Effort

$= a\ (KLOC)^{b}$ *Person-Month*

$= 3\ (400)^{1.12}$ *Person-Month*

≈ 2462 *Person-Month*

Development Time

$= c\ (Effort)^{d}$ *Months*

$= 2.5\ (2462)^{0.35}$ *Months*

≈ 38.4 *Months*

3. Embeded

KLOC = 400 | a = 3.6 b = 1.2 c = 2.5 d = 0.32

Effort

$= a\ (KLOC)^{b}$ *Person-Month*

$= 3.6\ (400)^{1.2}$ *Person-Month*

≈ 4772 *Person-Month*

Development Time

$= c\ (Effort)^{d}$ *Months*

$= 2.5\ (4772)^{0.32}$ *Months*

≈ 38 *Months*

PROJECT SCHEDULING

Project Scheduling

- Project scheduling is concerned with the techniques that can be employed to manage the activities that need to be undertaken during the development of a project.

Basic Principles:

Like all other areas of software engineering, a number of basic principles guide software project scheduling:

Compartmentalization. The project must be compartmentalized into a number of manageable activities and tasks. To accomplish compartmentalization, both theproduct and the process are refined.

Interdependency. The interdependency of each compartmentalized activity ortask must be determined. Some tasks must occur in sequence, while others can occur in parallel. Some activities cannot commence until the work product produced by another is available. Other activities can occur independently.

Time allocation. Each task to be scheduled must be allocated some number of work units (e.g., person-days of effort). In addition, each task must be assigned a start date and a completion date that are a function of the interdependencies and whether work will be conducted on a full-time or part-time basis.

Effort validation. Every project has a defined number of people on the software team. As time allocation occurs, you must ensure that no more than the allocated number of people has been scheduled at any given time.

Defined responsibilities. Every task that is scheduled should be assigned to a specific team member.

Defined outcomes. Every task that is scheduled should have a defined outcome.

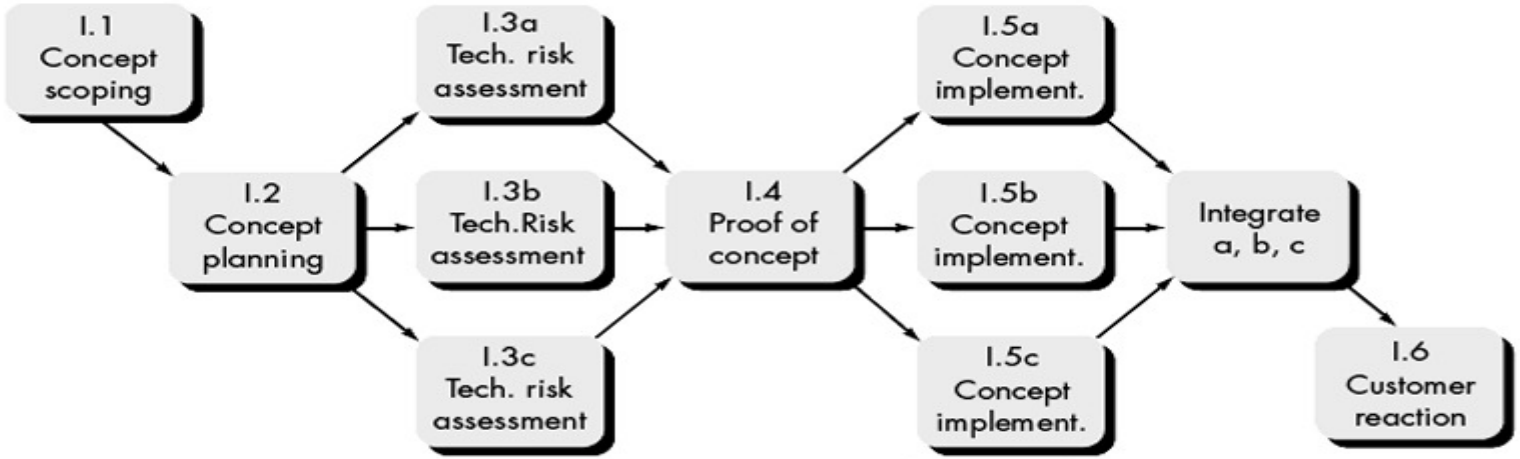
For software projects, the outcome is normally a work product (e.g., the design of a component) or a part of a work product. Work products are often combined in deliverables.

Defined milestones. Every task or group of tasks should be associated with a project milestone.

- **To schedule the project plan, a software project manager wants to do the following:**

1. Identify all the functions required to complete the project.
2. Break down large functions into small activities.
3. Determine the dependency among various activities.
4. Establish the most likely size for the time duration required to complete the activities.
5. Allocate resources to activities.
6. Plan the beginning and ending dates for different activities.
7. Determine the critical path. A critical way is the group of activities that decide the duration of the project.

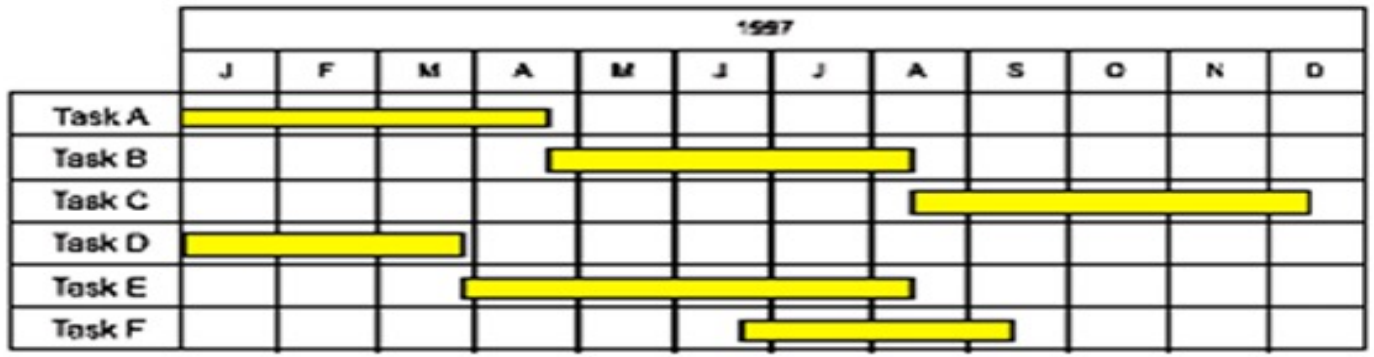
DEFINING A TASK NETWORK:
A *task network,* also called an *activity network,* is a graphic representation of the task flow for a project.



Scheduling of a software project does not differ greatly from scheduling of any multitask engineering effort.
Program evaluation and review technique (PERT) and the **critical path method** (CPM) are two project scheduling methods that can be applied to software development.
Interdependencies among tasks may be defined using a task network. Tasks,sometimes called the project **work breakdown structure** (WBS), are defined for the product as a whole or for individual functions.
Both PERT and CPM provide quantitative tools that allow you to
(1) Determine the critical path—the chain of tasks that determines the duration of the project
(2) Establish "most likely" time estimates for individual tasks by applying statistical models
(3) Calculate "boundary times" that define a time "window" for a particular task

Time-Line Charts (or) Gantt charts

The use of Gantt charts started during the industrial rev-olution of the late 1800's. An early industrial engineer named Henry Gantt developed these charts to improve factory efficiency. Gantt charts are developed using bars to represent each task. The length of the bar shows how long the task is expected to take to complete. Duration is easily shown on Gantt charts.



Once the information necessary for the generation of a time-line chart has been input, the majority of software project scheduling tools produce *project tables*—a tabular listing of all project tasks, their planned and actual start and end dates, and a variety of related information

Work tasks	Planned start	Actual start	Planned complete	Actual complete	Assigned person	Effort allocated	Notes
I.1.1 Identify needs and benefits							Scoping will require more effort/time
Meet with customers	wk1, d1	wk1, d1	wk1, d2	wk1, d2	BLS	2 pd	
Identify needs and project constraints	wk1, d2	wk1, d2	wk1, d2	wk1, d2	JPP	1 pd	
Establish product statement	wk1, d3	wk1, d3	wk1, d3	wk1, d3	BLS/JPP	1 pd	
Milestone: Product statement defined	wk1, d3	wk1, d3	wk1, d3	wk1, d3			
I.1.2 Define desired output/control/input (OCI)							
Scope keyboard functions	wk1, d4	wk1, d4	wk2, d2		BLS	1.5 pd	
Scope voice input functions	wk1, d3	wk1, d3	wk2, d2		JPP	2 pd	
Scope modes of interaction	wk2, d1		wk2, d3		MLL	1 pd	
Scope document diagnostics	wk2, d1		wk2, d2		BLS	1.5 pd	
Scope other WP functions	wk1, d4	wk1, d4	wk2, d3		JPP	2 pd	
Document OCI	wk2, d1		wk2, d3		MLL	3 pd	
FTR: Review OCI with customer	wk2, d3		wk2, d3		all	3 pd	
Revise OCI as required	wk2, d4		wk2, d4		all	3 pd	
Milestone: OCI defined	wk2, d5		wk2, d5				
I.1.3 Define the function/behavior							

- Project management can be defined as a structural way of **planning, scheduling, executing, monitoring and controlling** various phases of a project.
- To achieve the end goal of a project on time, **PERT and CPM are two project management techniques** that every management should implement. These techniques help in displaying the progress and series of actions and events of a project.

Abbreviation	
PERT – Project Evaluation and Review Technique	CPM – Critical Path Method
What does It Mean?	
PERT – PERT is a popular project management technique that is applicable when the time required to finish a project is not certain	CPM – CPM is a statistical algorithm which has a certain start and end time for a project
Model Type	
PERT – PERT is a probabilistic model	CPM – CPM is a deterministic model
Focus	
PERT – The main focus of PERT is to minimise the time required for completion of the project	CPM – The main focus of CPM is on a trade-off between cost and time, with a major emphasis on cost-cutting.

CPM models the activities and events of a project as a network. Activities are shown as nodes on the network and events that signify the beginning or ending of activities are shown as arcs or lines between the nodes

Steps in CPM Project Planning

1.Specify the individual activities.

2.Determine the sequence of those activities.

3.Draw a network diagram.

4.Estimate the completion time for each activity.

5.Identify the critical path (longest path through the network)

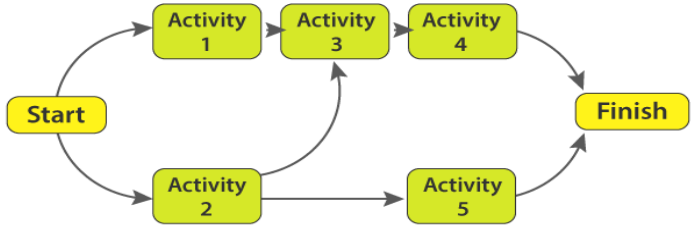
6.Update the CPM diagram as the project progresses

Advantages of CPM

•Provides an outline for long term coordination and planning of a project
•Recognizes critical activities
•Easy to plan, schedule and control project
•It improves productivity
•Manages the resource needed

Disadvantages of CPM

•For beginners its difficult to understand
•Software too expensive
•Sometimes, to structure CPM is too time-consuming
•It cannot control and form the schedule of a person involved in the project
•Allocation of resources cannot be monitored properly

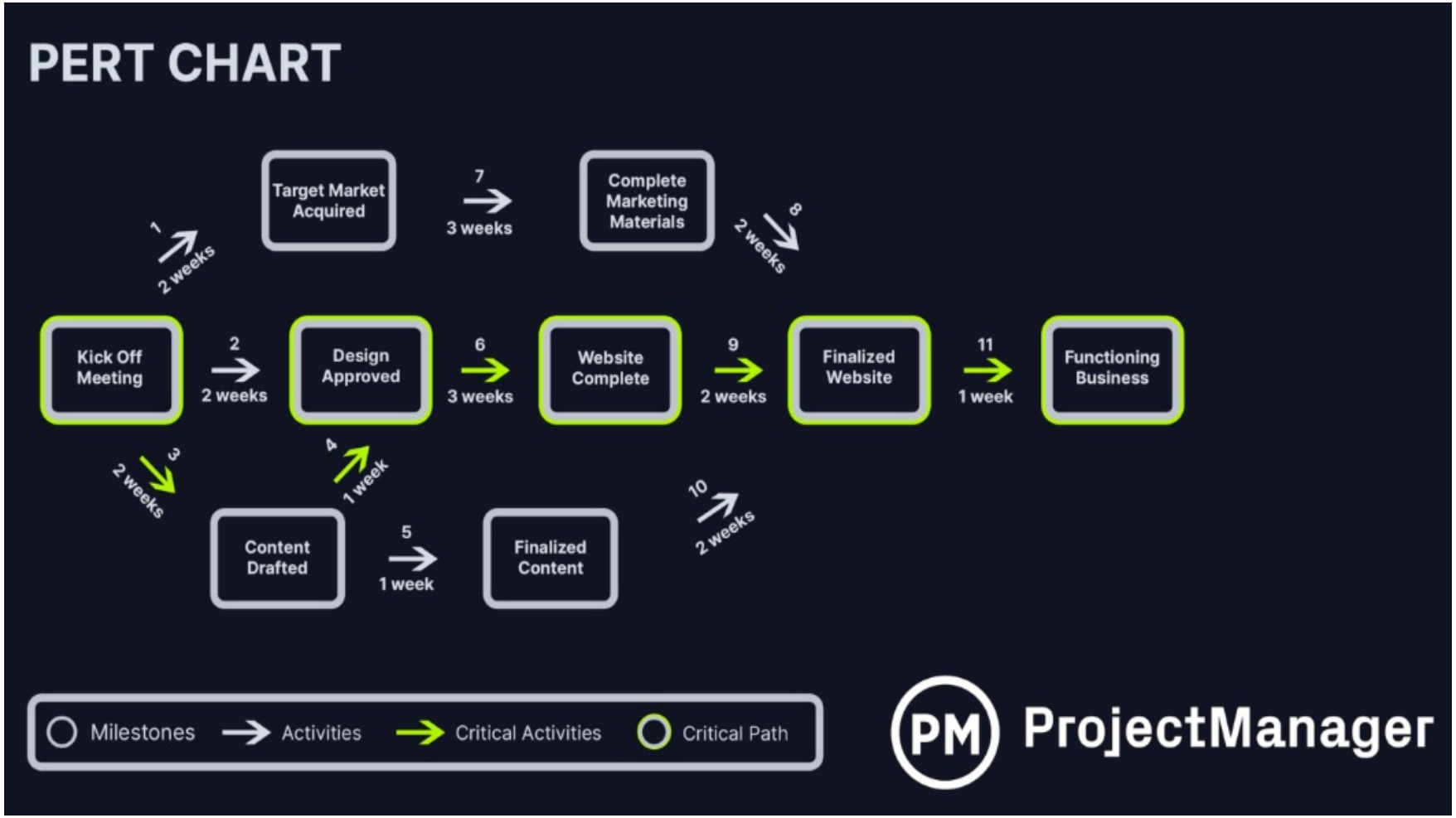


CPM in Project Management

- The Critical Path Method in project management is a step-by-step technique used in the **planning process that explains the critical and non-critical activities of a project**.
- CPM goals are to check **time-bound issues and process that causes blockage in the project.**
- The CPM is preferably applicable to projects that involve various activities that are associated with a complex method. Once CPM is applied, it will help you keep your projects on track.

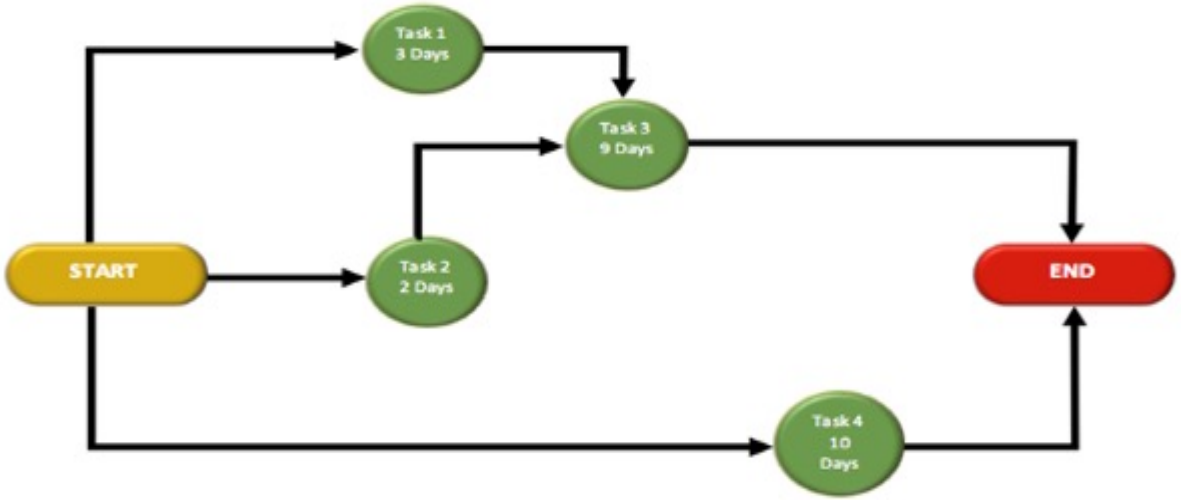
•Helps you **recognize the action that needs to be performed on time** so that the whole project is completed on time.
•Indicates which responsibilities can be delayed and for how long without affecting the overall project plan.
•Determines the least amount of time it will take to accomplish the project.
•Tells you the newest and latest time each activity can start on in order to manage the schedule.
The term of each action is listed above each joint in the diagram. For an individual path, insert the duration of each node to ascertain the total duration. The critical path is the one that has the longest duration.

- Begin building the PERT chart starting on the left and moving toward the right. The farthest left is the beginning and the farthest right is the end of the project. The arrows connect the nodes or milestones and indicate the time spend on tasks.



PERT

The Program Evaluation and Review Technique.(PERT) is a network model that allows for randomness in activity completion times.PERT is typically represented as an activity on arc network, in which the activities are rep-resented on the lines and milestones on the nodes.



Steps in the PERT Planning Process

PERT planning involves the following steps:

1.Identify the specific activities and milestones.

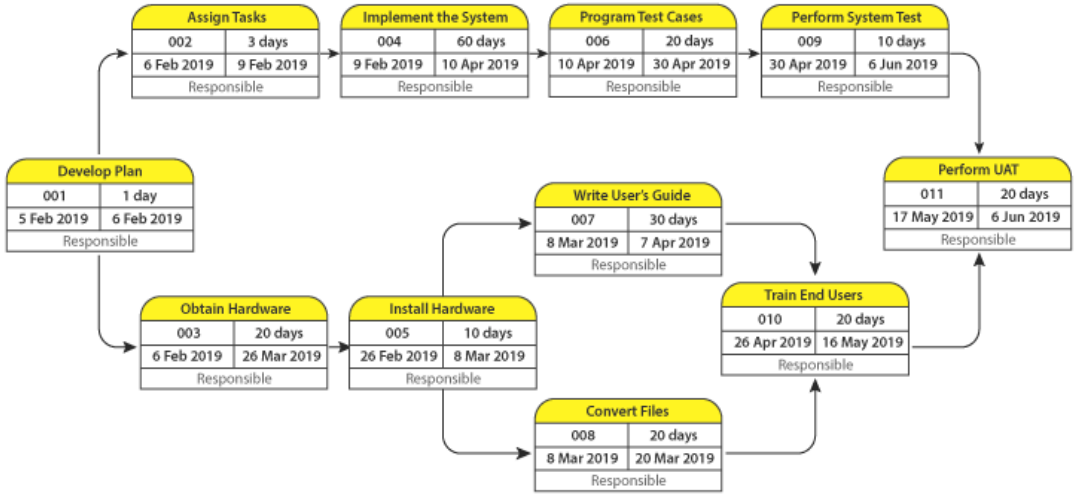
2.Determine the proper sequence of the activities.

3.Construct a network diagram.

4.Estimate the time required for each activity.

5.Determine the critical path.6.Update the PERT chart as the project progresses.

- Program (Project) Evaluation and Review Technique (PERT) is an activity to understand **the planning, arranging, scheduling, coordinating and governing of a project.**
- This program helps to understand the **technique of a study taken to complete a project, identify the least and minimum time taken to complete the whole project.** PERT was developed in the 1950s, with the aim of the cost and time of a project.



Risk Management

Contents

1. Source of Risk
2. Risk Management Process
3. Risk Identification
4. Risk Analysis & Projection or Prioritization
5. Risk Control Process

Source of Risk

➢**What is Risk?**

✓"Risk is an uncertain future event with a probability of occurrence and potential for loss"

➢**Source of Risk:**

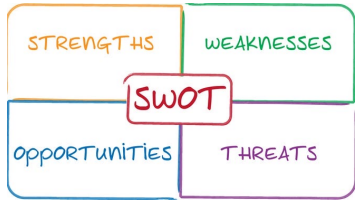
1. Misunderstanding of customer requirements.
2. Uncontrolled & continuous changing of customer requirements.
3. Unrealistic promises given to the customers.
4. Misunderstanding of real impact of new methodologies.
5. Miscalculation of robustness & extensibility of software design.
6. Miscalculation of Team work & group effectiveness.
7. Wrong budget estimation.

Risk Management

- Risk Management is an important part of project planning activities.
- It involves identifying and estimating the probability of risks with their order of impact on the project.

Risk Identification

- The project organizer needs to find out risk in the project as early as possible.
- So, the impact of risk can be reduced by making effective risk management planning.
- By doing Brainstorming, SWOT Analysis, Casual Mapping & Flowchart methods are used.



➢**There are different types of risks which can affect a software project:**

1. **Technology risks:** Software or Hardware technologies that are used to develop the system.
2. **People risks:** Risks that are connected with the person in the development team.
3. **Organizational risks:** Organizational environment where the software is being developed.
4. **Tools risks**: Software tools and other support software used to create the system.
5. **Requirement risks:** Changes to customer requirement & process of managing the requirements change.
6. **Estimation risks:** Management of estimation resources required to build the system

Risk Analysis & Projection or Prioritization

In Risk Analysis Process:

1. Identifying the problems causing risk in projects.
2. Identifying the probability of occurrence of problem.
3. Identifying the impact of problem.

➢**The probability of a risk can be categorized as:**

1. **Very Low (0-10%) :** Tolerable Risk (No harm)
2. **Low (10-25%) :** Low Risk (Minor effect)
3. **Moderate (25-50%) :** Medium Risk (Impact on Time)
4. **High (50-75%) :** High Risk (Impact on Time & Budget)
5. **Very high (+75%) : Intolerable Risk** (Impact on Output, Time, Budget & Performance)

Risk Control

- It is the process of managing risks to achieve desired outcomes.

1. Risk Planning:

- The risk planning technique considers all of the significant risks that have been identified and develop strategies to mitigate them.

➢**There are three main methods to plan for risk management:**

1. **Avoid the risk:** Discussing with client to change requirements, Decrease scope of work, Giving incentives to engineers to avoid the risk of human resources turnover etc.
2. **Transfer the risk:** The risky element developed by third party, Buying insurance cover etc.
3. **Risk reduction:** This means planning method to include the loss due to risk. If there is a risk that some key personnel might leave, new recruitment can be planned.

Risk Control

2. Risk Monitoring:

- This is an ongoing process throughout the project and requires continuous evaluation and assessment of potential risks.
- Any changes in the assumptions made about risks should be identified and appropriate actions are taken to manage those risks.

3. Risk Resolution:

- This process ensures that the project stays on track and risks are controlled within acceptable levels.
- The effectiveness of risk resolution depends on the accuracy of risk identification, analysis, and planning of risk solving .
- It has ability to respond promptly and effectively to any issues that arise during the project.