UNIT – I

Introduction to Project Planning: Introduction to Project Management:
Importance of Software Project Management. Categorization of Software Projects, Setting objectives.
Project Life Cycle and Effort Estimation: Software Process Models, Rapid Application development, Agile methods, Extreme Programming, Reliability Growth Models, Load Testing Model,Managing, Basics of Software estimation, Effort and Cost estimation techniques, COSMIC Full function points, COCOMO

*****

Write about the nature of Software.

1. Software is intangible:

- Software is largely intangible that cannot feel the shape of a piece of software and its design can be hard to visualize.
- It is difficult for people to assess its quality.

2. Software is easy to reproduce:

- The mass production of duplicate pieces of software is trivial.
- The process following completion of design is expensive.
- Therefore, all the cost of software in its development not in manufacturing.

3. The industry is labor -intensive:

- The software industry is labor intensive.
- Therefore, it is able to produce increasing the amount of products with less labor. However, it would require “intelligent machines”.

4. Untrained people can hack something together:

- Quality problems are hard to notice.
- A novice programmer can create a complex system that performs some useful function but it is highly disorganized in terms of its design.

5. Software is easy to modify:

- Software is physically easy to modify.Due to its complexity it is difficult to make changes. As a side of their modifications new bugs appear.

6. Software does not 'wear out':

- Software does not wear out like other engineering artifacts, but instead its design deteriorates as it is changed repeatedly.
- In ways that were not anticipated, thus making it complex.

Conclusion:

- Demand for software is high and rising.
- We are in perpetual 'software crisis'.
- We have to learn to 'engineer' software.
- Much software has poor design and is getting worse.

Explain about the Types of Software

There are many different types of software.

- Custom
- Generic
- Embedded
- Real time software
- Data processing software

1. Custom

- Custom software is developed to meet the specific needs of a particular customer.
- custom software is developed in-house within the same organization.
- Custom software is typically used by only a few people and its success depends on meeting their needs.
- Examples: air-traffic control systems and software for managing the specialized finances of large organizations.

2. Generic

- Generic software is designed to be sold on the open market.
- It is used by many people to run on general purpose computers.
- It is cheaper and more reliable
- Generic software is often called Commercial Off-The-Shelf software (COTS) and shrink-wrapped.
- Examples: spreadsheets,web browsers, operating systems, computer games etc.

3. Embedded

- Embedded software runs specific hardware devices.
- It is hard replace or upgrade.
- Examples: washing machines, DVD players,microwave ovens and automobiles.

4. Real time software

- Must react immediately
- Safety often a concern
- E.g. control and monitoring systems

5. Data processing software

- It is used to run businesses
- Accuracy and security of data are key
- The biggest design issues are how to organize the data and provide useful information to the users
- It performs functions such as recording sales, managing accounts, printing bills etc.

ructions.

1) Human-Intensive Process:-

Software development is primarily a human-intensive process. Even with tools for automation, coding, design, testing, and maintenance require human effort, creativity, and problem-solving skills.

2) Interaction with Other Software:-

Software often interacts with other systems or software. This interoperability and integration with other platforms or services distinguish it from many standalone physical products.

Describe how stakeholders play a role in software Engineering?

Ans: Many people are involved in a software engineering project and expect to benefit from its success. We will classify these stakeholders into four major categories, or roles, each having different motivations, and seeing the software engineering process somewhat differently like

1. **Users:**

   - ➢ These individuals are the end-users of the software.
   - ➢ Their goals include doing enjoyable or interesting work and gaining recognition for their efforts.
   - ➢ Users appreciate software that is easy to learn, improves their productivity, and makes their lives easier.

2. **Customers (or Clients):**

   - ➢ Customers are decision-makers who order and pay for the software.
   - ➢ They may or may not be the same as the end-users.
   - ➢ Their primary goal is either to increase profits or run their business more

effectively.

- ➢ They value software that helps their organization save money or improve overall productivity.

3. **Software Developers:**

- ➢ These are the people who create and maintain the software.

Within the development team, there are specialized roles (e.g., requirements specialists, database specialists, technical writers).

- ➢ Developers desire rewarding careers and may be motivated by solving challenging problems or gaining recognition for high-quality work.

4. **Development Managers:**

- ➢ These managers oversee the software development process.
- ➢ Their goal is to please the customer or maximize software sales while minimizing costs.
- ➢ They need knowledge of project management but may not be as familiar with technical details.
- ➢ Communication between developers and managers is crucial.

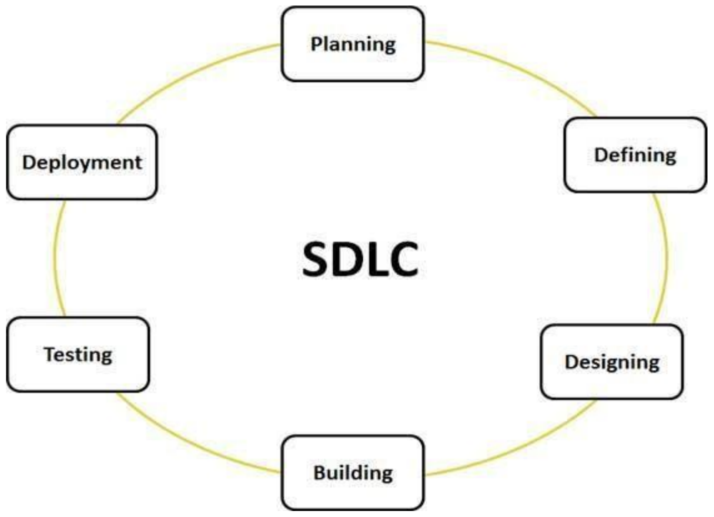
Note: sometimes one person plays different stakeholder roles simultaneously. For instance, if you're privately developing software for personal use, you'd embody all four roles.

Give a Comprehensive Overview of SDLC with a Neat Diagram

The Software Development Life Cycle (SDLC) is a framework that outlines the stages involved in planning, creating, testing, and delivering software applications. It provides a structured approach to software development, ensuring that the final product meets the required quality, timeline, and budget constraints.

The SDLC Phases:

Here are the typical phases of the SDLC, illustrated in the diagram below:



SDLC Phase Descriptions:

Here's a brief overview of each phase:

- **Planning:** Define project scope, goals, timelines, and resources.
- **Requirements Gathering:** Collect and document user requirements and expectations.
- **Analysis:** Break down requirements into smaller, manageable components.
- **Design:** Create a detailed design of the software architecture and components.
- **Implementation (Coding):** Write the code for the software application.
- **Testing:** Verify that the software meets the required quality and functionality standards.
- **Deployment:** Release the software to the production environment.
- **Maintenance:** Provide ongoing support, updates, and bug fixes to ensure the software remains relevant and functional.S

Key Takeaways:

- SDLC provides a structured approach to software development, ensuring quality, timeliness, and budget adherence.
- Each phase builds upon the previous one, ensuring a comprehensive and thorough software development process.
- The SDLC framework is flexible and can be tailored to specific project needs and methodologies (e.g., Agile, Waterfall).

Explain about Waterfall Model and state the advantages and disadvantages of it.

The waterfall model is a linear sequential development model that is often used in software engineering. It consists of a series of phases, where each phase must be completed before the next one can begin.

The phases of the waterfall model are:

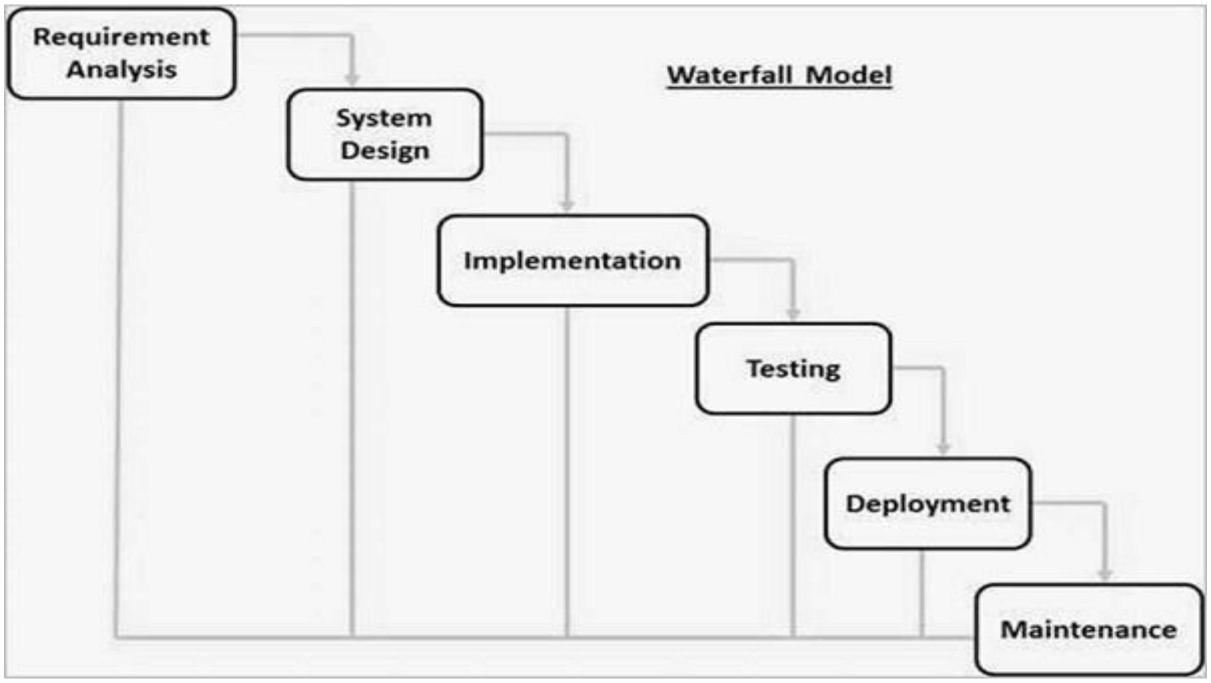
1) Requirements gathering: This phase involves defining the scope of the project, identifying the users' needs, and creating a detailed requirements document.**System design:** This phase involves designing the architecture of the software system, including the hardware and software components.

2) Implementation: This phase involves writing the code for the software system.

3) Testing: This phase involves testing the software system to ensure that it meets the requirements.

4) Integration: This phase involves combining the different components of the software system into a single product.

5) Maintenance: This phase involves maintaining the software system after it has been deployed.



Advantages of the waterfall model:

- Simple and easy to understand
- Well-defined phases
- Easy to manage
- Suitable for small projects with well-defined requirements

Disadvantages of the waterfall model:

- Rigid and inflexible
- Difficult to make changes during development
- High risk of failure
- Not suitable for large or complex projects

Why the waterfall model is not suitable for all projects?

- The Waterfall model is a sequential approach where you move from one phase to another in a linear way, like a waterfall flowing down.
- It's not suitable for all projects because it doesn't allow for much flexibility. If something changes in one phase, it can be tricky to go back and make adjustments.
- Some projects need more adaptability and room for changes as they progress, which is why other methodologies like Agile are used for those projects.
- Agile allows for more flexibility and iterative development.
- So, it really depends on the project's requirements and how much flexibility is needed throughout the process.

State the Pros and Cons of Waterfall Model.

PROS	CONS
Simple and easy to understand and use.	No working software is produced until late during the life cycle.

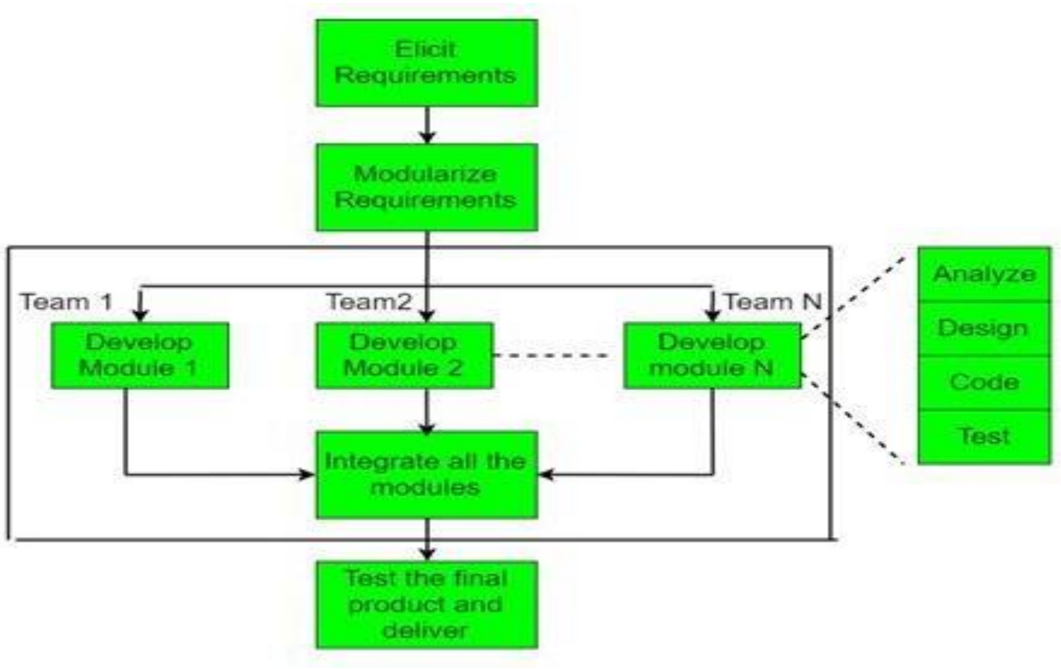
Easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.	High amounts of risk and uncertainty.
Phases are processed and completed one at a time.	Not a good model for complex and object-oriented projects.
Works well for smaller projects where requirements are very well understood.	Poor model for the projects where requirements are at a moderate to high risk of changing. So risk and uncertainty is high with this process model.
Clearly defined stages.	It is difficult to measure progress within stages.
Well understood milestones	Cannot accommodate changing requirements.
Easy to arrange tasks.	No working software is produced until late in the life cycle.
Process and results are well documented.	Adjusting scope during the life cycle can end a project.
	Integration is done as "big-bang" at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

Explain about Rapid Application Development Model (RAD)

- The RAD model or Rapid Application Development model is a type of software development methodology that emphasizes quick and iterative release cycles, primarily focusing on delivering working software in shorter timelines.
- Unlike traditional models such as the Waterfall model, RAD is designed to be more flexible and responsive to user feedback and changing requirements

throughout the development process.

This model consists of 4 basic phases:



1. **Requirements Planning –** This involves the use of various techniques used in requirements elicitation like brainstorming, task analysis, form analysis, user scenarios, FAST (Facilitated Application Development Technique), etc.

It also consists of the entire structured plan describing the critical data, methods to obtain it, and then processing it to form a final refined model.

2. **User Description –** This phase consists of taking user feedback and building the prototype using developer tools. In other words, it includes re-examination and validation of the data collected in the first phase. The dataset attributes are also identified and elucidated in this phase.
3. **Construction –** In this phase, refinement of the prototype and delivery takes place. It includes the actual use of powerful automated tools to transform processes and data models into the final working product. All the required modifications and enhancements are to be done in this phase.
4. **Cutover –** All the interfaces between the independent modules developed by separate teams have to be tested properly. The use of powerfully automated tools and subparts makes testing easier. This is followed by acceptance testing by the user.

When do you prefer the RAD Model?

1. **Well-understood Requirements:** When project requirements are stable and transparent, RAD is appropriate.
2. **Time-sensitive Projects:** Suitable for projects that need to be developed and delivered quickly due to tight deadlines.
3. **Small to Medium-Sized Projects:** Better suited for smaller initiatives requiring a controllable number of team members.
4. **High User Involvement:** Fits where ongoing input and interaction from users are essential.
5. **Innovation and Creativity**: Helpful for tasks requiring creative inquiry and innovation.
6. **Prototyping:** It is necessary when developing and improving prototypes is a key component of the development process.
7. **Low technological Complexity:** Suitable for tasks using comparatively straightforward technological specifications.

State various Advantages and disadvantages of Rapid Application Development Model (RAD)

Advantages:

- The use of reusable components helps to reduce the cycle time of the project. Feedback from the customer is available at the initial stages.
- Reduced costs as fewer developers are required.
- The use of powerful development tools results in better quality products in comparatively shorter periods.
- The progress and development of the project can be measured through the various stages.
- It is easier to accommodate changing requirements due to the short iteration time spans.
- Productivity may be quickly boosted with a lower number of employees.

Disadvantages:

- The use of powerful and efficient tools requires highly skilled professionals.
- The absence of reusable components can lead to the failure of the project.
- The team leader must work closely with the developers and customers to close the project on time.
- The systems which cannot be modularized suitably cannot use this model.
- Customer involvement is required throughout the life cycle.
- It is not meant for small-scale projects as in such cases, the cost of using automated tools and techniques may exceed the entire budget of the project.

State the Pros and Cons of RAD Model.

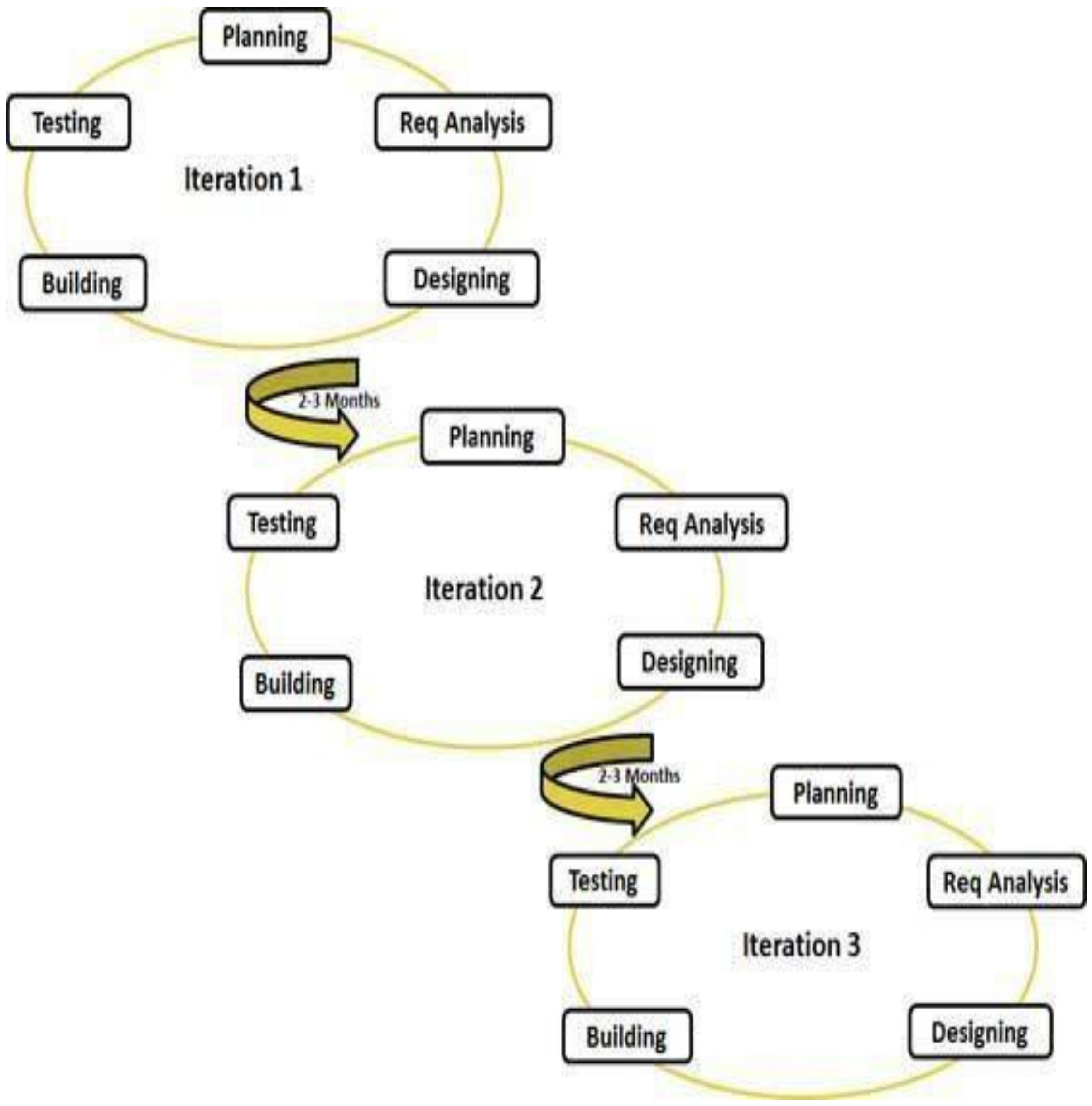
PROS	CONS
Changing requirements can be accommodated.	Reduced development time.
Progress can be measured.	Increases reusability of components.
Iteration time can be short with use of powerful RAD tools.	Dependency on technically strong team members for identifying business requirements.
Productivity with fewer people in short time.	Only system that can be modulated can be built using RAD.
Quick initial reviews occur.	Requires highly skilled developers/designer
Encourages customer feedback.	High dependency on modeling skills.
Integration from very beginning solves a lot of integration issues.	Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.
	Management complexity is more.
	Suitable for systems that are component based and scalable.
	Requires user involvement throughout the life cycle.
	Suitable for project requiring shorter development times.

Explain about Agile Software Development Method.

Agile Software Development Method:

Agile software development is an iterative and incremental approach to developing software. It breaks down the product into small, manageable builds, which are developed in iterations. Each iteration typically lasts from one to three weeks and involves cross-functional teams working simultaneously on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing. At the end of each iteration, a working product is displayed to the customer and important stakeholders

Here is a graphical illustration of the Agile Model:



The Agile model is widely used due to its flexibility and adaptability. Here are some reasons:

- Flexibility to Change: Agile development is focused on quick responses to change and continuous development. This allows teams to adapt to changing requirements and priorities, which is common in software development projects.
- Customer Satisfaction: Agile development involves continuous customer interaction, which ensures that the product meets the customer's requirements. This leads to higher customer satisfaction and reduced chances of project failure.
- Early Detection of Defects: Agile development involves iterative testing and feedback, which helps in early detection of defects and reduces the overall cost of defect fixing.
- Improved Team Collaboration: Agile development promotes cross-functional teams working together, which improves collaboration, communication, and motivation among team members.
- Faster Time-to-Market: Agile development allows for faster development and deployment of software, which enables companies to quickly respond to changing market conditions and stay ahead of the competition**.**

Overall, the agile model is widely used because it provides a flexible and adaptive approach to software development, which helps teams deliver high-quality software products quickly and efficiently.

Agile Frameworks in OOSE: (**This is just for idea not for the examination purpose**)

- Scrum: In a Scrum environment, OOSE principles are applied within each sprint. Scrum teams focus on delivering increments of functionality that align with user stories and requirements, and these functionalities are often represented by different objects or sets of interacting objects.
- Extreme Programming (XP): XP strongly emphasizes best practices such as Test Driven Development, refactoring, and pair programming, which complement the object-oriented design in OOSE. XP encourages developers to refine the object model continuously as new requirements emerge.
- Kanban: In Kanban, work items flow continuously, and OOSE allows developers to introduce and modify objects and their interactions fluidly. This helps in improving system modularity as tasks and features evolve.

State the Pros and Cons of Agile Model.

PROS	**CONS**
It is very realistic approach to software development.	Not suitable for handling complex dependencies.
Promotes teamwork and cross training.	More risk of sustainability, maintainability and extensibility.
Functionality can be developed rapidly and demonstrated.	An overall plan, an agile leader and agile Product Manager practice is a must without which it will not work.
Resource requirements are very less.	Strict delivery management dictates the scope, functionality to be delivered and

	adjustments to meet the deadlines.
Suitable for fixed or changing requirements.	Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
Delivers early partial working solutions.	There is very high individual dependency, since there is minimum documentation generated.
Good model for environments	Transfer of technology to new team members may be quite challenging due to lack of documentation.
Easy to manage and gives flexibility to developers.	

What is Object Orientation? Discuss about Procedural and Data abstraction.

Object-oriented systems is the combination of procedural abstraction with data abstraction. Object-oriented systems make use of *abstraction* in order to help make software less complex.

Procedural abstraction and the procedural paradigm

In programming, software has been organized around the notion of procedures and the programmer only needs to know how to call it and what it computes. The programmer's view of the system is thus made simpler.

In the procedural paradigm, the entire system is organized into a set of procedures. One 'main' procedure calls several other procedures, which in turn call others.

The procedural paradigm works very well when the main purpose of programs is to perform calculations with relatively simple data. However, as computers and applications have become more complex, so data abstraction has been introduced. Systems written using the procedural paradigm are complex if each procedure works with many types of data, or if each type of data has many different procedures that access and modify it.

Data abstraction

Data abstractions can help reduce some of a system's complexity. Records and structures were the first data abstractions to be introduced. The idea is to group together the pieces of data that describe some entity, so that programmers can manipulate that data as a unit.

However, even when using data abstraction, programmers still have to write complex code in many different places due to various constrains and conditions as shown below.

```
if account is of type checking then
        do something
```

```
    else if account is of type savings then
            do something else
    else

do yet anotherthing
    endif
```

Need for Software Project Management

Software is a non-physical product. Software development is a new stream in business and there is very little experience in building software products. Most of the software products are made to fit clients' requirements. The most important is that basic technology changes and advances so frequently and rapidly that the experience of one product may not be applied to the other one.

Such types of business and environmental constraints increase risk in software development hence it is essential to manage software projects efficiently. It is necessary for an organization to deliver quality products, keep the cost within the client's budget constraint, and deliver the project as per schedule. Hence, in order, software project management is necessary to incorporate user requirements along with budget and time constraints.

Types of Management in SPM

1. Conflict Management

Conflict management is the process to restrict the negative features of conflict while increasing the positive features of conflict. The goal of conflict management is to improve learning and group results including efficacy or performance in an organizational setting. Properly managed conflict can enhance group results.

2. Risk Management

Risk management is the analysis and identification of risks that is followed by synchronized and economical implementation of resources to minimize, operate and control the possibility or effect of unfortunate events or to maximize the realization of opportunities.

3. Requirement Management

It is the process of analyzing, prioritizing, tracking, and documenting requirements and then supervising change and communicating to pertinent stakeholders. It is a continuous process during a project.

4. Change Management

Change management is a systematic approach to dealing with the transition or transformation of an organization’s goals, processes, or technologies. The purpose of change management is to execute strategies for effecting change, controlling change, and helping people to adapt to change.

5. Software Configuration Management

Software configuration management is the process of controlling and tracking changes in the software, part of the larger cross-disciplinary field of configuration management. Software configuration management includes revision control and the inauguration of baselines.

6. Release Management

Release Management is the task of planning, controlling, and scheduling the built-in deploying releases. Release management ensures that the organization delivers new and enhanced services required by the customer while protecting the integrity of existing service.

Aspects of Software Project Management

The list of focus areas it can tackle and the broad upsides of Software Project Management is:

1. Planning

The software project manager lays out the complete project's blueprint. The project plan will outline the scope, resources, timelines, techniques, strategy, communication, testing, and maintenance steps. SPM can aid greatly here.

2. Leading

A software project manager brings together and leads a team of engineers, strategists, programmers, designers, and data scientists. Leading a team necessitates exceptional communication, interpersonal, and leadership abilities. One can only hope to do this effectively if one sticks with the core SPM principles.

3. Execution

SPM comes to the rescue here also as the person in charge of software projects (if well versed with SPM/Agile methodologies) will ensure that each stage of the project is completed successfully. measuring progress, monitoring to check how teams function, and generating status reports are all part of this process.

4. Time Management

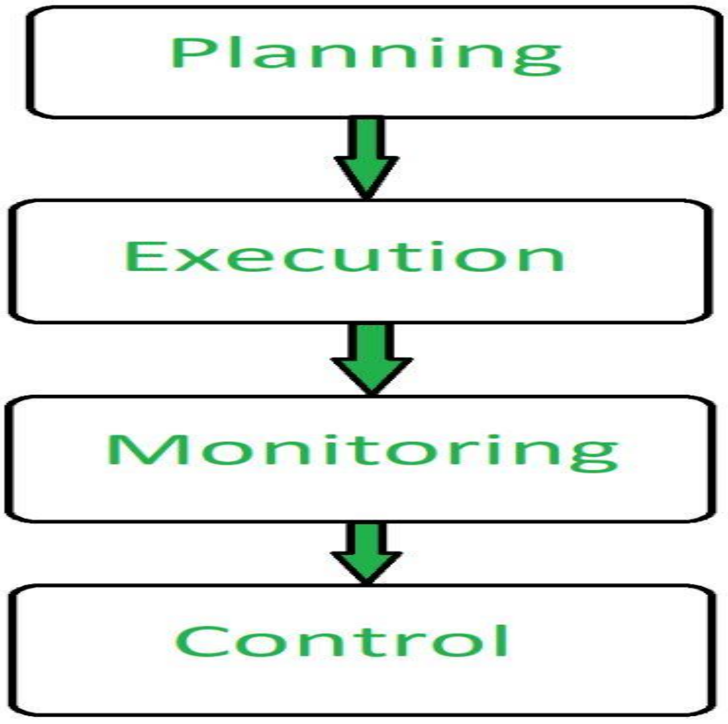
Abiding by a timeline is crucial to completing deliverables successfully. This is especially difficult when managing software projects because changes to the original project charter are unavoidable over time. To assure progress in the face of blockages or changes, software project managers ought to be specialists in managing risk and emergency preparedness. This Risk Mitigation and management is one of the core tenets of the philosophy of SPM.

5. Budget

Software project managers, like conventional project managers, are responsible for generating a project budget and adhering to it as closely as feasible, regulating spending, and reassigning funds as needed. SPM teaches us how to effectively manage the monetary aspect of projects to avoid running into a financial crunch later on in the project.

6. Maintenance

Software project management emphasizes continuous product testing to find and repair defects early, tailor the end product to the needs of the client, and keep the project on track. The software project manager makes ensuring that the product is thoroughly tested, analyzed, and adjusted as needed. Another point in favor of SPM.



How do you organize procedural abstractions in the context of data abstractions?

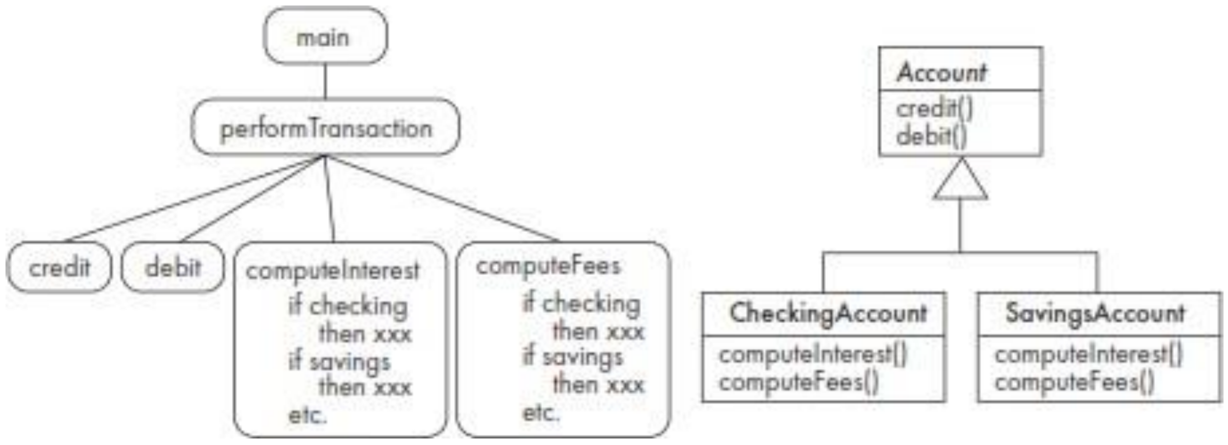
Organizing procedural abstractions in the context of data abstractions

The advantage of organizing programs around data abstractions is to make systems much simpler by putting all the procedures that access or modify a particular class of objects in one place, rather than having the procedures spread out all over the system. The idea of the object-oriented (OO) paradigm had become the best way to organize most systems.

Definition: The object-oriented paradigm is an approach to the solution of problems in which all computations are performed in the context of objects. The objects are instances of programming constructs, normally called classes, which are data abstractions and which contain procedural abstractions that operate on the objects.

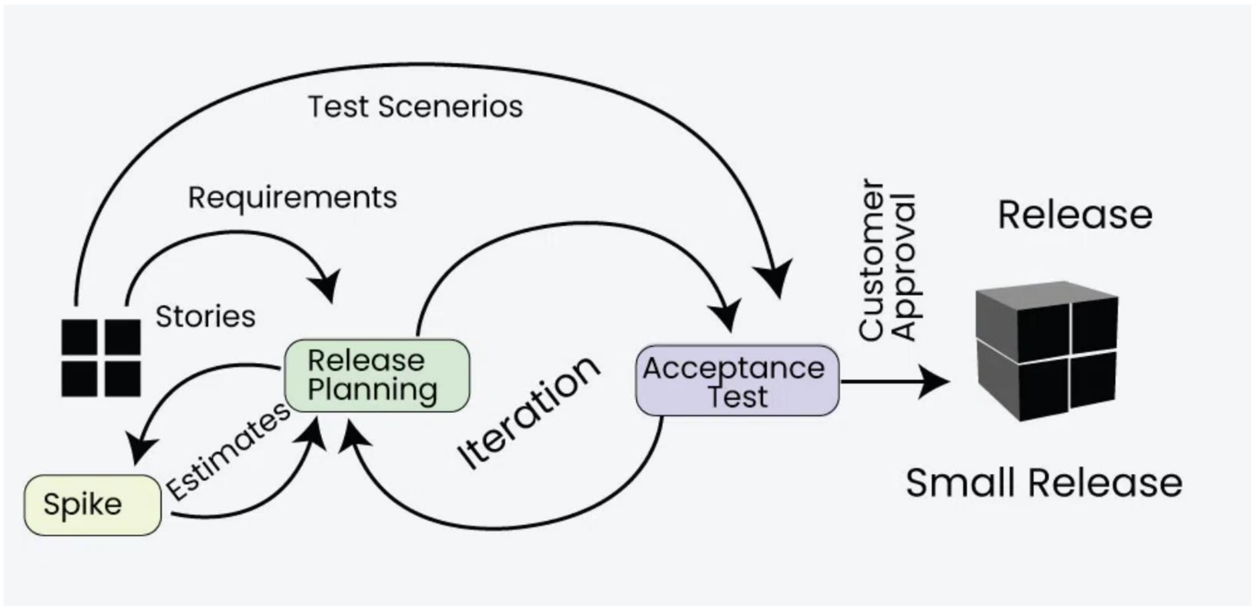
In the object-oriented paradigm, a running program can be seen as a collection of objects collaborating to perform a given task.

Figure summarizes the essential difference between the object-oriented and procedural paradigms. In the procedural paradigm (shown on the left), the code is organized into procedures that each manipulate different types of data. In the object-oriented paradigm (shown on the right), the code is organized into classes that each contain procedures for manipulating instances of that class alone.



What is Extreme Programming (XP)?

Extreme Programming (XP) is an Agile software development methodology that focuses on delivering high-quality software through frequent and continuous feedback, collaboration, and adaptation. XP emphasizes a close working relationship between the development team, the customer, and stakeholders, with an emphasis on rapid, iterative development and deployment.



Agile development approaches evolved in the 1990s as a reaction to documentation particularly the waterfall approach. Agile approaches are based on some common principles, some of which are:

1. Working software is the key measure of progress in a project.
2. For progress in a project, therefore software should be developed and delivered rapidly in small increments.
3. Even late changes in the requirements should be entertained.
4. Face-to-face communication is preferred over documentation.
5. Continuous feedback and involvement of customers are necessary for developing good-quality software.

6. A simple design that involves and improves with time is a better approach than doing an elaborate design up front for handling all possible scenarios.
7. The delivery dates are decided by empowered teams of talented individuals.

Extreme programming is one of the most popular and well-known approaches in the family of agile methods. an XP project starts with user stories which are short descriptions of what scenarios the customers and users would like the system to support. Each story is written on a separate card, so they can be flexibly grouped.

Good Practices in Extreme Programming

Some of the good practices that have been recognized in the extreme programming model and suggested to maximize their use are given below:

- **Code Review:** Code review detects and corrects errors efficiently. It suggests pair programming as coding and reviewing of written code carried out by a pair of programmers who switch their work between them every hour.
- **Testing:** Testing code helps to remove errors and improves its reliability. XP suggests test-driven development (TDD) to continually write and execute test cases. In the TDD approach, test cases are written even before any code is written.
- **Incremental development:** Incremental development is very good because customer feedback is gained and based on this development team comes up with new increments every few days after each iteration.
- **Simplicity:** Simplicity makes it easier to develop good-quality code as well as to test and debug it.
- **Design:** Good quality design is important to develop good quality software. So, everybody should design daily.
- **Integration testing:** Integration Testing helps to identify bugs at the interfaces of different functionalities. Extreme programming suggests that the developers should achieve continuous integration by building and performing integration testing several times a day.

Basic Principles of Extreme programming

XP is based on the frequent iteration through which the developers implement User Stories. User stories are simple and informal statements of the customer about the functionalities needed. A User Story is a conventional description by the user of a feature of the required system. It does not mention finer details such as the different scenarios that can occur. Based on User stories, the project team proposes Metaphors. Metaphors are a common vision of how the system would work. The development team may decide to build a Spike for some features. A Spike is a very simple program that is constructed to explore the suitability of a solution being proposed. It can be considered similar to a prototype. Some of the basic activities that are followed during software development by using the XP model are given below:

- **Coding:** The concept of coding which is used in the XP model is slightly different from traditional coding. Here, the coding activity includes drawing diagrams (modeling) that will be transformed into code, scripting a web-based system, and choosing among several alternative solutions.
- **Testing:** The XP model gives high importance to testing and considers it to be the primary factor in developing fault-free software.
- **Listening:** The developers need to carefully listen to the customers if they have to develop good quality software. Sometimes programmers may not have the depth knowledge of the system to be developed. So, the programmers should understand properly the functionality of the system and they have to listen to the customers.
- **Designing:** Without a proper design, a system implementation becomes too complex, and very difficult to understand the solution, thus making maintenance expensive. A good design results elimination of complex dependencies within a system. So, effective use of suitable design is emphasized.
- **Feedback:** One of the most important aspects of the XP model is to gain feedback to understand the exact customer needs. Frequent contact with the customer makes the development effective.
- **Simplicity:** The main principle of the XP model is to develop a simple system that will work efficiently in the present time, rather than trying to build something that would

take time and may never be used. It focuses on some specific features that are immediately needed, rather than engaging time and effort on speculations of future requirements.

- **Pair Programming:** XP encourages pair programming where two developers work together at the same workstation. This approach helps in knowledge sharing, reduces errors, and improves code quality.
- **Continuous Integration:** In XP, developers integrate their code into a shared repository several times a day. This helps to detect and resolve integration issues early on in the development process.
- **Refactoring:** XP encourages refactoring, which is the process of restructuring existing code to make it more efficient and maintainable. Refactoring helps to keep the codebase clean, organized, and easy to understand.
- **Collective Code Ownership:** In XP, there is no individual ownership of code. Instead, the entire team is responsible for the codebase. This approach ensures that all team members have a sense of ownership and responsibility towards the code.
- **Planning Game:** XP follows a planning game, where the customer and the development team collaborate to prioritize and plan development tasks. This approach helps to ensure that the team is working on the most important features and delivers value to the customer.
- **On-site Customer:** XP requires an on-site customer who works closely with the development team throughout the project. This approach helps to ensure that the customer’s needs are understood and met, and also facilitates communication and feedback.

Applications of Extreme Programming (XP)

Some of the projects that are suitable to develop using the XP model are given below:

- **Small projects:** The XP model is very useful in small projects consisting of small teams as face-to-face meeting is easier to achieve.
- **Projects involving new technology or Research projects:** This type of project faces changing requirements rapidly and technical problems. So XP model is used to complete this type of project.

- **Web development projects:** The XP model is well-suited for web development projects as the development process is iterative and requires frequent testing to ensure the system meets the requirements.
- **Collaborative projects:** The XP model is useful for collaborative projects that require close collaboration between the development team and the customer.
- **Projects with tight deadlines:** The XP model can be used in projects that have a tight deadline, as it emphasizes simplicity and iterative development.
- **Projects with rapidly changing requirements: The** XP model is designed to handle rapidly changing requirements, making it suitable for projects where requirements may change frequently.

Life Cycle of Extreme Programming (XP)

The Extreme Programming Life Cycle consist of five phases:

1. **Planning:** The first stage of Extreme Programming is planning. During this phase, clients define their needs in concise descriptions known as user stories. The team calculates the effort required for each story and schedules releases according to priority and effort.
2. **Design:** The team creates only the essential design needed for current user stories, using a common analogy or story to help everyone understand the overall system architecture and keep the design straightforward and clear.
3. **Coding:** Extreme Programming (XP) promotes pair programming i.e**.** wo developers work together at one workstation, enhancing code quality and knowledge sharing. They write tests before coding to ensure functionality from the start (TDD), and frequently integrate their code into a shared repository with automated tests to catch issues early.

4. **Testing:** Extreme Programming (XP) gives more importance to testing that consist of both unit tests and acceptance test**.** Unit tests, which are automated, check if specific features work correctly. Acceptance tests, conducted by customers, ensure that the overall system meets initial requirements. This continuous testing ensures the software's quality and alignment with customer needs.
5. **Listening:** In the listening phase regular feedback from customers to ensure the product meets their needs and to adapt to any changes.

Values of Extreme Programming (XP)

There are five core values of Extreme Programming (XP)

1. **Communication:** The essence of communication is for information and ideas to be exchanged amongst development team members so that everyone has an understanding of the system requirements and goals. Extreme Programming (XP) supports this by allowing open and frequent communication between members of a team.
2. **Simplicity:** Keeping things as simple as possible helps reduce complexity and makes it easier to understand and maintain the code.
3. **Feedback:** Feedback loops which are constant are among testing as well as customer involvements which helps in detecting problems earlier during development.
4. **Courage:** Team members are encouraged to take risks, speak up about problems, and adapt to change without fear of repercussions.
5. **Respect**: Every member's input or opinion is appreciated which promotes a collective way of working among people who are supportive within a certain group.

Advantages of Extreme Programming (XP)

- **Slipped schedules:** Timely delivery is ensured through slipping timetables and doable development cycles.
- **Misunderstanding the business and/or domain −** Constant contact and explanations are ensured by including the client on the team.
- **Canceled projects:** Focusing on ongoing customer engagement guarantees open communication with the consumer and prompt problem-solving.
- **Staff turnover:** Teamwork that is focused on cooperation provides excitement and goodwill. Team spirit is fostered by multidisciplinary cohesion.

- **Costs incurred in changes:** Extensive and continuing testing ensures that the modifications do not impair the functioning of the system. A functioning system always guarantees that there is enough time to accommodate changes without impairing ongoing operations.
- **Business changes:** Changes are accepted at any moment since they are seen to be inevitable.

Explain About Reliability Growth Models

Reliability Growth Modeling

A reliability growth model is a simulation of how system dependability evolves overtime throughout the testing process. When system failures are identified, the underlying flaws that are generating these failures are corrected, and the system's dependability should improve through system testing and debugging. The conceptual reliability growth model must next be converted into a mathematical model in order to forecast dependability.

Reliability growth modeling entails comparing observed reliability at various periods in time with known functions that demonstrate potential changes in reliability. An equal step function, for example, implies that the dependability of a system rises linearly with each release. It is feasible to forecast the system's dependability at some future point in time by comparing observed reliability increase with one of these functions. As a result, reliability growth models may be utilized to help in project planning.

The reliability growth model group measures and forecasts the improvement of reliability programs through testing. The growth model depicts a system's dependability or failure rate as a function of time or the number of test cases. The models in this category are listed below.

What is Jelinski Moranda software reliability model?

The **Jelinski-Moranda Software Reliability Model** is a mathematical model used to predict the reliability of software systems. It was developed by M.A. Jelinski and P.A. Moranda in 1972 and is based on the assumption that the rate of software failures follows a non-homogeneous Poisson process. This model assumes that the software system can be represented as a series of independent components, each with its own failure rate. The failure rate of each component is assumed to be constant over time.

The model assumes that software failures occur randomly over time and that the probability of failure decreases as the number of defects in the software is reduced.

The Jelinski-Moranda model uses an exponential distribution to model the rate of fault detection and assumes that the fault detection rate is proportional to the number of remaining faults in the software. The model can be used to predict the number of remaining faults in the software and to estimate the time required to achieve the desired level of reliability.

Assumptions Based on Jelinski-Moranda Model

- The number of faults in the software is known.
- The rate of fault detection is constant over time.
- The software system operates in a steady-state condition.
- The faults in the software are assumed to be independent and identically distributed.
- The fault removal process is assumed to be perfect, meaning that once a fault is detected, it is immediately removed without introducing any new faults.
- The testing process is assumed to be exhaustive, meaning that all faults in the software will eventually be detected.
- The model assumes that the software system will not be modified during the testing period and that the number and types of faults in the software will remain constant.
- The Jelinski-Moranda model assumes that faults are introduced into the software during the development process and that they are not introduced by external factors such as hardware failures or environmental conditions.
- The model assumes that the testing process is carried out using a specific testing methodology and that the results are consistent across different testing methodologies.

One limitation of the Jelinski-Moranda model is that it assumes a constant fault detection rate, which may not be accurate in practice. Additionally, the model does not take into account factors such as software complexity, hardware reliability, or user behavior, which can also affect the reliability of the software system.

Overall, the Jelinski-Moranda model is a useful tool for predicting software reliability, but it should be used in conjunction with other techniques and methods for software testing and quality assurance.

oftware Engineering Littlewood and Verall's Model

Last Updated : 24 Feb, 2023

•

Littlewood and Verrall's Model is a software reliability model that was proposed by Littlewood and Verrall in the 1990s. It is an extension of the Jelinski-Moranda (J-M) model and is also known as the J-M/L-V model. It estimates the reliability of software systems.

Assumptions:

The assumptions in this model include the following:

1. Failures are independent and identically distributed
2. Failure rates over time due to changes in patterns and software environment
3. The failure rate function λ(t) is non-decreasing and non-negative over time.
4. The time intervals between failures are distributed exponentially.
5. The size of the software system under test is known and has remained constant over time.

Comparison between Littlewood and Verall's Model and the Jelinski-Moranda (J-M) Model

Littlewood and Verall's Model	**Jelinski-Moranda (J-M) Model**
To predict the expected number of faults remaining in a software system at any given time	To predict the program failure rate at each interval of time
The faults in the software are corrected at a constant rate and the rate of correction is proportional to the number of faults present	There are N initial faults in the program and each fault is independent and equally likely to cause failure. The time intervals between failures are independent of each

Littlewood and Verall’s Model	Jelinski-Moranda (J-M) Model
	other and whenever a failure occurs, a corresponding fault is removed
R(t) = R(0) * e^(-lambda * t)	$\lambda(ti) = \varphi\ [N-(i-1)]$

It can be seen that Littlewood and Verall’s Model is focused on the number of faults remaining in the system, while the J-M model is focused on the program failure rate at each interval of time.

In general, Littlewood and Verrall’s Model is a more sophisticated and realistic approach to modeling software reliability than the J-M model. It is a useful tool for predicting software reliability in real-world applications because it accounts for variable failure rates.

What is Load Testing

Load testing determines the behavior of the application when multiple users use it at the same time. It is the response of the system measured under varying load conditions.

1. The load testing is carried out for normal and extreme load conditions.
2. Load testing is a type of performance testing that simulates a real-world load on a system or application to see how it performs under stress.
3. The goal of load testing is to identify bottlenecks and determine the maximum number of users or transactions the system can handle.
4. It is an important aspect of software testing as it helps ensure that the system can handle the expected usage levels and identify any potential issues before the system is deployed to production.

During load testing, various scenarios are simulated to test the system’s behavior under different load conditions. This can include simulating a high number of concurrent users, simulating numerous requests, and simulating heavy network traffic. The system's performance is then measured and analyzed to identify any bottlenecks or issues that may occur.

For a deeper dive into Load Testing and other crucial software testing techniques, you can also explore this comprehensive Software Testing Course, which provides practical insights and hands-on training.

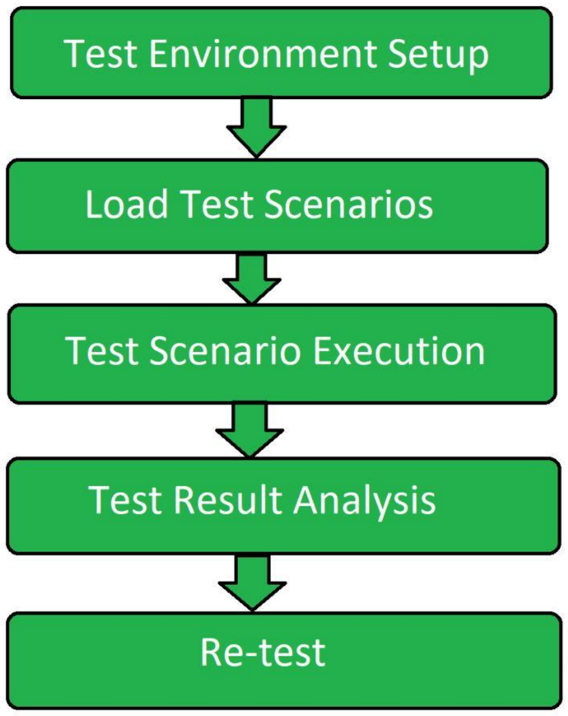
Load Testing Techniques

1. **Stress testing:** Testing the system's ability to handle a high load above normal usage levels
2. **Spike testing:** Testing the system's ability to handle sudden spikes in traffic
3. **Soak testing**: Testing the system's ability to handle a sustained load over a prolonged period of time
4. **Tools for Performance Testing:** Make use of specialized load testing tools like Locust, Gatling, JMeter, LoadRunner, and Apache Benchmark. These tools assist in gathering performance measurements and simulating a large number of users.
5. **Specify the Test Objectives:** Clearly state what your load test's goals are. Recognize the required response times, transaction volumes and expected user behavior.
6. **Determine Crucial Situations:** Determine the essential user scenarios that correspond to common usage patterns. A variety of actions, including user logins, searches, form submissions and other significant interactions, should be covered by these scenarios.

Objectives of Load Testing

1. **Evaluation of Scalability:** Assess the system's ability to handle growing user and transaction demands. Find the point at which the system begins to function badly.
2. **Planning for Capacity:** Describe the system's ability to accommodate anticipated future increases in the number of users, transactions and volume of data. Making well-informed decisions regarding infrastructure upgrades is made easier by this.
3. **Determine bottlenecks:** Identify and localize bottlenecks in the application or infrastructure's performance. Finding the places where the system's performance can suffer under load is part of this.
4. **Analysis of Response Time:** For crucial transactions and user interactions, track and evaluate response times. Make that the system responds to changes in load with reasonable response times.
5. **Finding Memory Leaks**: Find and fix memory leaks that may eventually cause a decline in performance. Make sure the programme doesn't use up too many resources when it's running.

Load Testing Process



Load Testing

1. Test Environment Setup**:** Firstly create a dedicated test environment setup for performing the load testing. It ensures that testing would be done in a proper way.
2. **Load Test Scenario:** In second step load test scenarios are created. Then load testing transactions are determined for an application and data is prepared for each transaction.
3. **Test Scenario Execution:** Load test scenarios that were created in previous step are now executed. Different measurements and metrices are gathered to collect the information.
4. **Test Result Analysis:** Results of the testing performed is analyzed and various recommendations are made.
5. **Re-test:** If the test is failed then the test is performed again in order to get the result in correct way.

Metrics of Load Testing

Metrics are used in knowing the performance of load testing under different circumstances. It tells how accurately the load testing is working under different test cases. It is usually carried out after the preparation of load test scripts/cases. There are many metrics to evaluate the load testing. Some of them are listed below.

1. Average Response Time

It tells the average time taken to respond to the request generated by the clients or customers or users. It also shows the speed of the application depending upon the time taken to respond to the all requests generated.

2. Error Rate

The Error Rate is mentioned in terms of percentage denotes the number of errors occurred during the requests to the total number of requests. These errors are usually raised when the application is no longer handling the request at the given time or for some other technical problems. It makes the application less efficient when the error rate keeps on increasing.

3. Throughput

This metric is used in knowing the range of bandwidth consumed during the load scripts or tests and it is also used in knowing the amount of data which is being used for checking the request that flows between the user server and application main server. It is measured in kilobytes per second.

4. Requests Per Second

It tells that how many requests are being generated to the application server per second. The requests could be anything like requesting of images, documents, web pages, articles or any other resources.

5. Concurrent Users

This metric is used to take the count of the users who are actively present at the particular time or at any time. It just keeps track of count those who are visiting the application at any time without raising any request in the application. From this, we can easily know that at which time the high number of users are visiting the application or website.

6. Peak Response Time

Peak Response Time measures the time taken to handle the request. It also helps in finding the duration of the peak time(longest time) at which the request and response cycle is handled and finding that which resource is taking longer time to respond the request.

Eliminate testing issues and streamline your development process using AI-powered testing partner that automates repetitive tasks, ensures test reliability, and boosts test coverage to allow developers to focus on what truly *matters: innovation.*

Load Testing Tools

1. Apache Jmeter: is an open-source tool used for performance testing and measuring the load and functional behavior of web applications. It simulates multiple users

sending requests to a web server, analyzes the server's response, and measures performance metrics such as response time, throughput, and resource utilization.

2. WebLoad: It is a performance testing tool designed to simulate user load on web applications and measure their behavior under various conditions. It helps identify performance bottlenecks and ensure that web applications can handle expected traffic.
3. NeoLoad: It is a performance testing tool used to simulate user traffic and measure how well applications handle load and stress. It helps identify bottlenecks and performance issues by generating virtual users to test the application's scalability and reliability.
4. LoadNinja: It is a cloud-based performance testing tool that enables users to simulate real-world user loads on their applications. It provides detailed insights into application performance and scalability by running load tests in real-time, helping teams identify and resolve performance bottlenecks.
5. HP Performance Tester: HP Performance Tester, now known as Micro Focus LoadRunner, is a performance testing tool used to simulate virtual users and measure how well an application handles various loads. It helps identify performance bottlenecks by generating load on the application and analyzing its response times and behavior .

Advantages of Load Testing :

Load testing has several advantages that make it an important aspect of software testing:

1. **Identifying bottlenecks:** Load testing helps identify bottlenecks in the system such as slow database queries, insufficient memory, or network congestion. This helps

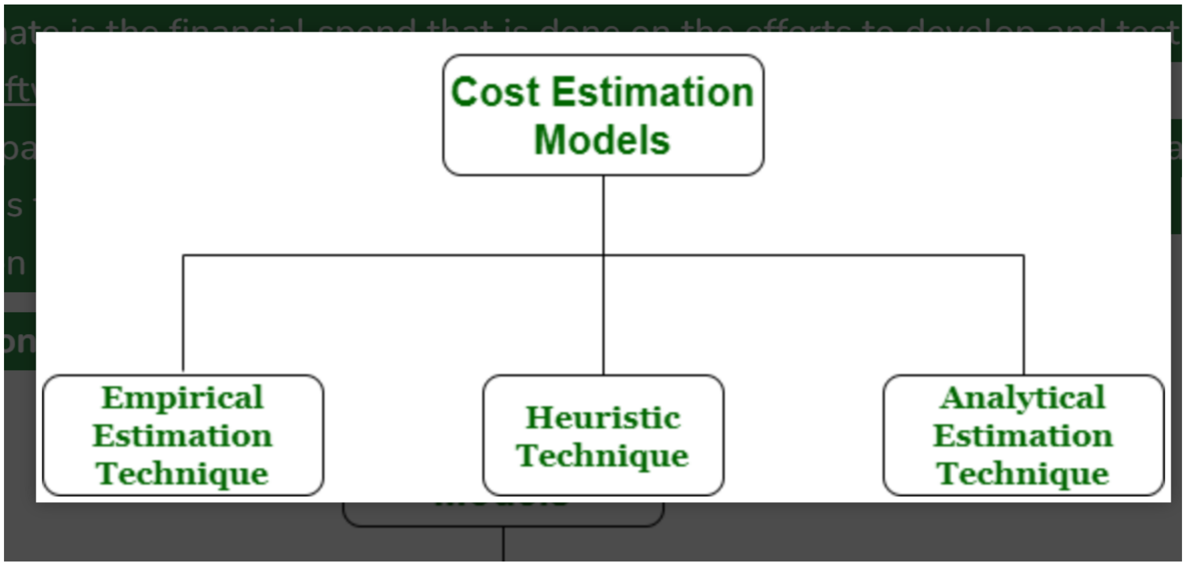
developers optimize the system and ensure that it can handle the expected number of users or transactions.

2. **Improved scalability:** By identifying the system’s maximum capacity, load testing helps ensure that the system can handle an increasing number of users or transactions over time. This is particularly important for web-based systems and applications that are expected to handle a high volume of traffic.
3. **Improved reliability:** Load testing helps identify any potential issues that may occur under heavy load conditions, such as increased error rates or slow response times. This helps ensure that the system is reliable and stable when it is deployed to production.

Cost Estimation Models in Software Engineering

Cost estimation simply means a technique that is used to find out the cost estimates. The cost estimate is the financial spend that is done on the efforts to develop and test software in Software Engineering. Cost estimation models are some mathematical algorithms or parametric equations that are used to estimate the cost of a product or a project. Various techniques or models are available for cost estimation, also known as Cost Estimation Models.

Cost Estimation Models as shown below :


Cost Estimation Models
Empirical Estimation Technique
Heuristic Technique
Analytical Estimation Technique

1. **Empirical Estimation Technique –** Empirical estimation is a technique or model in which empirically derived formulas are used for predicting the data that are a required and essential part of the software project planning step. These techniques are usually based on the data that is collected previously from a project and also based on some guesses, prior experience with the development of similar types of projects, and assumptions. It uses the size of the software to estimate the effort. In this technique, an educated guess of project parameters is made. Hence, these models are based on common sense. However, as there are many activities involved in empirical estimation techniques, this technique is formalized. For example Delphi technique and Expert Judgement technique.

2. **Heuristic Technique –** Heuristic word is derived from a Greek word that means "to discover". The heuristic technique is a technique or model that is used for solving problems, learning, or discovery in the practical methods which are used for achieving immediate goals. These techniques are flexible and simple for taking quick decisions through shortcuts and good enough calculations, most probably when working with complex data. But the decisions that are made using this technique are necessary to be optimal. In this technique, the relationship among different project parameters is expressed using mathematical equations. The popular heuristic technique is given by Constructive Cost Model (COCOMO). This technique is also used to increase or speed up the analysis and investment decisions.

3. **Analytical Estimation Technique –** Analytical estimation is a type of technique that is used to measure work. In this technique, firstly the task is divided or broken down into its basic component operations or elements for analyzing. Second, if the standard time is available from some other source, then these sources are applied to each element or component of work. Third, if there is no such time available, then the work is estimated based on the experience of the work. In this technique, results are derived by making certain basic assumptions about the project. Hence,

What is Functional Point Analysis?

Function Point Analysis was initially developed by Allan J. Albrecht in 1979 at IBM and has been further modified by the **International Function Point User's Group (IFPUG) i**n 1984, to clarify rules, establish standards, and encourage their use and evolution. Allan J. Albrecht gave the initial definition**,** Functional Point Analysis gives a dimensionless number defined in function points which we have found to be an effective relative measure of function value delivered to our customer. A systematic approach to measuring the different functionalities of a software application is offered by function point metrics. Function point metrics evaluate functionality from the perspective of the user, that is, based on the requests and responses they receive.

What are the Objectives of Functional Point Analysis

1. **Encourage Approximation:** FPA helps in the estimation of the work, time, and materials needed to develop a software project. Organizations can plan and manage projects more accurately when a common measure of functionality is available.
2. **To assist with project management:** Project managers can monitor and manage software development projects with the help of FPA. Managers can evaluate productivity, monitor progress, and make well-informed decisions about resource allocation and project timeframes by measuring the software's functional points.
3. **Comparative analysis:** By enabling benchmarking, it gives businesses the ability to assess how their software projects measure up to industry standards or best practices in terms of size and complexity. This can be useful for determining where improvements might be made and for evaluating how well development procedures are working.
4. **Improve Your Cost-Benefit Analysis:** It offers a foundation for assessing the value provided by the program concerning its size and complexity, which helps with cost-benefit analysis. Making educated judgements about project investments and resource allocations can benefit from having access to this information.

5. **Comply with Business Objectives:** It assists in coordinating software development activities with an organization's business objectives. It guarantees that software development efforts are directed toward providing value to end users by concentrating on user-oriented functionality.

Constructive Cost Model (COCOMO)

The Constructive Cost Model (COCOMO) is a software cost estimation model that helps predict the effort, cost, and schedule required for a software development project. Developed by Barry Boehm in 1981, COCOMO uses a mathematical formula based on the size of the software project, typically measured in lines of code (LOC).

What is the COCOMO Model?

The COCOMO Model is a procedural cost estimate model for software projects and is often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality. It was proposed by Barry Boehm in 1981 and is based on the study of 63 projects, which makes it one of the best-documented models.

The key parameters that define the quality of any software projects, which are also an outcome of COCOMO, are primarily effort and schedule:

1. **Effort:** Amount of labor that will be required to complete a task. It is measured in person-months units.
2. **Schedule:** This simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put in. It is measured in the units of time such as weeks, and months.

Types of Projects in the COCOMO Model

In the COCOMO model, software projects are categorized into three types based on their complexity, size, and the development environment. These types are:

1. **Organic:** A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.
2. **Semi-detached**: A software project is said to be a Semi-detached type if the vital characteristics such as team size, experience, and knowledge of the various programming environments lie in between organic and embedded. The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience better guidance and creativity. Eg: Compilers or , Embedded Systems can be considered Semi-Detached types.
3. **Embedded:** A software project requiring the highest level of complexity, creativity, and experience requirement falls under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

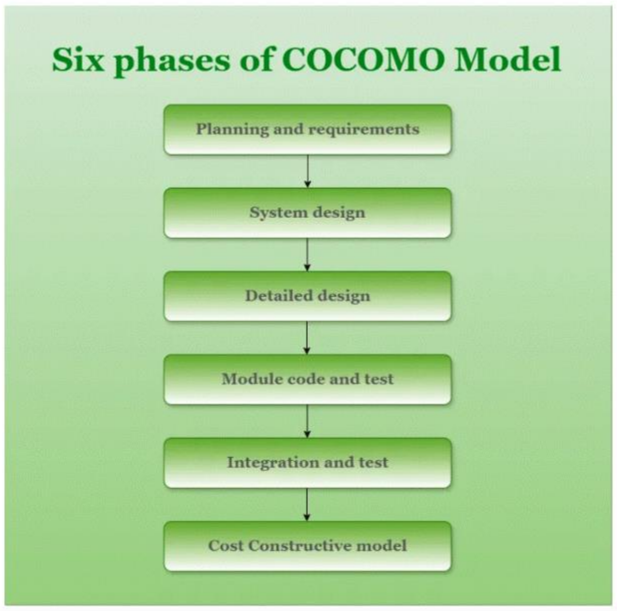
Comparison of these three types of Projects in COCOMO Model

Aspects	Organic	Semidetached	Embedded
Project Size	2 to 50 KLOC	50-300 KLOC	300 and above KLOC
Complexity	Low	Medium	High
Team Experience	Highly experienced	Some experienced as well as inexperienced staff	Mixed experience, includes experts
Environment	Flexible, fewer constraints	Somewhat flexible, moderate constraints	Highly rigorous, strict requirements
Effort Equation	E = 2.4(400)1.05	E = 3.0(400)1.12	E = 3.6(400)1.20
Example	Simple payroll system	New system interfacing with existing systems	Flight control software

Detailed Structure of COCOMO Model

Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step of the software engineering process. The detailed model uses different effort multipliers for each cost driver attribute. In detailed COCOMO, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:



Phases of COCOMO Model

1. **Planning and requirements:** This initial phase involves defining the scope, objectives, and constraints of the project. It includes developing a project plan that outlines the schedule, resources, and milestones
2. **System design:** : In this phase, the high-level architecture of the software system is created. This includes defining the system's overall structure, including major components, their interactions, and the data flow between them**.**

3. **Detailed design:** This phase involves creating detailed specifications for each component of the system. It breaks down the system design into detailed descriptions of each module, including data structures, algorithms, and interfaces.
4. **Module code and test:** This involves writing the actual source code for each module or component as defined in the detailed design. It includes coding the functionalities, implementing algorithms, and developing interfaces.
5. **Integration and test:** This phase involves combining individual modules into a complete system and ensuring that they work together as intended.
6. **Cost Constructive model:** The **Constructive Cost Model (COCOMO)** is a widely used method for estimating the cost and effort required for software development projects.

Different models of **COCOMO** have been proposed to predict the cost estimation at different levels, based on the amount of accuracy and correctness required. All of these models can be applied to a variety of projects, whose characteristics determine the value of the constant to be used in subsequent calculations. These characteristics of different system types are mentioned below. Boehm's definition of organic, semidetached, and embedded systems:

Importance of the COCOMO Model

1. **Cost Estimation:** To help with resource planning and project budgeting, COCOMO offers a methodical approach to software development cost estimation.
2. **Resource Management:** By taking team experience, project size, and complexity into account, the model helps with efficient resource allocation.
3. **Project Planning**: COCOMO assists in developing practical project plans that include attainable objectives, due dates, and benchmarks.

4. **Risk management**: Early in the development process, COCOMO assists in identifying and mitigating potential hazards by including risk elements.
5. **Support for Decisions**: During project planning, the model provides a quantitative foundation for choices about scope, priorities, and resource allocation.
6. **Benchmarking**: To compare and assess various software development projects to industry standards, COCOMO offers a benchmark.
7. **Resource Optimization:** The model helps to maximize the use of resources, which raises productivity and lowers costs.

Types of COCOMO Model

There are three types of COCOMO Model:

- Basic COCOMO Model
- Intermediate COCOMO Model
- Detailed COCOMO Model

1. Basic COCOMO Model

The Basic COCOMO model is a straightforward way to estimate the effort needed for a software development project. It uses a simple mathematical formula to predict how many person-months of work are required based on the size of the project, measured in thousands of lines of code (KLOC).

It estimates effort and time required for development using the following expression:

$E = a*(KLOC)b$ PM

$T\ dev = c*(E)d$

Person required = Effort/ Time

Where,

E is effort applied in Person-Months

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

T dev is the development time in months

a, b, c are constants determined by the category of software project given in below table.

The above formula is used for the cost estimation of the basic COCOMO model and also is used in the subsequent models. The constant values a, b, c, and d for the Basic Model for the different categories of the software projects are:

Software Projects	**A**	**B**	**c**	**D**
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

1. The effort is measured in Person-Months and as evident from the formula is dependent on Kilo-Lines of code. The development time is measured in months.
2. These formulas are used as such in the Basic Model calculations, as not much consideration of different factors such as reliability, and expertise is taken into account, hence forth the estimate is rough.

Example of Basic COCOMO Model

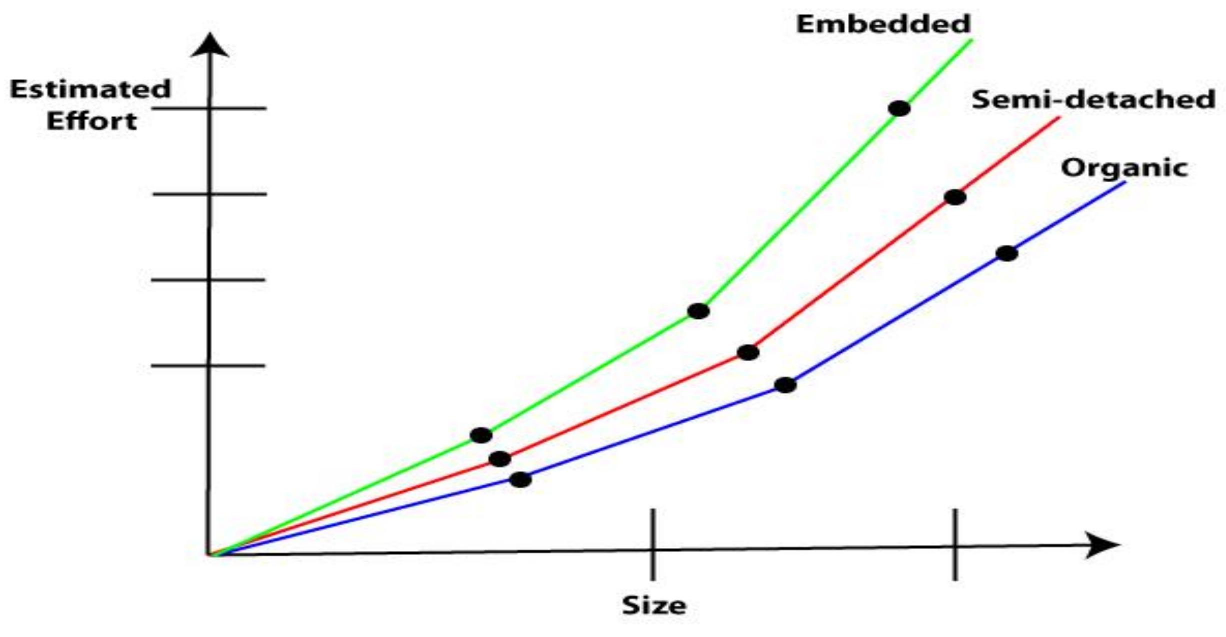
Suppose that a Basic project was estimated to be 400 KLOC (kilo lines of code). Calculate effort and time for each of the three modes of development. All the constants value provided in the following table:

Solution

From the above table we take the value of constant a,b,c and d.

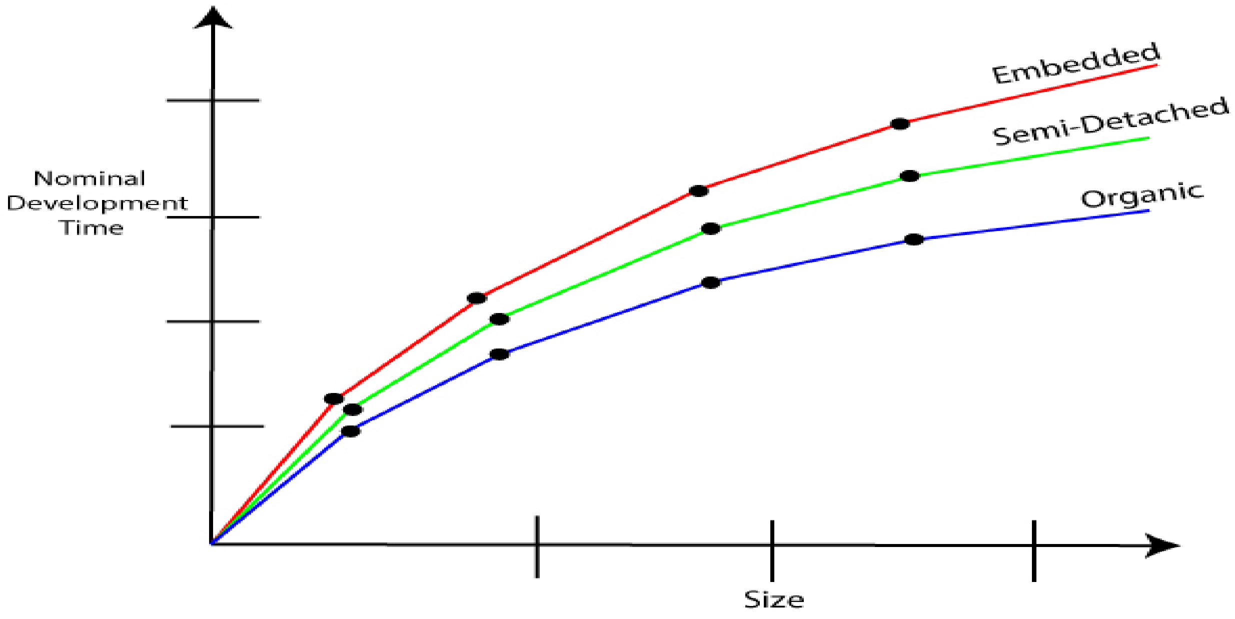
1. For organic mode,
   - effort = $2.4 \times (400)_{1.05} \approx 1295$ person-month.
   - dev. time = $2.5 \times (1295)_{0.38} \approx 38$ months.
2. For semi-detach mode,
   - effort = $3 \times (400)_{1.12} \approx 2462$ person-month.
   - dev. time = $2.5 \times (2462)_{0.35} \approx 38$ months.
3. For Embedded mode,
   - effort = $3.6 \times (400)_{1.20} \approx 4772$ person-month.
   - dev. time = $2.5 \times (4772)_{0.32} \approx 38$ months

Some insight into the basic COCOMO model can be obtained by plotting the estimated characteristics for different software sizes. Fig shows a plot of estimated effort versus product size. From fig, we can observe that the effort is somewhat superliner in the size of the software product. Thus, the effort required to develop a product increases very rapidly with project size.



4. Effort versus product size

The development time versus the product size in KLOC is plotted in fig. From fig it can be observed that the development time is a sub linear function of the size of the product, i.e. when the size of the product increases by two times, the time to develop the product does not double but rises moderately. This can be explained by the fact that for larger products, a larger number of activities which can be carried out concurrently can be identified. The parallel activities can be carried out simultaneously by the engineers. This reduces the time to complete the project. Further, from fig, it can be observed that the development time is roughly the same for all three categories of products. For example, a 60 KLOC program can be developed in approximately 18 months, regardless of whether it is of organic, semidetached, or embedded type.



Development time versus size

From the effort estimation, the project cost can be obtained by multiplying the required effort by the manpower cost per month. But, implicit in this project cost computation is the assumption that the entire project cost is incurred on account of the manpower cost alone. In addition to manpower cost, a project would incur costs due to hardware and software required for the project and the company overheads for administration, office space, etc.

It is important to note that the effort and the duration estimations obtained using the COCOMO model are called a nominal effort estimate and nominal duration estimate. The term nominal implies that if anyone tries to complete the project in a time shorter than the estimated duration, then the cost will increase drastically. But, if anyone completes the project over a longer period of time than the estimated, then there is almost no decrease in the estimated cost value.

Intermediate Model: The basic Cocomo model considers that the effort is only a function of the number of lines of code and some constants calculated according to the various software systems. The intermediate COCOMO model recognizes these facts and refines the initial estimates obtained through the basic COCOMO model by using a set of 15 cost drivers based on various attributes of software engineering.

Classification of Cost Drivers and their attributes:

(i) Product attributes -

- Required software reliability extent
- Size of the application database
- The complexity of the product

Hardware attributes -

- Run-time performance constraints
- Memory constraints
- The volatility of the virtual machine environment
- Required turnabout time

Personnel attributes -

- Analyst capability
- Software engineering capability
- Applications experience
- Virtual machine experience
- Programming language experience

Project attributes -

- Use of software tools
- Application of software engineering methods
- Required development schedule

The cost drivers are divided into four categories:

Intermediate Model: The basic Cocomo model considers that the effort is only a function of the number of lines of code and some constants calculated according to the various software systems. The intermediate COCOMO model recognizes these facts and refines the initial estimates obtained through the basic COCOMO model by using a set of 15 cost drivers based on various attributes of software engineering.

Classification of Cost Drivers and their attributes:

(i) Product attributes -

- Required software reliability extent
- Size of the application database
- The complexity of the product

Hardware attributes -

- Run-time performance constraints
- Memory constraints
- The volatility of the virtual machine environment
- Required turnabout time

Personnel attributes -

- Analyst capability
- Software engineering capability
- Applications experience
- Virtual machine experience
- Programming language experience

Project attributes -

- Use of software tools
- Application of software engineering methods
- Required development schedule

The cost drivers are divided into four categories:

Cost Drivers	RATINGS					
	Very low	Low	Nominal	High	Very High	Extra High
Product Attributes						
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
Computer Attributes						
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	..

Cost Drivers	RATINGS					
	Very low	Low	Nominal	High	Very high	Extra high
Personnel Attributes						
ACAP	1.46	1.19	1.00	0.86	0.71	..
AEXP	1.29	1.13	1.00	0.91	0.82	..
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	..	..
Project Attributes						
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	..

Intermediate COCOMO equation:

$$E=a_i\ (KLOC)\qquad b_i*EAF$$

$$D=c_i\ (E)d_i$$

Coefficients for intermediate COCOMO

Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

3. Detailed COCOMO Model: Detailed COCOMO incorporates all qualities of the standard version with an assessment of the cost driver?s effect on each method of the software engineering process. The detailed model uses various effort multipliers for each cost driver property. In detailed cocomo, the whole software is differentiated into multiple modules, and then we apply COCOMO in various modules to estimate effort and then sum the effort.

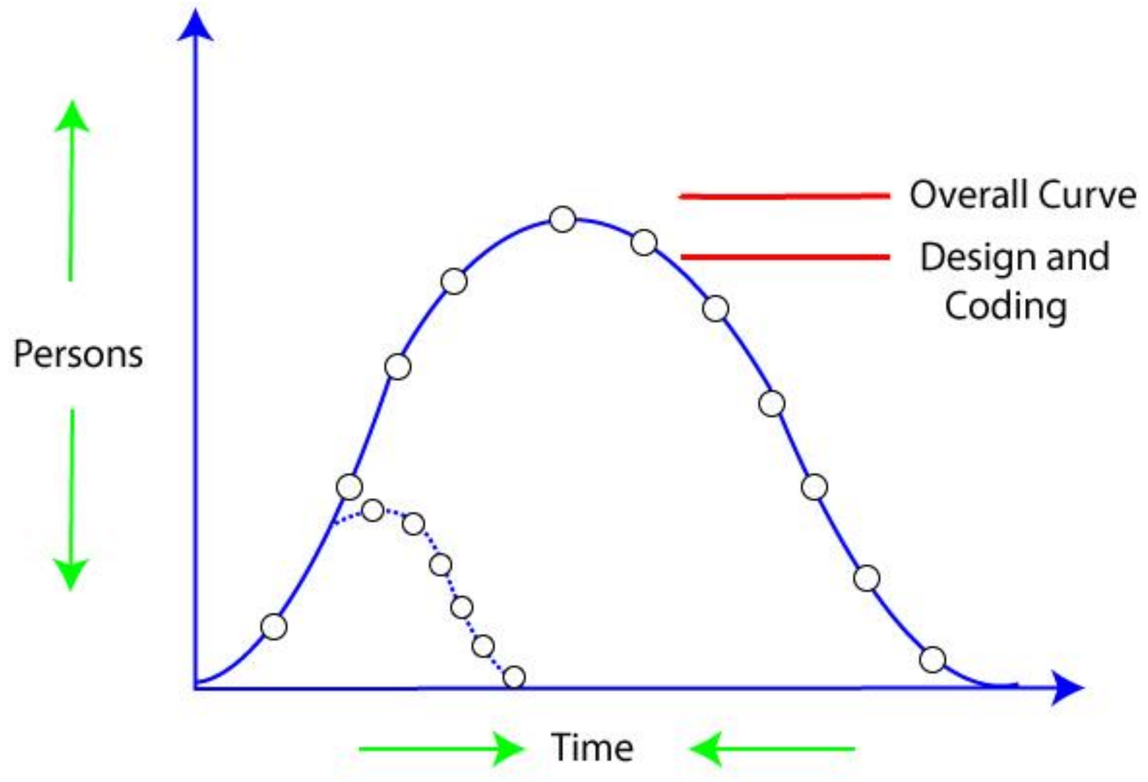
The Six phases of detailed COCOMO are:

1. Planning and requirements
2. System structure
3. Complete structure
4. Module code and test
5. Integration and test
6. Cost Constructive model

The effort is determined as a function of program estimate, and a set of cost drivers are given according to every phase of the software lifecycle.

Putnam Resource Allocation Model

The Lawrence Putnam model describes the time and effort requires finishing a software project of a specified size. Putnam makes a use of a so-called The Norden/Rayleigh Curve to estimate project effort, schedule & defect rate as shown in fig:



The Rayleigh manpower loading Curve

Putnam noticed that software staffing profiles followed the well known Rayleigh distribution. Putnam used his observation about productivity levels to derive the software equation:

$$L = C_k K^{1/3} t_d^{4/3}$$

The various terms of this expression are as follows:

K is the total effort expended (in PM) in product development, and L is the product estimate in **KLOC** .s

What is Risk?

"Tomorrow problems are today's risk." Hence, a clear definition of a "risk" is a problem that could cause some loss or threaten the progress of the project, but which has not happened yet.

These potential issues might harm cost, schedule or technical success of the project and the quality of our software device, or project team morale.

Risk Management is the system of identifying addressing and eliminating these problems before they can damage the project.

We need to differentiate risks, as potential issues, from the current problems of the project.