SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF MANAGEMENT STUDIES

SBAA3017 SOFTWARE PROJECT MANAGEMENT

Syllabus with Course objectives and Course outcomes

COURSE OBJECTIVES

- **To understand the fundamental principles of software project management.**
- **To have a good knowledge of responsibilities of project manager.**
- **To be familiar with the different methods and techniques used for project management.**

UNIT 1 INTRODUCTION 9 Hrs.

Defining of Software Development Process - Process - Tailoring the Process - Improving the process discipline - Need for implementing discipline. Software Production Process - Identify the Software Model - Software Process Models : Waterfall Model, Prototyping Model, RAD Model, Incremental Model, Spiral Model, Component Assembly Model - Software Life Cycle.

UNIT 2 SOFTWARE DEVELOPMENT 9 Hrs.

Software Development Team - Three Vital Aspects of Software Project Management - The Team - Meaning of Leadership - Communicating in Harmony - Personality traits - Project Organizations. Project Planning: Top-Down and Bottom-Up Planning - Types of Activity - Project Duration : Schedule Monitoring Tools - Gantt Chart, PERT Chart, Critical Path.

UNIT 3 PROJECT REVIEW 9 Hrs.

Tracking Meetings - Recovery plans - Schedule Work & Escalation Meetings. Project Engineering: Product Requirements - Understanding the Customer Problem to solve - Initial Investigation, Strategies for determining information requirements, Information gathering Tools - Product Objectives.

UNIT 4 PROBLEM SOLVING 9 Hrs.

Product Specifications - Defining the Final Product - Data Flow Diagram, Data Dictionary, Structured English, Decision Trees, Decision Tables - Feasibility Study. Software Testing : Test Plan - Development Testing : Verification and Validation -

General Testing Methods : White Box and Black Box Testing - Unit Testing - System Integration Testing - Validation Testing - System testing.

UNIT 5 SOFTWARE QUALITY 9 Hrs.

Software Quality - Quality Measures - FURPS - Software Quality Assurance - Software Reviews - Format Technical Review (FTR) Formal Approaches to SQA - Software Reliability - Introduction to SQA - The Software Quality Assurance Plan - Formal approaches to SQA - Clean room Methodology.

Max. 45 Hrs.

COURSE OUTCOMES

On completion of the course, student will be able to

CO1 - Apply project management concepts and techniques to an IT project.

CO2 - Identify issues that could lead to IT project success or failure.

CO3 - Explain project management in terms of the software development process.

CO4 - Describe the responsibilities of IT project managers.

CO5 - Apply project management concepts through working in a group as team leader

CO6 - Be an active team member on an IT project.

UNIT – I - INTRODUCTION – SBAA3017

UNIT 1 INTRODUCTION

Defining of Software Development Process - Process - Tailoring the Process - Improving the process discipline - Need for implementing discipline. Software Production Process - Identify the Software Model - Software Process Models: Waterfall Model, Prototyping Model, RAD Model, Incremental Model, Spiral Model, Component Assembly Model - Software Life Cycle.

1. Process

Project management process is **an administration process for the planning and control of the services or the implementation of a project**

The results of one of these processes are:

1. delivery of the project product
2. achievement of the project objectives
3. documentation of the learning processes

1.1 Project Management Process consists of the following 4 stages:

- Feasibility study
- Project Planning
- Project Execution
- Project Termination

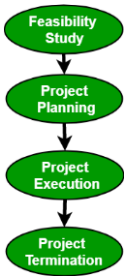


Fig.1.1 Process

1.1.1 Feasibility Study

- Feasibility study explores system requirements to determine project feasibility.

1.1.2 Project Planning

A detailed plan stating a stepwise strategy to achieve the listed objectives is an integral part of any project. Planning consists of the following activities:

- Set objectives or goals
- Develop strategies
- Develop project policies
- Determine courses of action
- Making planning decisions
- Set procedures and rules for the project
- Develop a software project plan
- Prepare budget
- Conduct risk management

- Document software project plans
- This step also involves the construction of a work breakdown structure(WBS). It also includes size, effort, schedule, and cost estimation using various techniques.

1.1.3 Project Execution

- A project is executed by choosing an appropriate software development lifecycle model(SDLC)
- It includes a number of steps including requirements analysis, design, coding, testing and implementation, testing, delivery, and maintenance.
- There are a number of factors that need to be considered while doing so including the size of the system, the nature of the project, time and budget constraints, domain requirements, etc.
- An inappropriate SDLC can lead to the failure of the project.

1.1.4 Project Termination:

- There can be several reasons for the termination of a project.
- Though expecting a project to terminate after successful completion is conventional, but at times, a project may also terminate without completion.
- Projects have to be closed down when the requirements are not fulfilled according to given time and cost constraints.
- Some of the reasons for failure include:

- Fast-changing technology
- Project running out of time
- Organizational politics
- Too much change in customer requirements
- Project exceeding budget or funds
- Once the project is terminated, a post-performance analysis is done. Also, a final report is published describing the experiences, lessons learned, recommendations for handling future projects.

1.2 Tailoring the Process

- Tailoring is the process of referencing framework documents, standards and other relevant sources and utilizing those elements that provide processes, tools and techniques that are suitable for that particular organization.
- It also includes modifying existing processes currently in use by the organization.
- The result of tailoring is that the project management methodology will be suitable for use in specific types of projects, and a tailored methodology will reflect the size,

complexity, and duration of the project as appropriate for the organizational context along with adaptation to the industry within which the project is undertaken.

- tailoring at two levels:
- Summary
- Detailed.

1.2.1 Summary-Level Tailoring

In summary-level tailoring, depending on the project characteristics, the project manager applies overall guidelines for tailoring the standard process. That is, it provides some general rules regarding certain types of detailed activities. To perform this step, the project manager first identifies certain characteristics of the project. For development projects, the following characteristics are used for tailoring:

- Experience and skill level of the team and the project manager
- Clarity of the requirements
- Project duration
- Application criticality

1.2.2 Detailed Tailoring

Detailed tailoring covers execution of activities, their review, and documentation needs. Tailoring guidelines may specify an activity as optional, in which case the project manager can decide whether or not to execute the activity. Similarly, preparation of some documents may be optional, in which case the project manager decides whether or not the project needs the document. For review, the general alternatives are Do group review, do one-person review, or Do not review. In addition, a project manager may add some new activities or may repeat some activities.

- When detailed tailoring is finished, the sequence of activities to be performed in the process for the project is defined. These definitions are then used to plan and schedule activities and form the basis of project execution. The tailoring performed is highlighted in the project plan, so the process definition and tailoring also are reviewed when the plan is reviewed.
- Process discipline is the adherence to well-thought-out and well-defined processes that are executed daily. The rules are often related to the work of process engineers and local leaders, e.g. line leaders, and production supervisors. Or rather, the lack of a particular type of work in this case.

1.3 Process Improvement Discipline

- As businesses try to accelerate growth while running lean, there’s always a desire to reduce costs through process improvement. Like the examples above, this could include:
- Improving product quality

- Upgrade service quality
- Improve delivery times
- Reduce billing cycles
- Make production more efficient

1.3.1 LEAN Technology

- LEAN Tech, also known as LEAN manufacturing, was a process that originated with Toyota. It was implemented to streamline the company's production chain and dramatically reduce operating and overhead costs.
- The key idea is to base process improvement on the customer perspective; taking the time to understand what they value from the product and then using LEAN process improvement to eliminate unnecessary waste, errors and other things that drive up costs. By focusing on value, the entire process is organized to drive more of what the customer is willing to pay for.

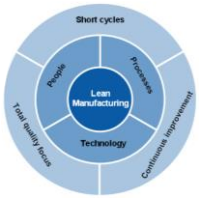


Fig.1.2 Lean Technology

1.3.2 Six Sigma

- Six Sigma is a process improvement example that focuses on achieving the maximum level of obtainable quality within an organization. At the Six Sigma level, that is a rating of near 100% perfection (or 99.9966%).
- The Six Sigma approach looks closely at the root cause of problems, defects, and variations that reduce the effectiveness of processes. At its heart, there's a philosophy of constant improvement that is in place to improve results consistently and progressively until that max level of perfection is achieved.

1.3.3 Total Quality Management (TQM)

- Total quality management (TQM) is a project management technique or strategy that is implemented to assure that an awareness of quality is embedded in all phases of the project from conception to completion. Used in industries from manufacturing to aerospace, total quality management requires the careful and consistent review of all phases of a project, and the coordinated effort of all involved. Communication is key to the process. Standards must be developed, procedures well defined, and all involved must follow strict adherence to the plan to assure its success.

1.3.3.1 The first paradigm Total Quality Management

- The first paradigm of the strategy, “total,” calls for an integrated system of dependent variables. Here, planning is of the utmost importance. In the planning stage obstacles to success can be identified, confronted, and conquered. Procedures for inspection, evaluating, reporting, and rectifying anomalies are established that will assure consistency and quality will be developed and implemented. All parties involved, from workers to management must become familiar with the process and committed to its success.

1.3.3.2 The second paradigm Total Quality Management

- The second paradigm of the total quality management (TQM) strategy, “quality,” requires that once a standard has been established it must never be violated. This necessitates a certain amount of cross training, so that one party along the chain of project management can spot the errors that might have occurred at a prior stage in the process. This not only assures quality, but also tends to inspire greater commitment and conscientiousness among the individuals involved.

1.3.3.3 The Third paradigm Total Quality Management

The third paradigm of total quality management (TQM) is management itself. In order to ensure success, the strategy must function as a cohesive element that unites the individual efforts of all involved.

The system must be managed properly in order to ensure the quality of the strategy itself. Therefore, it is the role of project management role to oversee the training of all involved in the process, to ascertain whether the process is being strictly adhered to, and to develop and implement corrective measures designed to rectify any problems that might divert the strategy from its goal.

1.4 The Importance of a Disciplined Process

- A disciplined software process serves two main purposes: Helps **developers** better understand what they are doing
- Helps **managers** make more accurate predictions about how long a project will take
    - Predictability is crucial for setting reasonable goals and planning resource allocation.

1.4.1 Understanding

- As software developers work through a disciplined process, they are developing a complex mental roadmap of: The values of the client
- The concepts that are important to the client
- Software patterns for achieving the desired behavior
- Implementation methods
- **Common sense** and **experience** both support the importance of this understanding.

1.4.2 Common Sense

- Suppose that a software development process has been in progress for several months or years. Then: One of the development team members has changed jobs so that a replacement is needed, or
- The project is behind schedule so management has allocated more people to work on the project.
- So a new member, who knows neither the process nor the client, has been added to the team.
- Will he/she be as effective as the other team members?
- Not likely.

1.4.3 Experience

- Brooks has discussed management problems arising from adding people to a project that is behind schedule in order to catch up: Often, the result is that the project slips further behind schedule.
- The reason: The new people do not have a mental roadmap.
- In order to acquire it, experienced team members will have to dedicate some time to bringing them up to speed.
- This may seem obvious, but managers have to go through a similar process to learn how to manage effectively.
- A lot of facts are obvious, but that does not mean that you will see them when they are important: You do not see things that are obvious until you have had experiences that stress their importance.
- You also need to recognize **that** they are important.

1.5 Software Development Methodologies

- There are a few different software development methodologies including
    - Waterfall
    - Agile
    - Rapid application development (RAD)
    - Extreme programming
    - Rational unified process (RUP)
    - Scrum

Software Development Approaches

1.6 Generic Software Process Models

1. **The waterfall model** (Linear Sequential Model)
    - Separate and distinct phases of specification and development

2. **Evolutionary development**
    - Specification, development and testing are interleaved

- **Formal systems development (example - ASML)**
    1. A mathematical system model is formally transformed to an implementation

1.6.1 Waterfall Methodology

- **Waterfall methodology** – A sequence of phases in which the output of each phase becomes the input for the next

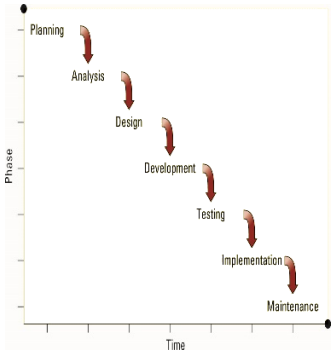


Fig.1.3 Waterfall model

Waterfall model Requirement and Design

Artefacts produced in the requirements and Design phases

- **SRS -S**oftware **R**equirements **s**pecification document
- SRS might include:
    - User usage stories (scenarios) – **Use cases**.
    - Static analysis – **class diagrams**.
    - Behavioural analysis – **sequence diagrams**, **state charts**.

The specification and design activities are heavily time consuming.

The requirements document

- The requirements document is the official statement of what is required of the system developers.
- Should include both a definition of user requirements and a specification of the system requirements.
- **It is NOT a design document**. As far as possible, it should set of WHAT the system should do rather than HOW it should do it

Users of a requirements document (SRS)

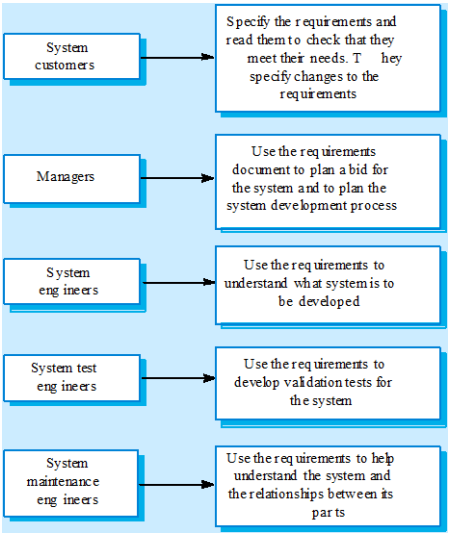


Fig.1.4 Users of SRS

The Waterfall process characterization:

- Figure out what
- Figure out how
- Then do it
- Verify was done right
- Hand product to customer
- Perfect requirement definition?

Waterfall model problems

- **Inflexible partitioning** of the project into distinct stages
- Difficult to respond to changing customer requirements

This model is only appropriate when the requirements are well-understood

Waterfall model describes a staged development process

- Based on hardware engineering models
- Change during development is unlikely
- Widely used in large systems: military and aerospace industries

Why Not a Waterfall

Because software is different:

- **No fabrication steps**
    - Program code is another design level
    - Hence, no “commit” step – software can always be changed.

- **No body of experience for design analysis (yet)**
    - Most analysis (testing) is done on program code
    - Hence, problems not detected until late in the process
- **Waterfall model takes a static view of requirements**
    - Ignore changing needs
    - Lack of user involvement once specification is written
- **Unrealistic separation of specification from the design**
- **Doesn't accommodate prototyping, reuse, etc**

1.6.2 Evolutionary development

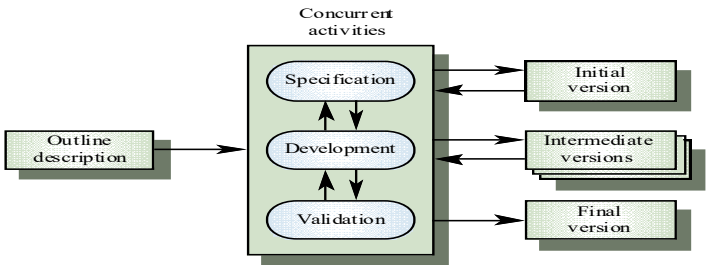


Fig.1.5 Evolutionary development

Characteristics of Evolutionary Development

- Modern development processes take ***evolution*** as fundamental, and try to provide ways of managing, rather than ignoring, the risk.
- System requirements always ***evolve*** during a project.
- Specification is ***evolved*** in conjunction with the software – No complete specification in the system development contract. Difficult for large customers.
- Two (related) process models:
    - **Incremental development**
    - **Spiral development**

Incremental Development

- Rather than deliver the system as a single delivery, **the development and delivery is broken down into increments** with each increment delivering part of the required functionality.
- **User requirements are prioritised,** and **the highest priority requirements are included in early increments.**
- **Once the development of an increment is started, the requirements are frozen** though requirements for later increments can continue to evolve.

Incremental Development – Problems

- Lack of process visibility.
- Systems are often poorly structured.
- **Applicability claims in the literature:**
    - For small or medium-size interactive systems.
    - For parts of large systems (e.g. the user interface).
    - For short-lifetime systems.

Incremental means adding, iterative means reworking (by Alistair Cockburn)

- **Incremental development** is a staging and scheduling strategy in which the various parts of the system are developed at different times or rates and integrated as they are completed. The alternative is to develop the entire system with a big bang integration at the end.
- **Iterative development** is a rework scheduling strategy in which time is set aside to revise and improve parts of the system. The alternative development is to get it right the first time (or at least declare that it is right!).

Iterate	**Increment**
fundamentally means *"change"*.	fundamentally means *"add onto"*.
repeating the process on the *same* section of work	repeating the process on a *new* section of work.
repeat the process (, design, implement, evaluate),	repeat the process (, design, implement, evaluate),

1.6.3 The Prototyping Model

- One main idea behind prototyping is for the development of fast prototypes and customer availability for feedback.
- Often prototyping tools are used to help
- Developers respond to feedback and add additional parts as application evolves into an acceptable product.
- Recognize this process can be **inserted** into the SDLC or other models.
- One main idea behind prototyping is for the development of fast prototypes and customer availability for feedback.
- Often prototyping tools are used to help

- Developers respond to feedback and add additional parts as application evolves into an acceptable product.
- Recognize this process can be **inserted** into the SDLC or other models.

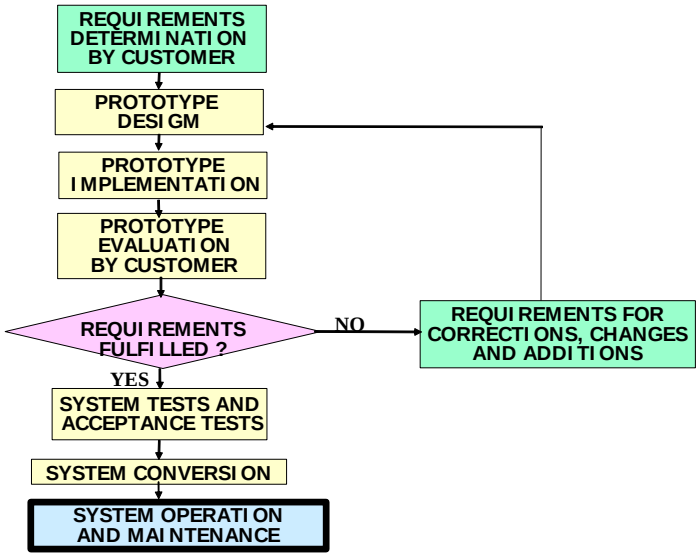


Fig1.6. Prototyping Model

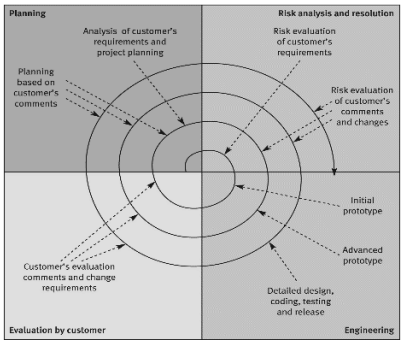
A good approach for small to medium-sized projects.Very important: customer involvement

Prototyping Model

- Advantages
    - Shorter development process
    - Substantial savings in development resources (time)
    - Better fit to customer requirements and reduced risk of project failure
    - Easier and faster user comprehension of new system
- Disadvantages
    - Less flexibility and adaptability to changes and additions
    - Reduced preparation for unexpected instances of failure

1.6.4 The Spiral Model

- A heavy-weight, plan-driven, highly-structured approach for large projects.
- Especially designed for those with higher chances of failure.
- Combines iterative model, emphasizes risk assessment, customer participation, prototyping, and more
- Definitely an iterative process.



Can see each spiral includes
Planning
Risk Analysis / Resolution
Engineering activities
(design, code, test…)
Customer Evaluation
(errors, changes, new requirements…)

Fig.1.7 Spiral Model

Win Win spiral Model

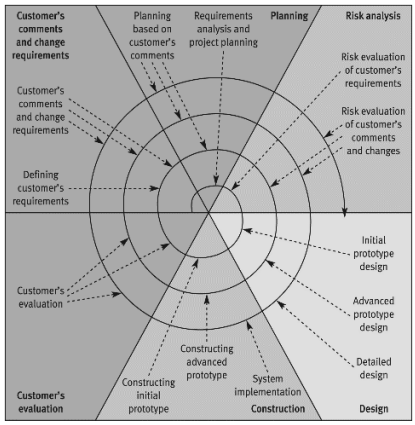


Fig.1.7 Win Win Spiral Model

customer with improved chances for changes; developer better chances to stay within budget and time. Done by increased emphasis on customer participation and on engineering activities. Extra sections in spiral dedicated to customer actions and developer engineering.

The **Win-Win Spiral Model** is a model of process based on **Theory W'** which is a management theory and approach "based on making winners of all the system's key stakeholder as necessary and sufficient for the project success."
"The original spiral model uses a cyclic approach to develop increasingly detailed elaborations of a software system's definition, culminating in incremental releases of the system's operational capability.
The win-win spiral process explicitly emphasizes continuous collaborative involvement of a software product's stakeholder in it's early definition and development stages.
Features of win-win spiral process model

- It provides an explicit set of goals for collaborative software definition and development.
- It embeds collaborative activities explicitly within a robust life-cycle model the spiral model.

The resulting process uses the "Theory W" of win-win model approach to converge on a system's next level objectives, constraints, and alternatives.
The win-win spiral model process uses two steps:
1. Identifying the system's stakeholders and their win conditions.
2. Reconciling win conditions through negotiations to arrive at a mutually satisfactory set of objectives, constraints, and alternatives for the next level.

There are following seven steps in a win-win spiral model.

Step 1. Identifying the next-level stakeholders.

- In this step you identify the next level stakeholders.
- These are the peoples who you need to make happy after the current phase of development.

Step 2. Identify stakeholder's win conditions

Step 3. Reconcile win conditions. Establish next level objectives, constraints, alternatives.

Step 4. Evaluate product and process alternatives, resolves the risks.

- In this step, the current project is examined carefully to consider alternatives and to resolves any risks that may have been found.
- If they have not been found but later cause problems, blame someone no longer on the project.

Step 5. Defining next level of product and process, Including partitions

- At this step, the next level is further defined and may partition the system into subsystem that can be developed in parallel cycles.
- This is particularly useful if a system can be broken into easy part and hard part because you can delegate the hard part of someone else.

Step 6. Validate Product and Process Definition

Step 7. Review Commitment

1.6.5 Incremental process model

Incremental process model is also know as Successive version model.

First, a simple working system implementing only a few basic features is built and then that is delivered to the customer. Then thereafter many successive iterations/ versions are implemented and delivered to the customer until the desired system is released.

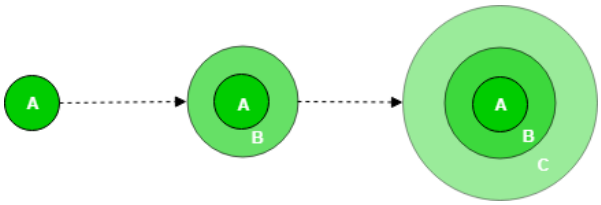


Fig.1.8 Incremental

A, B, C are modules of Software Product that are incrementally developed and delivered.

Life cycle activities

Requirements of Software are first broken down into several modules that can be incrementally constructed and delivered. At any time, the plan is made just for the next increment and not for any kind of long term plans. Therefore, it is easier to modify the version as per the need of the customer. Development Team first undertakes to develop core features (these do not need services from other features) of the system.

Once the core features are fully developed, then these are refined to increase levels of capabilities by adding new functions in Successive versions. Each incremental version is usually developed using an iterative waterfall model of development.

As each successive version of the software is constructed and delivered, now the feedback of the Customer is to be taken and these were then incorporated in the next version. Each version of the software has more additional features over the previous ones.

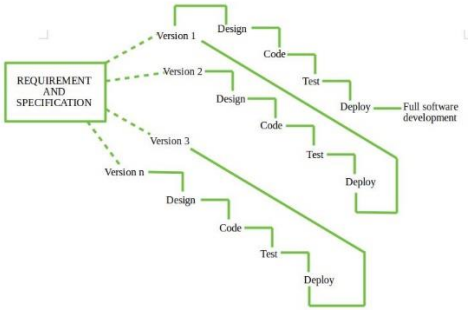


Fig.1.9 Life cycle activities

After Requirements gathering and specification, requirements are then spitted into several different versions starting with version-1, in each successive increment, next version is constructed and then deployed at the customer site. After the last version (version n), it is now deployed at the client site.

Types of Incremental model

`1. Staged Delivery Model – Construction of only one part of the project at a time.

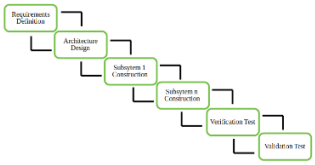


Fig.1.10 Staged Delivery Model

2. **Parallel Development Model –** Different subsystems are developed at the same time. It can decrease the calendar time needed for the development, i.e. TTM (Time to Market), if enough Resources are available.

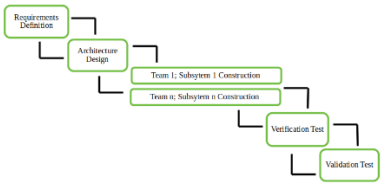


Fig.1.10 Parallel Development Model

When to use

1. Funding Schedule, Risk, Program Complexity, or need for early realization of benefits.
2. When Requirements are known up-front.
3. When Projects having lengthy developments schedules.
4. Projects with new Technology.

Advantages

Error Reduction (core modules are used by the customer from the beginning of the phase and then these are tested thoroughly)
Uses divide and conquer for breakdown of tasks.
Lowers initial delivery cost.
Incremental Resource Deployment.

Disadvantages

- Requires good planning and design.
- Total cost is not lower.
- Well defined module interfaces are required.

1.6.6 RAD Model

Rapid Application Development model relies on prototyping and rapid cycles of iterative development to speed up development and elicit early feedback from business users. After each iteration, developers can refine and validate the features with stakeholders. RAD model is also characterized by reiterative user testing and the re-use of software components. Hence, RAD has been instrumental in reducing the friction points in delivering successful enterprise applications.

WaveMaker makes use of the RAD model to provide a Rapid Application Development platform to create web and mobile applications. The following diagram depicts Wavemaker RAD platform architecture, based on the MVC (Model-View- Controller) pattern. Open standards, easy customization and rapid prototyping are central to the platform.

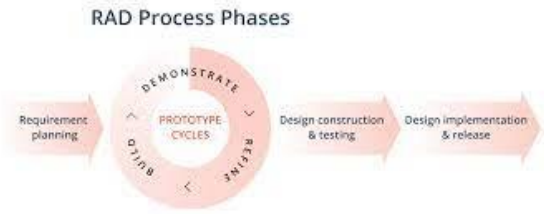


Fig1.11 RAD Process

Phases in RAD Model:

- ❖ Business Modeling
- ❖ Data Modeling
- ❖ Process Modeling
- ❖ Application Modeling

Testing and Turnover

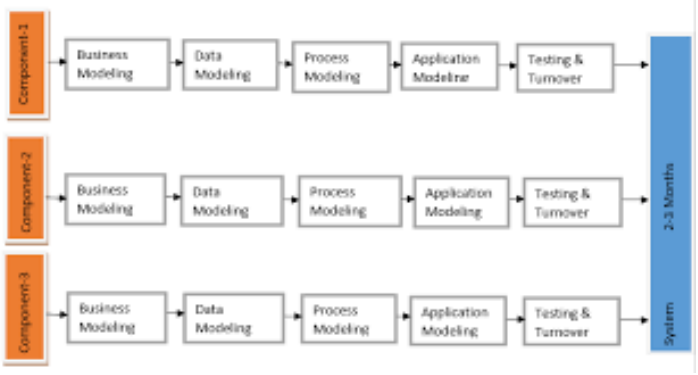


Fig.1.12. Testing and turnover

Business Modeling: In this phase of development business model should be designed based on the information available from different business activities. Before start the development there should be a complete picture of business process functionality.

Data Modeling: Once the business modeling phase over and all the business analysis completed, all the required and necessary data based on business analysis are identified in data modeling phase.

Process Modeling: All the data identified in data modeling phase are planned to process or implement the identified data to achieve the business functionality flow. In this phase all the data modification process is defined.

Application Modeling: In this phase application id developed and coding completed. With help of automation tools all data implemented and processed to work as real time. **Testing and turnover:** All the testing activates are performed to test the developed application.

Advantages of RAD Model:

- Fast application development and delivery.
- Lest testing activity required.
- Visualization of progress.
- Less resources required.
- Review by the client from the very beginning of development so very less chance to miss the requirements.
- Very flexible if any changes required.
- Cost effective.
- Good for small projects.

Disadvantages of RAD Model:

- High skilled resources required.
- On each development phase client's feedback required.
- Automated code generation is very costly.
- Difficult to manage.
- Not a good process for long term and big projects.

Proper modularization of project required.

1.6.7 Agile methods.

Agile software development methodology is a process for developing software (like other software development methodologies – Waterfall model, V-Model, Iterative model etc.) However, **Agile methodology** differs significantly from other methodologies. In English, Agile means 'ability to move quickly and easily' and responding swiftly to change – this is a key aspect of Agile software development as well.

Brief overview of Agile Methodology

- In traditional software development methodologies like Waterfall model, a project can take several months or years to complete and the customer may not get to see the end product until the completion of the project.

- At a high level, non-Agile projects allocate extensive periods of time for Requirements gathering, design, development, testing and UAT, before finally deploying the project.
- In contrast to this, Agile projects have Sprints or iterations which are shorter in duration (Sprints/iterations can vary from 2 weeks to 2 months) during which pre-determined features are developed and delivered.
- Agile projects can have one or more iterations and deliver the complete product at the end of the final iteration.

Example of Agile software development

Example: Google is working on project to come up with a competing product for MS Word, that provides all the features provided by MS Word and any other features requested by the marketing team. The final product needs to be ready in 10 months of time. Let us see how this project is executed in traditional and Agile methodologies.

In traditional Waterfall model –

- At a high level, the project teams would spend 15% of their time on gathering requirements and analysis (1.5 months)
- 20% of their time on design (2 months)
- 40% on coding (4 months) and unit testing
- 20% on System and Integration testing (2 months).
- At the end of this cycle, the project may also have 2 weeks of User Acceptance testing by marketing teams.
- In this approach, the customer does not get to see the end product until the end of the project, when it becomes too late to make significant changes.

project schedule in traditional software development.

With **Agile development** methodology –

- In the **Agile methodology**, each project is broken up into several 'Iterations'.
- All Iterations should be of the same time duration (between 2 to 8 weeks).
- At the end of each iteration, a working product should be delivered.
- In simple terms, in the Agile approach the project will be broken up into 10 releases (assuming each iteration is set to last 4 weeks).
- Rather than spending 1.5 months on requirements gathering, in Agile software development, the team will decide the basic core features that are required in the product and decide which of these features can be developed in the first iteration.
- Any remaining features that cannot be delivered in the first iteration will be taken up in the next iteration or subsequent iterations, based on priority.
- At the end of the first iterations, the team will deliver a working software with the features that were finalized for that iteration.

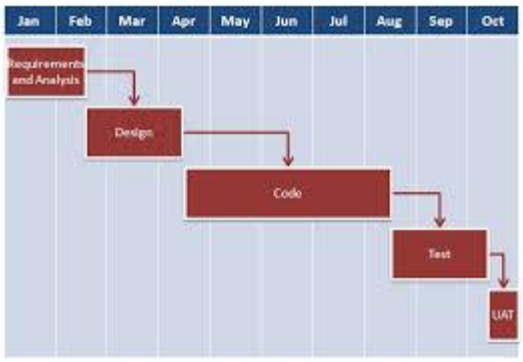


Fig.1.13 Iteration in Agile model

There will be 10 iterations and at the end of each iteration the customer is delivered a working software that is incrementally enhanced and updated with the features that were shortlisted for that iteration.

Fig.1.14 Agile development model

1.6.8 Component Based Model (CBM)

The component-based assembly model uses object-oriented technologies. In object-oriented technologies, the emphasis is on the creation of classes. Classes are the entities that encapsulate data and algorithms.

In component-based architecture, classes (i.e., components required to build application) can be uses as reusable components. This model uses various characteristics of spiral model. This model is evolutionary by nature. Hence, software development can be done using iterative approach. In CBD model, multiple classes can be used. These classes are basically the prepackaged components.

The model works in following manner:

Step-1:

First identify all the required candidate components, i.e., classes with the help of application data and algorithms.

Step-2:

If these candidate components are used in previous software projects then they must be present in the library.

Step-3:

Such preexisting components can be excited from the library and used for further development.

Step-4:

But if the required component is not present in the library, then build or create the component as per requirement.

Step-5:

Place this newly created component in the library. This makes one iteration of the system.

Step-6:

Repeat steps 1 to 5 for creating n iterations, where n denotes the number of iterations required to develop the complete application.

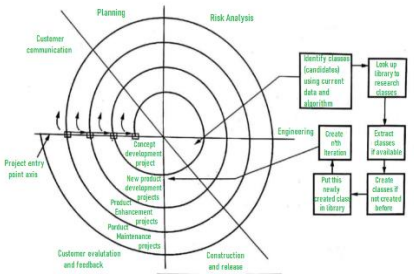


Fig.1.15 Win win spiral model

1.7 Software Development Life Cycle

Software Development Life Cycle is the application of standard business practices to building software applications. It's typically divided into six to eight steps: Planning, Requirements, Design, Build, Document, Test, Deploy, Maintain. Some project managers will combine, split, or omit steps, depending on the project's scope. These are the core components recommended for all software development projects.

SDLC is a way to measure and improve the development process. It allows a fine-grain analysis of each step of the process. This, in turn, helps companies maximize efficiency at each stage. As computing power increases, it places a higher demand on software and developers. Companies must reduce costs, deliver software faster, and meet or exceed their customers' needs. SDLC helps achieve these goals by identifying inefficiencies and higher costs and fixing them to run smoothly.

1. **Planning**

In the Planning phase, project leaders evaluate the terms of the project. This includes calculating labor and material costs, creating a timetable with target goals, and creating the project's teams and leadership structure.

Planning can also include feedback from stakeholders. Stakeholders are anyone who stands to benefit from the application. Try to get feedback from potential customers, developers, subject matter experts, and sales reps.

Planning should clearly define the scope and purpose of the application. It plots the course and provisions the team to effectively create the software. It also sets boundaries to help keep the project from expanding or shifting from its original purpose.

2. Define Requirements

Defining requirements is considered part of planning to determine what the application is supposed to do and its requirements. For example, a social media application would require the ability to connect with a friend. An inventory program might require a search feature.

Requirements also include defining the resources needed to build the project. For example, a team might develop software to control a custom manufacturing machine. The machine is a requirement in the process.

3. Design and Prototyping

The Design phase models the way a software application will work. Some aspects of the design include:

Architecture – Specifies programming language, industry practices, overall design, and use of any templates or boilerplate
User Interface – Defines the ways customers interact with the software, and how the software responds to input
Platforms – Defines the platforms on which the software will run, such as Apple, Android, Windows version, Linux, or even gaming consoles
Programming – Not just the programming language, but including methods of solving problems and performing tasks in the application
Communications – Defines the methods that the application can communicate with other assets, such as a central server or other instances of the application
Security – Defines the measures taken to secure the application, and may include SSL traffic encryption, password protection, and secure storage of user credentials

Prototyping can be a part of the Design phase. A prototype is like one of the early versions of software in the Iterative software development model. It demonstrates a basic idea of how the application looks and works. This "hands-on" design can be shown to stakeholders. Use feedback o improve the application. It's less expensive to change the Prototype phase than to rewrite code to make a change in the Development phase.

4. **Software Development**

This is the actual writing of the program. A small project might be written by a single developer, while a large project might be broken up and worked by several teams. Use an Access Control or Source Code Management application in this phase. These systems help developers track changes to the code. They also help ensure compatibility between different team projects and to make sure target goals are being met.

The coding process includes many other tasks. Many developers need to brush up on skills or work as a team. Finding and fixing errors and glitches is critical. Tasks often hold up the development process, such as waiting for test results or compiling code so an application can run. SDLC can anticipate these delays so that developers can be tasked with other duties.

Software developers appreciate instructions and explanations. Documentation can be a formal process, including wiring a user guide for the application. It can also be informal, like comments in the source code that explain why a developer used a certain procedure. Even companies that strive to create software that's easy and intuitive benefit from the documentation.

Documentation can be a quick guided tour of the application's basic features that display on the first launch. It can be video tutorials for complex tasks. Written documentation like user guides, troubleshooting guides, and FAQ's help users solve problems or technical questions.

5. **Testing**

It's critical to test an application before making it available to users. Much of the testing can be automated, like security testing. Other testing can only be done in a specific environment – consider creating a simulated production environment for complex deployments. Testing should ensure that each function works correctly. Different parts of the application should also be tested to work seamlessly together—performance test, to reduce any hangs or lags in processing. The testing phase helps reduce the number of bugs and glitches that users encounter. This leads to a higher user satisfaction and a better usage rate.

6. **Deployment**

In the deployment phase, the application is made available to users. Many companies prefer to automate the deployment phase. This can be as simple as a payment portal and download link on the company website. It could also be downloading an application on a smartphone.

Deployment can also be complex. Upgrading a company-wide database to a newly-developed application is one example. Because there are several other systems used by the database, integrating the upgrade can take more time and effort.

7. **Operations and Maintenance**

At this point, the development cycle is almost finished. The application is done and being used in the field. The Operation and Maintenance phase is still important, though. In this phase, users discover bugs that weren't found during testing. These errors need to be resolved, which can spawn new development cycles.

In addition to bug fixes, models like Iterative development plan additional features in future releases. For each new release, a new Development Cycle can be launched.

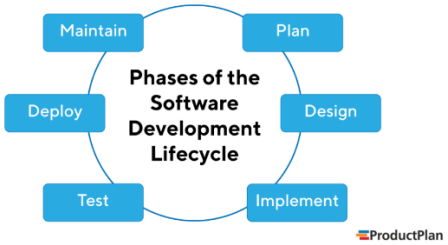


Fig.1.16 Phase in Software lifecycle model

REFERENCE

1. Huges. B, Cottorell M, Rajib M, Software Project Management McGraw Hill, 5th Edition, 2017.
2. Roger S Pressman, Software Engineering, McGraw Hill, 8th Edition, 2019
3. https://www.javatpoint.com/software-engineering-sdlc-models

Question Format:

Part-A			
Q.No	**Questions**	**Competence**	**BT Level**
1.	Define software process model	Knowledge	BTL 1
2.	List out the phases in software process model	Remember	BTL 1
3.	List various software process models.	Remember	BTL 1
4.	Illustrate the RAD development model.	Analysis	BTL 4
5.	Write down the major aims of the RAD model.	Understand	BTL 2
6.	Appraise the advantages of the RAD model	Evaluate	BTL 5
7.	Categorize the task regions in the Spiral model	Analysis	BTL 4
8.	Summarize the characteristics of SRS	Understand	BTL2
9.	Define Tailoring	Knowledge	BTL2
10.	Compare few differences between RAD Model vs Traditional SDLC	Understand	BTL2

Part-B			
Q.No	**Questions**	**Competence**	**BT Level**
1.	Illustrate software process and process models.	Analyze	BTL 4
2.	Summarize about Rapid Application development.	Understand	BTL 2
3.	Criticize the software development model where the condition becomes both the user and developer "wins"	Evaluate	BT5
4.	Write note on Software Development Life Cycle	Create	BT6
5.	Identify and explain the software development life cycle model that is chosen if the development team has less experience on similar projects.	Analyze	BT 4
6.	Rewrite the phases involved in software development model	Create	BT 6
7.	Summarize a note on software Requirement Specification (SRS)	Understand	BT 6

UNIT – II - SOFTWARE DEVELOPMENT – SBAA3017

UNIT 2 SOFTWARE DEVELOPMENT

Software Development Team - Three Vital Aspects of Software Project Management - The Team - Meaning of Leadership - Communicating in Harmony - Personality traits - Project Organizations. Project Planning: Top-Down and Bottom-Up Planning - Types of Activity - Project Duration : Schedule Monitoring Tools - Gantt Chart, PERT Chart, Critical Path.

2. Software Development Team

- Product owner
- Project manager
- UX / UI designers
- Business analyst
- Software developers
- Team lead / Tech lead
- Scrum master

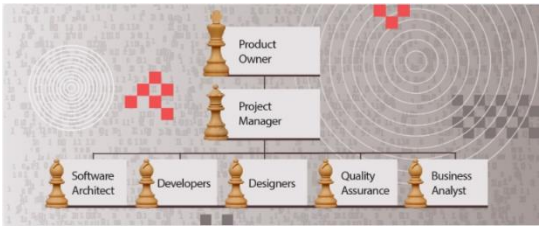


Fig.2.1 Software Development Team

- **Product owner**, in the case of an outsourced project, this is the client with a vision of how the end-product should look, who are the end-users and what it should do.
- **Project manager** is a person responsible for managing and leading the whole team. Their role is to efficiently optimize the work of the team, ensure the product is meeting the requirements and identify the goals for the team.
- **Software architect** is a highly-skilled software developer that has to think through all the aspects of the project and is responsible for making high level design choices, as well as select technical standards (for instance, determines the technology stack to use).
- **Developers** or product engineers are team members that apply their knowledge of engineering and programming languages in software development.
- **Experience designers** ensure that the product is easy and pleasant to use. They conduct user interviews, market research, and design a product with end-users in mind.
- **QA** or tester is responsible for the Quality Assurance and makes sure the product is ready to use.
- **Business Analyst**'s role is to uncover the ways to improve the product. They interact with stakeholders to understand their problems and needs, and later document and analyze them to find a solution.

2.1 5 Steps to Building an Effective Software Development Team

Define the kind of development team type that fits for your project

First things first, before we dive deeper into what pay attention to when building a development team, you need to decide on the kind of team you want to create. Establishing a clear software development team structure is an important first step into an overall success of your project.

- Here you have to choose from the three main types:

Generalists

They are Jack of all trades, so to say. Generalists possess a broad range of knowledge and expertise. Generally, these types of teams are designed to handle end-to-end solutions. The advantage of generalists is in the fact that they can provide a complete solution to the problem. However, they also have some drawbacks — if your project requires a higher level of expertise in some area, generalists will find themselves at a loss as they may lack the knowledge and skills.

Specialists

Unlike generalists, this type of teams consists of members that are highly skilled in a particular field. The advantage of specialists is clear — they can address a specific matter with all their knowledge and expertise, resulting in more efficient and effective work. On the other hand, communication is not exactly a forte of this type of software development team. Often times being a very narrow specialist, team members may lack general understanding of what are the roles of other team members, and thus making communication between them somewhat ineffective.

Hybrid team

As you've probably guessed, this type of team is a mix of the previous two. It seems like this approach combines the best from the two worlds, where specialists focus on functional parts, and generalists are responsible for the communication and cooperation inside the team. This dream of a team, however, comes with a constraint — usually this type of software development team is more difficult to gather in terms of time and financial resources. So which one is it? Well, that's for you to decide. Ideally, strive for the balance of generalists and specialists within your team to achieve better results.

Decide on the software development team size

- Now that you've established what kind of team you want to build, let's talk about the size. There really isn't a magical number when it comes to the size of your software development team. Smaller teams are easier to manage on the one hand, but in this case, every team member plays a crucial role in the project and losing even one person can have a significant impact on the overall result. With bigger teams the challenge is in managing the communication.
- Complexity of your project
- Budget
- Deadline
- Available resources

Establish clear roles and goals

Now this one seems like an obvious one — the roles inside your team are clear. There are designers, developers, and probably a tester, right? Yes but no. The roles of an effective software development team are more versatile and complex than that.

2.2 A Leadership Story

- A group of workers and their leaders are set a task of clearing a road through a dense jungle on a remote island to get to the coast where an estuary provides a perfect site for a port.
- The leaders organise the labour into efficient units and monitor the distribution and use of capital assets – progress is excellent. The leaders continue to monitor and evaluate progress, adjusting along the way to ensure the progress is maintained and efficiency increased wherever possible.
- Then, one day amidst all the hustle and bustle and activity, one person climbs up a nearby tree. The person surveys the scene from the top of the tree.
- And shouts down to the assembled group below…
- "Wrong Way!"
- (Story adapted from Stephen Covey (2004) "The Seven Habits of Highly Effective People" Simon & Schuster).
- "Management is doing things right, leadership is doing the right things" (Warren Bennis and Peter Drucker)

2.2.1 Leadership

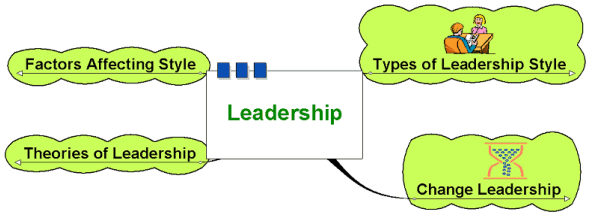


Fig.2.2 Parameters in leadership

2.2.2 Personality Traits

In psychology, the Big Five personality traits are five broad domains or dimensions of personality that are used to describe human personality. The theory based on the Big Five factors is called the five-factor model (FFM).

2.2.3 Types of Leadership Style

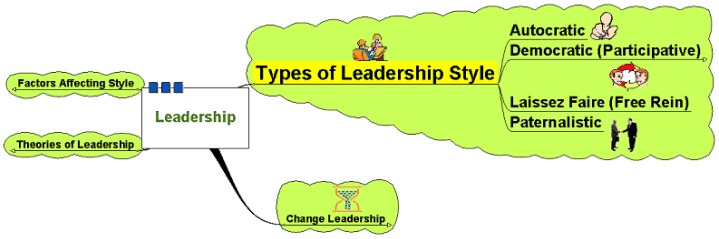


Fig.2.3 Types of leadership styles

- **Autocratic:**
    - Leader makes decisions without reference to anyone else
    - High degree of dependency on the leader
    - Can create de-motivation and alienation of staff
    - May be valuable in some types of business where decisions need to be made quickly and decisively
- **Democratic:**
- Encourages decision making from different perspectives – leadership may be emphasised throughout the organisation
    - Consultative: process of consultation before decisions are taken
    - Persuasive: Leader takes decision and seeks to persuade others that the decision is correct
    - May help motivation and involvement
    - Workers feel ownership of the firm and its ideas
    - Improves the sharing of ideas and experiences within the business
    - Can delay decision making
- **Laissez-Faire:**
- 'Let it be' – the leadership responsibilities are shared by all
- Can be very useful in businesses where creative ideas are important
- Can be highly motivational, as people have control over their working life
- Can make coordination and decision making time-consuming and lacking in overall direction
- Relies on good teamwork and good interpersonal relations

Paternalistic:

- Leader acts as a 'father figure'
- Paternalistic leader makes decision but may consult
- Believes in the need to support staff

2.2.4 Change Leadership

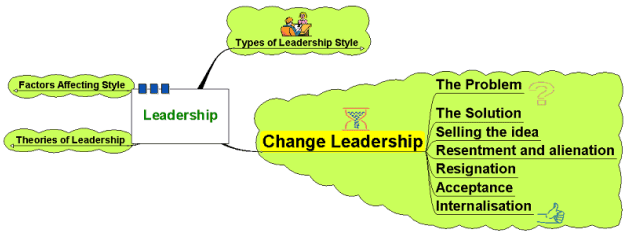


Fig.2.4 Change Leadership

- The most challenging aspect of business is leading and managing change
- The business environment is subject to fast-paced economic and social change
- Modern business must adapt
  and be flexible to survive
- Problems in leading change stem mainly from human resource management
- Leaders need to be aware of how change impacts on workers:
- Series of self-esteem states identified by Adams et al and cited by Garrett

2.2.5 Theories of Leadership

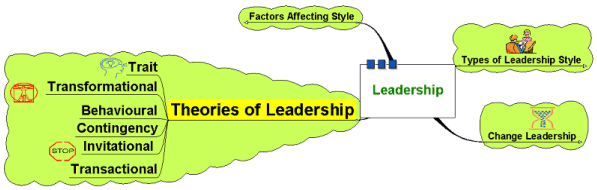


Fig2.5 Theories of Leadership

Trait theories

- **Is there a set of characteristics that determine a good leader?**
  - Personality?
  - Dominance and personal presence?
  - Charisma?
  - Self-confidence?
  - Achievement?
  - Ability to formulate a clear vision?

Theories of Leadership

- Trait theories:

- Is there a set of characteristics
  that determine a good leader?
    - Personality?
    - Dominance and personal presence?
    - Charisma?
    - Self confidence?
    - Achievement?
    - Ability to formulate a clear vision?
    - Are such characteristics
      inherently gender biased?
    - Do such characteristics
      produce good leaders?
    - Is leadership more than
      just bringing about change?
    - Does this imply that leaders are born not bred?
- Behavioural:
- Imply that leaders can be trained – focus on the way of doing things
    - Structure based behavioural theories – focus on the leader instituting structures – task orientated

Relationship based behavioural theories – focus on the development and maintenance of relationships – process orientated

Contingency Theories

- Leadership as being more flexible – different leadership styles used at different times depending on the circumstance.
- Suggests leadership is not a fixed series of characteristics that can be transposed into different contexts

May depend on

- Type of staff
- History of the business
- Culture of the business
- Quality of the relationships
- Nature of the changes needed
- Accepted norms within the institution

- Transformational:
    - to a business or organisation
- Requires:
    - Long term strategic planning
    - Clear objectives
    - Clear vision
    - Leading by example – walk the walk
    - Efficiency of systems and processes
- Invitational Leadership:
    - Improving the atmosphere and message sent out by the organisation
    - Focus on reducing negative messages
      sent out through the everyday actions of the business both externally and, crucially, internally
    - Review internal processes to reduce these
    - Build relationships and sense of belonging and identity with the organisation – that gets communicated to customers, etc.
- Transactional Theories:
    - Focus on the management
      of the organisation
    - Focus on procedures and efficiency
    - Focus on working to rules
      and contracts
    - Managing current issues
      and problems

Factors Affecting Style

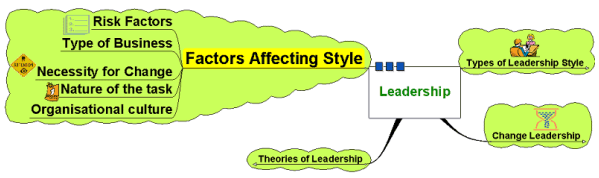


Fig. 2.6 Factors Affecting Style

Leadership style may be dependent on various factors:

- based on degree of risk involved

- Type of business – creative business
  or supply driven?
- How important change is –
  change for change's sake?
- Organisational culture – may be long embedded
  and difficult to change
- Nature of the task – needing cooperation? Direction? Structure?

2.3 Personality Traits

- In psychology, the Big Five personality traits are five broad domains or dimensions of personality that are used to describe human personality. The theory based on the Big Five factors is called the five-factor model (FFM).

These are called the "big five"

- Openness to Experience
- Conscientiousness
- Extraversion
- Agreeableness

  Neuroticism

How it works:

- The idea is that you get a positive or negative score in each of the five categoriesThese positive or negative scores allow you to understand your personality traits and perhaps your parenting style

Openness to Experience

- Loves new experiences and spontaneity
- Tolerates and explores the unfamiliar

Conscientiousness Describes how an individual performs tasks

- Careful,
- hardworking,
- neat,
- organized, etc.

Conscientiousness High Scorers Organized Reliable Hard-working

- Self-disciplined
- Punctual
- Persevering Low Scorers

- Aimless
- Unreliable
- Careless
- Negligent
- Weak-willed Lazy

Extraversion

- Describes interpersonal behavior and how much you prefer to be alone or with othersAlso linked to your ability to experience positive emotions or “get pumped”

2.4 PROJECT ORGANIZATION

- The main objective of Project organization is to establish the relationship among:
- The work to be done
- The people doing
- The work
- the work place(s)

A project organization is a structure that facilitates the coordination and implementation of project activities. Its main reason is to create an environment that fosters interactions among the team members with a minimum amount of disruptions, overlaps and conflict. One of the important decisions of project management is the form of organizational structure that will be used for the project.

Each project has its unique characteristics, and the design of an organizational structure should consider the organizational environment, the project characteristics in which it will operate, and the level of authority the project manager is given. A project structure can take on various forms with each form having its own advantages and disadvantages. One of the main objectives of the structure is to reduce uncertainty and confusion that typically occurs at the project initiation phase. The structure defines the relationships among members of the project management and the relationships with the external environment. The structure defines the authority by means of a graphical illustration called an organization chart.

A properly designed project organization chart is essential to project success. An organization chart shows where each person is placed in the project structure. An organization chart is drawn in pyramid form where individuals located closer to the top of the pyramid have more authority and responsibility than members located toward the bottom. It is the relative locations of the individuals on the organization chart that specifies the working relationships, and the lines connecting the boxes designate formal supervision and lines of communication between the individuals.

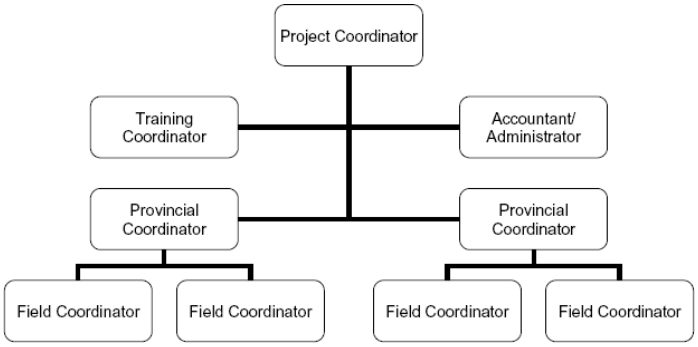


Fig 2.7 Project Organization

Creating the project structure is only a part of organizing the project; it is the actual implementation and application that takes the most effort. The project organization chart establishes the formal relationships among project manager, the project team members, the development organization, the project, beneficiaries and other project stakeholders. This organization must facilitate an effective interaction and integration among all the major project participants and achieve open and effective communication among them.

The project manager must create a project structure that will meet the various project needs at different phases of the project. The structure cannot be designed too rigid or too lose, since the project organization's purpose is to facilitate the interaction of people to achieve the project ultimate goals within the specified constraints of scope, schedule, budget and quality. The objective in designing a project structure is to provide a formal environment that the project manager can use to influence team members to do their best in completing their assignment and duties. The structure needs to be designed to help develop collaboration among individual team members; all in a cost-effective way with a minimum of duplication of effort and overlaps.

The organization chart has a limited functionality; it only shows the hierarchical relationship among the team members but does not shows how the project organization will work, it is for that reason that the design should consider factors that will facilitate the operation of the structure; these include communications, information flows, coordination and collaboration among its members.

2.5 FACTORS IN DESIGNING A PROJECT STRUCTURE

There are two design factors that significantly influence the process of developing a project management structure. These are the level of specialization, and the need for coordination. The project manager should consider these factors at the moment of designing the project organization in order to maximize the effectiveness of the structure.

Specialization affects the project structure by the degree of specialty in technical areas or development focus; projects can be highly specialized and focus on a specific area of development or have different broad specializations in many areas of development. For large projects that have multiple specializations or technical areas, each area may have a different need; from differences in goals, approaches and methodologies, all of which influence the way the project will implement its activities. A project that has two components, a reconstruction and education, will need to manage different approaches based on the specialization of each one. In the education component, the needs is for a structure more open and informal, where the time horizon is longer, with more emphasis on sharing and generation of new ideas in order to achieve innovation and creativity. In a reconstruction component, there are specific goals, a

need for a rigid, hierarchical structure, and there is a defined time horizon with little sharing of ideas. While specialization allows each project component to maximize their productivity to attain their departmental goals, the dissimilarities may lead to conflict among the members or leads of each component. In general, the greater the differences, the more problems project managers have in getting them to work together.

Coordination is required to bring unity to the various elements that make up a project. The project work is organized around a work breakdown structure (WBS) that divides the overall project goals into specific activities or tasks for each project area or component; the project manager must design an organizational structure that ensure that the various components are integrated so that their efforts contribute to the overall project goal. Integration is the degree of collaboration and mutual understanding required among the various project components to achieve project goals. Most projects are characterized by the division of labor and task interdependencies, creating the need for integration to meet project objectives. This need is greatest when there are many project components that have different specializations. The goal of the project management structure is the achievement of harmony of individual efforts toward the accomplishment of the group goals. The project manager's principal responsibility is to develop integrating strategies to ensure that a particular component or activity is organized in a way that all of the components, parts, subsystems, and organizational units fit together as a functioning, integrated whole according to the project master plan.

2.5.1 TYPES OF PROJECT ORGANIZATIONS STRUCTURES

The several factors to consider when deciding on the design of project organizational structures, especially within an existing organization, the factor that has a significant is the extent of authority and responsibility top management is prepared to delegate to the project manager. An important function of the organizations' top management is to design an organization that fully supports project management. This is done by redesigning the organization to emphasize the nature of the projects and adapting how roles and responsibilities are assigned.

The organization needs to define the project manager's job, degree of authority and autonomy, and relationship to both the organization, other projects and to other units in the organization. Upper management also should specify communication channels, methods of conflict resolution between the project and the rest of the organization. Development organizations are usually organized around programmatic focus areas such as health or education. These areas are usually called program units and are centered on a specific development field. In this environment a project has three organization structures available for design and all are defined by the level of organizational authority given to the project manager:

- **Programmatic** based, in which project managers have authority only within the program focus or area
- **Matrix based**, in which the project manager shares responsibility with other program unit managers
- **Project based**, in which project managers have total authority.

Programmatic Based

The programmatic focus refers to a traditional structure in which program sector managers have formal authority over most resources. It is only suitable for projects within one program sector. However, it is not suitable for projects that require a diverse mix of people with different expertise from various program sectors. In a programmatic based organization, a project team

is staffed with people from the same area. All the resources needed for the project team come from the same unit. For instance, if the project is related to the health area, the project resources come from the health unit.

The most obvious advantage of programmatic based projects is that there are clear lines of authority, in large projects the project managers tend to also be the program unit manager. There is no need to negotiate with other program units for resources, since all

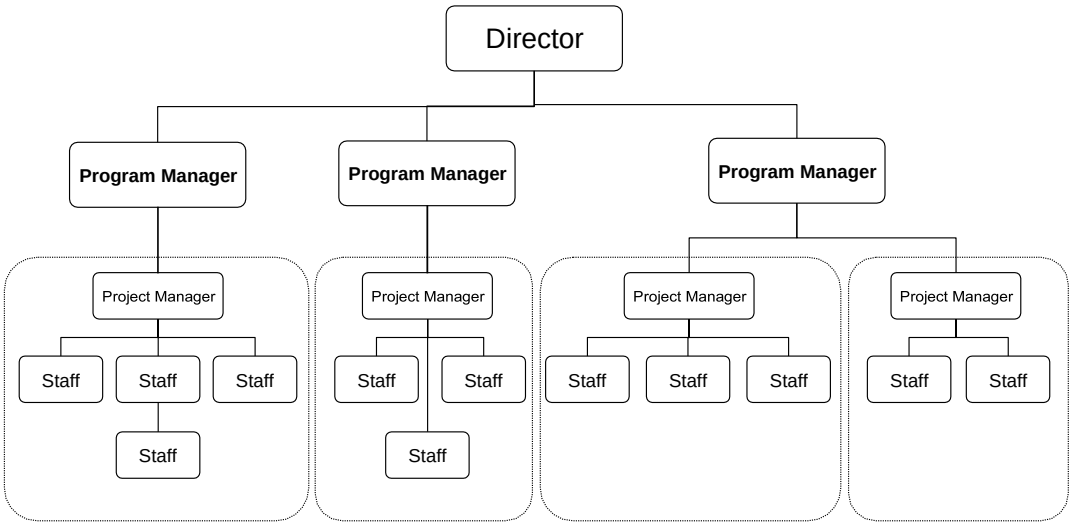


Fig.2.8 Programmatic organization

of the staff needed for the project will come from the same program area. Another advantage of this type of organization is that the team members are usually familiar with each other, since they all work in the same area. The team members also tend to bring applicable knowledge of the project.

A major disadvantage of the programmatic based organization is that the program area may not have all of the specialists needed to work on a project. A nutrition project with a water component, for instance, may have difficulty acquiring specialty resources such as civil engineers, since the only people available will work in their own program unit. Another disadvantage is that project team members may have other responsibilities in the program unit since they may not be needed full- time on a project. They may be assigned to other projects, but it is more typical that they would have support responsibilities that could impact their ability to meet project deadlines.

Matrix Based

Matrix based project organizations allow program units to focus on their specific technical competencies and allow projects to be staffed with specialists from throughout the organization. For instance, nutrition specialists may report to one program unit, but would be allocated out to work on various projects. A health specialist might report to the health unit, but be temporarily assigned to a project in another project that needs health expertise. It is common for people to report to one person in the programmatic unit, while working for one or two project managers from other projects in different programmatic units.

The main advantage of the matrix based organization is the efficient allocation of all resources, especially scarce specialty skills that cannot be fully utilized by only one

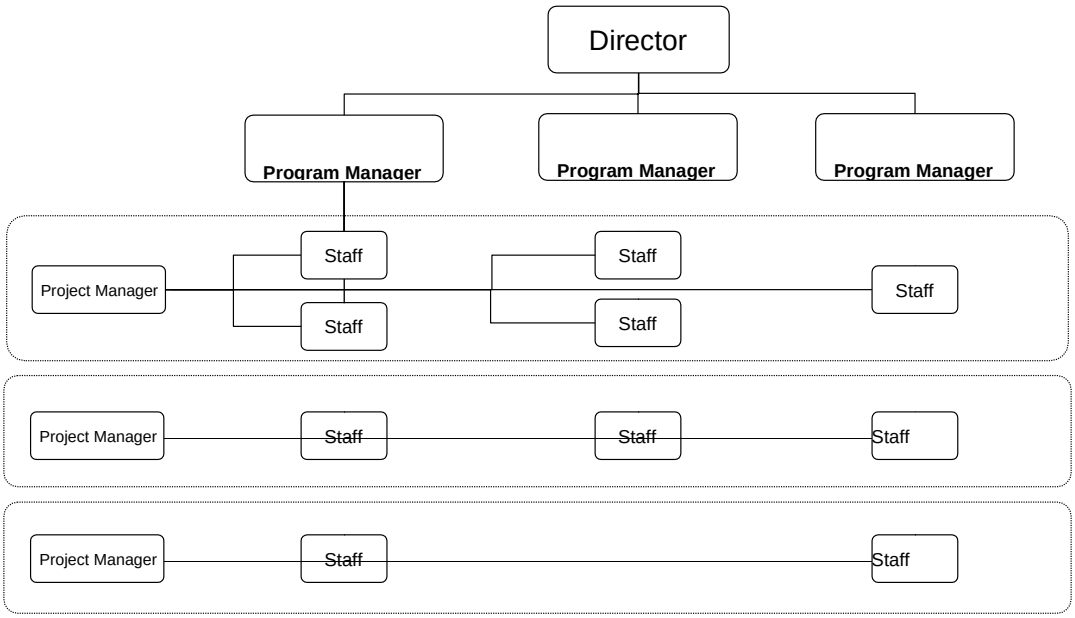


Fig 2.9 Matrix based organization

project. For instance, monitoring and evaluation specialists may not be utilized full-time on a project, but can be fully leveraged by working on multiple projects.

The matrix based organization is also the most flexible when dealing with changing programmatic needs and priorities. Additional advantages to matrix management are: it allows team members to share information more readily across the unit boundaries, allows for specialization that can increase depth of knowledge and allow professional development and career progression to be managed. It is easier for a program unit manager to loan an employee to another manager without making the change permanent. It is therefore easier to accomplish work objectives in an environment when task loads are shifting rapidly between programmatic units.

The main disadvantage is that the reporting relationships are complex. Some people might report to programmatic unit managers for whom little work is done, while actually working for one or more project managers. It becomes more important for staff members to develop strong time management skills to ensure that they fulfill the work expectations of multiple managers. This organization also requires communication and cooperation between multiple programmatic unit managers and project managers since that all be competing for time from the same resources.

Matrix management can put some difficulty on project managers because they must work

closely with other managers and workers in order to complete the project. The programmatic managers may have different goals, objectives, and priorities than the project managers, and these would have to be addressed in order to get the job done. An approach to help solve this situation is a variation of the Matrix organization which includes a coordinating role that either supervises or provides support to the project managers. In some organizations this is know as the Project Management Office (PMO), dedicated to provide expertise, best practices, training, methodologies and guidance to project managers.

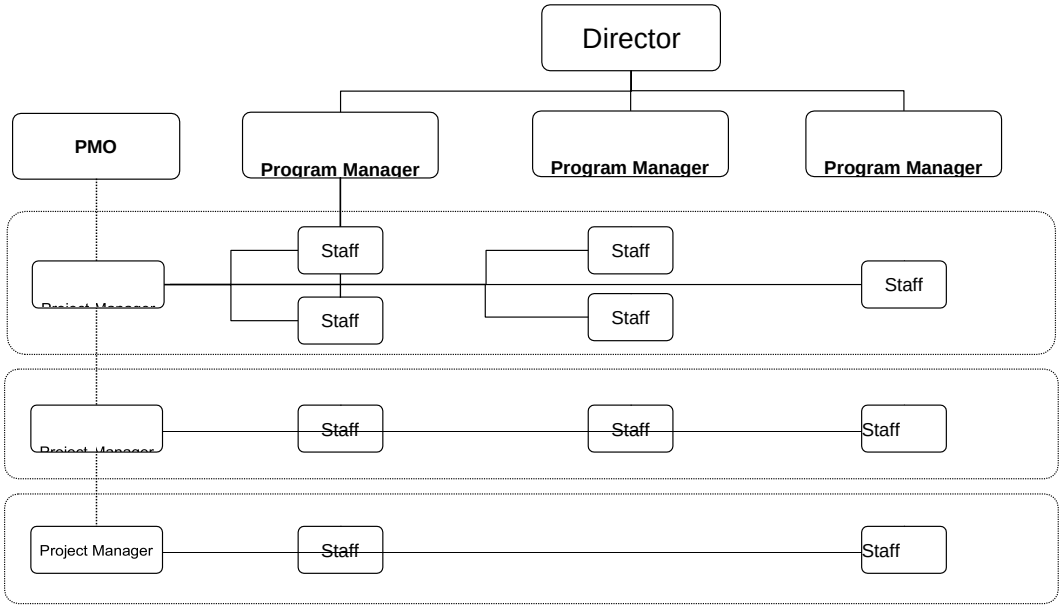


Fig. 2.10 Project management office

The PMO unit also defines and maintains the standards of project management processes within the organization. The PMO strives to standardize and introduce economies of scale in the implementation of projects. The PMO is the source of documentation, guidance and metrics on the practice of project management and implementation. The PMO can also help in the prioritization of human resources assigned to projects.

Project Based

In this type of organization project managers have a high level of authority to manage and control the project resources. The project manager in this structure has total authority over the project and can acquire resources needed to accomplish project objectives from within or outside the parent organization, subject only to the scope, quality, and budget constraints identified in the project.

In the project-based structure, personnel are specifically assigned to the project and report directly to the project manager. The project manager is responsible for the performance appraisal and career progression of all project team members while on the project. This leads to increased project loyalty. Complete line authority over project efforts affords the project manager strong project controls and centralized lines of communication. This leads to rapid reaction time and improved responsiveness. Moreover, project personnel are retained on an exclusive rather than shared or part-time basis. Project teams develop a strong sense of project identification and ownership, with deep loyalty efforts to the project and a good understanding of the nature of project's activities, mission, or goals.

Pure project-based organizations are more common among large and complicated projects. These large projects can absorb the cost of maintaining an organization whose structure has some duplication of effort and the less than cost-efficient use of resources. In fact, one major disadvantage of the project-based organization is the costly and inefficient use of personnel. Project team members are generally dedicated to one project at a time, even though they may rarely be needed on a full-time basis over the life cycle of the project. Project managers may tend to retain their key personnel long after the work is completed, preventing their contribution to other projects and their professional development.

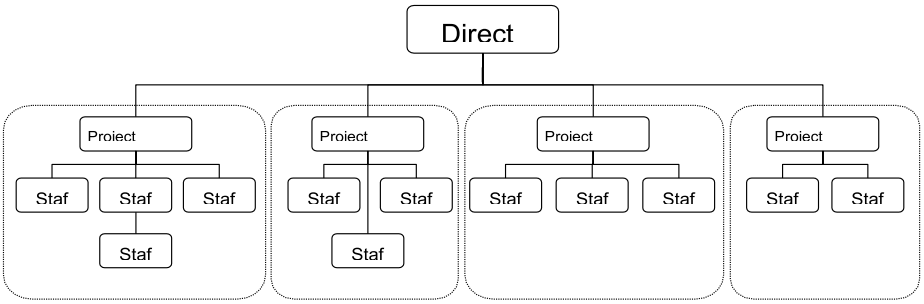


Fig.2.10 project-based organizations

In this type of organization, limited opportunities exist for knowledge sharing between projects, and that is a frequent complaint among team members concerning the lack of career continuity and opportunities for professional growth. In some cases, project personnel may experience a great deal of uncertainty, as organizations or donor's priorities shift or the close of the project seems imminent.

One disadvantage is duplication of resources since scarce resources must be duplicated on different projects. There can also be concerns about how to reallocate people and resources when projects are completed. In a programmatic focus organization, the people still have jobs within the program unit. In a project-based organization it is not always clear where everyone is reassigned when the project is completed. Another disadvantage is that resources may not be needed as a full time for the entire length of the project, increasing the need to manage short term contracts with consultants and other subject matter experts.

A variety of this pure project approach is temporarily project-based organizations. This organization consists of a project team pulled together temporarily from their program unit and led by a project manager that does not report to a programmatic unit. The project manager has the full authority and supervision of the project team.
Another design is based on a mixed structure that includes a matrix, programmatic focus and project based; this mix reflects the need for more flexibility in a development organization to accommodate different requirements. For example a health program may have a couple of projects short term and long term all reporting to the program manager. An education project may be organized on a matrix using resources part-time from other units, and a large water project organized as a fully project-based were all staff report to the project manager. It is not unusual to find this type of mixed designs on development organizations.

2.6 PROJECT GOVERNANCE

Project governance can be defined as an organization's overall process for sharing decision rights about projects and monitoring the performance of project interventions. All development

organizations have some form of project governance. Those with effective governance have actively designed a set of project governance mechanisms (committees, budgeting processes, approvals, etc.) that encourage behavior consistent with the organization's mission, strategy, values, norms, and culture.

The objective of project governance is to establish clear levels or authority and decision making including the planning, influencing and conducting of the policy and affairs of the project. It involves the people, policies and processes that provide the framework within which project managers make decisions and take actions to optimize outcomes related to their areas of responsibility. This is achieved by defining and identifying the roles, responsibilities and accountability of all people involved in a project, including their interaction and level of coordination with internal and external dependencies.

The organization's management team is responsible for setting up and supporting the governance structure before the project initiates its activities to ensure that all key decisions are made at the right time. The management team defines the project governance in a document that outlays the roles and responsibilities for decision making in the project team and stakeholders; this may include the creation of a project committee and its high-level operating rules. A good project governance document helps projects by defining the procedures to follow escalation of issues, defines the decision-making structure, roles and responsibilities of each key stakeholder about the different processes in the project from communications to budget change authorizations.

Some development organizations may choose the have a standing project governance committee that oversees all projects and defines the decision-making structure for each project. In this case the committee may include decisions about project proposal approvals and supervision of the organizations' project portfolio in general, including the selection or appointment of project managers.

2.7 Project Planning Bottom-up vs. Top-down

A Software Project is the complete methodology of programming advancement from requirement gathering to testing and support, completed by the execution procedures, in a specified period to achieve intended software product.
When it comes to projects, there are two schools of thought

- Top-down
- bottom-up

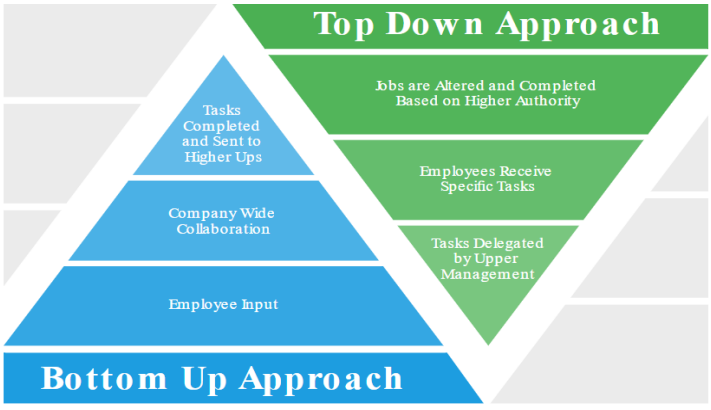


Fig 2.11 top down and bottom up approach

Top-down approach

with the final deliverable (project goal) and break it down into smaller, more manageable tasks. These tasks can be further broken down into subtasks — great details — and then assigned to individual teams and/or team members within that team.

Example: You need to update the FAQ section on your website. This is a task your team has completed in the past. The scope is clear, the timeline defined, the subtasks straightforward. A perfect time for a simple, top-down approach.Top-down tends to be more autocratic (we are telling you what to do, now go do it).

Bottom-up approach

The bottom-up approach to project management means that you begin with brainstorming possible solutions to meet that final deliverable. In other words, you know what the project goal is, but are not sure (yet) how to get there. A bottom-up approach involves all members of the team working together to determine the necessary tasks to reach that final end product.

Example: You are embarking on an entirely new product based on feedback from your customers. You need input from the entire team as this is a process you've never been through before. A bottom-up approach works best in this situation.

Bottom-up tends to be more democratic (we are not sure how to do it, but as a team, we'll figure it out).

In this blog post, we'll review different styles of management/leadership, discuss the pros and cons of top-down vs. bottom-up for project management, and share how monday.com can help navigate whatever approach you choose!

What are top-down/bottom-up leadership styles?

When it comes to *management and leadership styles*, there are also two different approaches. Again, surprise surprise, top-down vs. bottom-up. At monday.com, we are strong advocates for a top-down approach to managing and leading. In fact, leading through a bottom-up approach is one of the 5 pitfalls first-time (and sometimes more experienced) managers tend to make.

We suggest including a real-time "stand-up" meeting as one of a manager's daily tasks. These quick, specific meetings provide a high-level update on the plan, checking in with each team member to ensure they are on track to accomplishing their goals — understanding roadblocks, what's important, and what's not needed to meet the end goal.

If one approaches leading through a bottom-up style, it becomes muddy real quick. It's challenging to understand where a project stands by just looking at task completion status — you are missing the data that supports that specific task.

Managing from the bottom up is like knowing the completion of a mountain climb by counting the steps you've made towards the top—the chances you've made the right assumptions for this count is like winning the lottery. Counting visible checkpoints is better and the only thing you really need.

Generally speaking, this top-down approach to leadership works well in designers and software

development as a “reverse product engineering” style means a better final product.

which is better: top-down or bottom-up

it depends.

A top-down approach to project management tends to work better when there is a clear direction and an overall understanding of how a project fits into the larger goals of the organization. With repeatable projects — projects teams have successfully completed before — a top-down approach often makes the most sense as a process has already been established.

A good project manager is able to quickly identify the big-picture tasks, while the team focuses on the day-to-day deliverables.

Possible downside: Some of the finer project details may be overlooked.

A bottom-up approach to project management tends to work better for brand new projects, ones where your team does not have prior experience. This approach means that teams can begin to brainstorm and identify unknown risks.

Using this approach means more of the team will have input into the process — a freeflow of ideas and information is created. This, in turn, often means there will be more “out of the box” type thinking as individuals think through various solutions to the task(s) at hand.

Typically, a bottom-up approach means there are more details (and maybe even more tasks).

Possible downside: Time-consuming and resource-heavy.

When it comes to estimating task duration, a critical component to any project plan, managers often use both a bottom-up and a top-down approach:

- **Bottom-up**: Allows teams to estimate how long each sub-task will take. This time then rolls up into an overall time-to-project-completion estimate.
- **Top-down**: Starting with an estimate of how long the entire project will take, then breaking it down into the various tasks.

Leveraging both a bottom-up and top-down approach concurrently means ensures a more accurate overall time estimate.

2.8 Activities

Software Project Management consists of many activities, that includes planning of the project, deciding the scope of product, estimation of cost in different terms, scheduling of tasks, etc.

The list of activities are as follows:

1. Project planning and Tracking
2. Project Resource Management

3. Scope Management
4. Estimation Management
5. Project Risk Management
6. Scheduling Management
7. Project Communication Management
8. Configuration Management

Now we will discuss all these activities -

1. Project Planning: It is a set of multiple processes, or we can say that it a task that performed before the construction of the product starts.

2. Scope Management: It describes the scope of the project. Scope management is important because it clearly defines what would do and what would not. Scope Management create the project to contain restricted and quantitative tasks, which may merely be documented and successively avoids price and time overrun.

3. Estimation management: This is not only about cost estimation because whenever we start to develop software, but we also figure out their size(line of code), efforts, time as well as cost.

If we talk about the size, then Line of code depends upon user or software requirement.

If we talk about effort, we should know about the size of the software, because based on the size we can quickly estimate how big team required to produce the software.

If we talk about time, when size and efforts are estimated, the time required to develop the software can easily determine.

And if we talk about cost, it includes all the elements such as:

- Size of software
- Quality
- Hardware
- Communication
- Training
- Additional Software and tools
- Skilled manpower

4. Scheduling Management: Scheduling Management in software refers to all the activities to complete in the specified order and within time slotted to each activity. Project managers define multiple tasks and arrange them keeping various factors in mind.

For scheduling, it is compulsory -

- Find out multiple tasks and correlate them.
- Divide time into units.
- Assign the respective number of work-units for every job.
- Calculate the total time from start to finish.
- Break down the project into modules.

5. Project Resource Management: In software Development, all the elements are referred to as resources for the project. It can be a human resource, productive tools, and libraries.

Resource management includes:

- Create a project team and assign responsibilities to every team member
- Developing a resource plan is derived from the project plan.
- Adjustment of resources.

6. Project Risk Management: Risk management consists of all the activities like identification, analyzing and preparing the plan for predictable and unpredictable risk in the project.

Several points show the risks in the project:

- The Experienced team leaves the project, and the new team joins it.
- Changes in requirement.
- Change in technologies and the environment.
- Market competition.

7. Project Communication Management: Communication is an essential factor in the success of the project. It is a bridge between client, organization, team members and as well as other stakeholders of the project such as hardware suppliers.

From the planning to closure, communication plays a vital role. In all the phases, communication must be clear and understood. Miscommunication can create a big blunder in the project.

8. Project Configuration Management: Configuration management is about to control the changes in software like requirements, design, and development of the product.

The Primary goal is to increase productivity with fewer errors.

Some reasons show the need for configuration management:

- Several people work on software that is continually update.
- Help to build coordination among suppliers.

- Changes in requirement, budget, schedule need to accommodate.
- Software should run on multiple systems.

Tasks perform in Configuration management:

- Identification
- Baseline
- Change Control
- Configuration Status Accounting
- Configuration Audits and Reviews

People involved in Configuration Management:

Fig 2.11 Configuration Management

2.9 Project Scheduling and Staffing

Once the effort is estimated, various schedules (or project duration) are possible, depending on the number of resources (people) put on the project.

For example, for a project whose effort estimate is 56 person-months, a total schedule of 8 months is possible with 7 people. A schedule of 7 months with 8 people is also possible, as is a schedule of approximately 9 months with 6 people.

Manpower and months are not interchangeable in a software project. A schedule cannot be simply obtained from the overall effort estimate by deciding on average staff size and then determining the total time requirement by dividing the total effort by the average staff size. Brooks has pointed out that person and months (time) are not interchangeable. According to Brooks , "... man and months are interchangeable only for activities that require no communication among men, like sowing wheat or reaping cotton. This is not even approximately true of software...."

For instance, in the example here, a schedule of 1 month with 56 people is not possible even though the effort matches the requirement. Similarly, no one would execute the project in 28 months with 2 people. In other words, once the effort is fixed, there is some flexibility in setting the schedule by appropriately staffing the project, but this flexibility is not unlimited.

Empirical data also suggests that no simple equation between effort and schedule fits well. In a project, the scheduling activity can be broken into two sub activities: determining the overall schedule (the project duration) with major milestones, and developing the detailed schedule of the various tasks.

Overall Scheduling :-One method to determine the normal (or nominal) overall schedule is to determine it as a function of effort. Any such function has to be determined from data from completed projects using statistical techniques like fitting a regression curve through the scatter plot obtained by plotting the effort and schedule of past projects. This curve is generally nonlinear because the schedule does not grow linearly with effort. Many models follow this approach. The IBM Federal Systems Division found that the total n duration, M, in calendar months can be estimated by

$$M = 4.1E^{.36}.$$

In COCOMO, the equation for schedule for an organic type of software is

$$M = 2.5E^{.38}.$$

It should be clear that schedule is not a function solely of effort. Hence, the schedule determined in this manner is not really fixed. However, it can be used as a guideline or check of the schedules reasonableness, which might be decided based on other factors.

One rule of thumb, called the square root check, is sometimes used to check the schedule of medium-sized projects. This check suggests that the proposed schedule can be around the square root of the total effort in person-months. This schedule can be met if suitable resources are assigned to the project. For example, if the effort estimate is 50 person-months, a schedule of about 7 to 8 months will be suitable.

Once the effort is estimated, various schedules (or project duration) are possible, depending on the number of resources (people) put on the project.

For example, for a project whose effort estimate is 56 person-months, a total schedule of 8 months is possible with 7 people. A schedule of 7 months with 8 people is also possible, as is a schedule of approximately 9 months with 6 people.

Manpower and months are not interchangeable in a software project. A schedule cannot be simply obtained from the overall effort estimate by deciding on average staff size and then determining the total time requirement by dividing the total effort by the average staff size. Brooks has pointed out that person and months (time) are not interchangeable. According to Brooks , ". man and months are interchangeable only for activities that require no communication among men, like sowing wheat or reaping cotton. This is not even approximately true of software...."

For instance, in the example here, a schedule of 1 month with 56 people is not possible even though the effort matches the requirement. Similarly, no one would execute the project in 28 months with 2 people. In other words, once the effort is fixed, there is some flexibility in setting the schedule by appropriately staffing the project, but this flexibility is not unlimited.

Empirical data also suggests that no simple equation between effort and schedule fits well. In a project, the scheduling activity can be broken into two sub activities: determining the overall schedule (the project duration) with major milestones and developing the detailed schedule of the various tasks.

Overall Scheduling:-One method to determine the normal (or nominal) overall schedule is to determine it as a function of effort. Any such function has to be determined from data from completed projects using statistical techniques like fitting a regression curve through the scatter plot obtained by plotting the effort and schedule of past projects. This curve is generally nonlinear because the schedule does not grow linearly with effort. Many models follow this approach. The IBM Federal Systems Division found that the total n duration, M, in calendar months can be estimated by

$$M = 4.1E^{.36}.$$

In COCOMO, the equation for schedule for an organic type of software is

$$M = 2.5E^{.38}.$$

It should be clear that schedule is not a function solely of effort. Hence, the schedule determined in this manner is not really fixed. However, it can be used as a guideline or check of the schedule's reasonableness, which might be decided based on other factors.

One rule of thumb, called the square root check, is sometimes used to check the schedule of medium-sized projects. This check suggests that the proposed schedule can be around the square root of the total effort in person-months. This schedule can be met if suitable resources are assigned to the project. For example, if the effort estimate is 50 person-months, a schedule of about 7 to 8 months will be suitable.

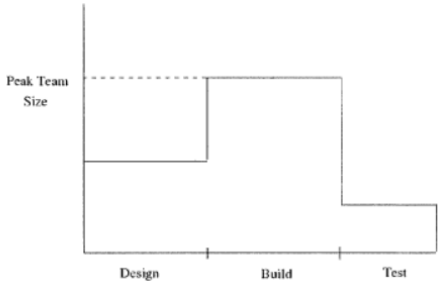


Fig.2.12 Overall Scheduling

2.10 Software Cost Estimation

Whenever we develop a software project, main questions that arise in our mind is how much it will cost to develop and how much time it will take for development. These estimates are

necessary and needed before initiating development. But main critical problem that arises during software cost estimation is lack of case studies of projects usually created in a well-documented manner.

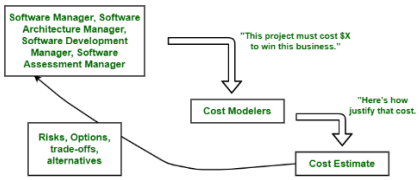
The software industry has inconsistently defined and explained metrics or atomic units of measure, data from real and actual projects are largely and highly suspect in terms of consistency and comparability. There are many questions as debates among developers and vendors of software cost estimation models and tools.

The main topics of these debates are of given below:

- Which model of cost estimation should be used?
- Whether or not to measure software size in source lines of code or function points.
- What constitutes a good estimate?

Nowadays, there are several models available of cost estimation like COCOMO model, Checkpoint, ESTIMACS, SLIM, Knowledge Plan, etc.). Among all of them, COCOMO model is one of most open and well-documented cost estimation models. At present, most of real-world use of cost models is bottom-up rather than top-down.
Below, diagram is given that illustrates and represents predominant practice. The manager of software project defines and describes target cost of software, and after then until target cost can be justified, it manipulates parameters and size.

The process is described in diagram is very necessary to analyze and predict cost risks and understand sensitivities and trade-offs objectively. It simply forces manager of software project to examine and find out risks associated with achieving target costs and to discuss and explain this gained information with other stakeholders.



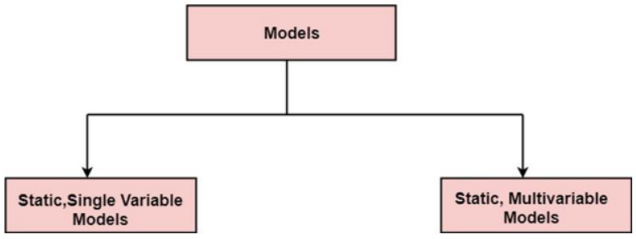
The Predominant Cost Estimation Process

Following are the attributes that Good Software Cost Estimate Contains :
It is simply conceived i.e. planned and supported by project manager, architecture team, development team, and test team responsible for performing work and task.
All the stakeholders generally accept it as ambitious but realizable.
It is based on a well-defined and efficient cost model of software on a credible basis.
It is also based on a similar project experience database that includes and contains similar processes, relevant technologies, relevant environments, relevant quality requirements, and all similar people.
It is also defined and explained in much amount of detail so that all of its key risks are simply understood and probability of success is objectively assessed.
Extrapolating from a good estimate, an ideal estimate would be derived from a mature cost model with an experience base that generally reflects more similar projects that are done by same team with similar mature processes and tools.
Use of Cost Estimation –

One needs to choose and determine how many engineers are required for project to develop and establish a schedule during planning stage.
While monitoring project's progress, one needs to access whether project is progressing towards achieving goal according to procedure and whether it takes corrective action or not.

Cost Estimation Models

A model may be static or dynamic. In a static model, a single variable is taken as a key element for calculating cost and time. In a dynamic model, all variable are interdependent, and there is no basic variable.



Static, Single Variable Models:

- When a model makes use of single variables to calculate desired values such as cost, time, efforts, etc. is said to be a single variable model. The most common equation is:

$$C=aL^b$$

- C = Costs
  L= size
  a and b are constants
- The Software Engineering Laboratory established a model called SEL model, for estimating its software production. This model is an example of the static, single variable model.
- $E=1.4L^{0.93}$
  $DOC=30.4L^{0.90}$
  $D=4.6L^{0.26}$
- **Where** E= Efforts (Person Per Month)
  DOC=Documentation (Number of Pages)
  D = Duration (D, in months)
  L = Number of Lines per code

Static, Multivariable Models:

- These models are based on method (1), they depend on several variables describing various aspects of the software development environment. In some model, several variables are needed to describe the software development process, and selected equation combined these variables to give the estimate of time & cost. These models are called multivariable models.
- WALSTON and FELIX develop the models at IBM provide the following equation gives a relationship between lines of source code and effort:

- **$E=5.2L^{0.91}$**
- In the same manner duration of development is given by
- **$D=4.1L^{0.36}$**
- The productivity index uses 29 variables which are found to be highly correlated productivity as follows:

$$I = \sum_{i=1}^{29} W_i X_i$$

Where Wi is the weight factor for the ith variable and Xi={-1,0,+1} the estimator gives Xi one of the values -1, 0 or +1 depending on the variable decreases, has no effect or increases the productivity.

Example: Compare the Walston-Felix Model with the SEL model on a software development expected to involve 8 person-years of effort.

i. Calculate the number of lines of source code that can be produced.
ii. Calculate the duration of the development.
iii. Calculate the productivity in LOC/PY
iv. Calculate the average manning

2. **Solution:**

The amount of manpower involved = 8PY=96persons-months

(a)Number of lines of source code can be obtained by reversing equation to give:

Software Cost Estimation

$$L = \left(\frac{E}{a}\right) 1/b$$

$$L\ (SEL) = (96/1.4)^{1/.93} = 94264\ LOC$$

$$L\ (SEL) = (96/5.2)^{1/.91} = 24632\ LOC$$

(b)Duration in months can be calculated by means of equation

$$D\ (SEL) = 4.6\ (L)\ 0.26$$

$$= 4.6\ (94.264)0.26 = 15\ months$$

$$D\ (W\text{-}F) = 4.1\ L^{0.36}$$

$$= 4.1\ (24.632)0.36 = 13\ months$$

(c) Productivity is the lines of code produced per persons/month (year)

$$P\,(SEL) = \frac{94264}{8} = 11783\ \frac{LOC}{Person} - Years$$

$$P\,(Years) = \frac{24632}{8} = 3079\ \frac{LOC}{Person} - Years$$

(d)Average manning is the average number of persons required per month in the project

$$M\,(SEL) = \frac{96P-M}{15M} = 6.4Persons$$

$$M\,(W\text{-}F) = \frac{96P-M}{13M} = 7.4Persons$$

COCOMO Model

Boehm proposed COCOMO (Constructive Cost Estimation Model) in 1981.COCOMO is one of the most generally used software estimation models in the world. COCOMO predicts the efforts and schedule of a software product based on the size of the software.

- The necessary steps in this model are:

1. Get an initial estimate of the development effort from evaluation of thousands of delivered lines of source code (KDLOC).
2. Determine a set of 15 multiplying factors from various attributes of the project.
3. Calculate the effort estimate by multiplying the initial estimate with all the multiplying factors i.e., multiply the values in step1 and step2.

The initial estimate (also called nominal estimate) is determined by an equation of the form used in the static single variable models, using KDLOC as the measure of the size. To determine the initial effort E_i in person-months the equation used is of the type is shown below

$$\mathbf{E_i = a*(KDLOC)b}$$

The value of the constant a and b are depends on the project type.

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.

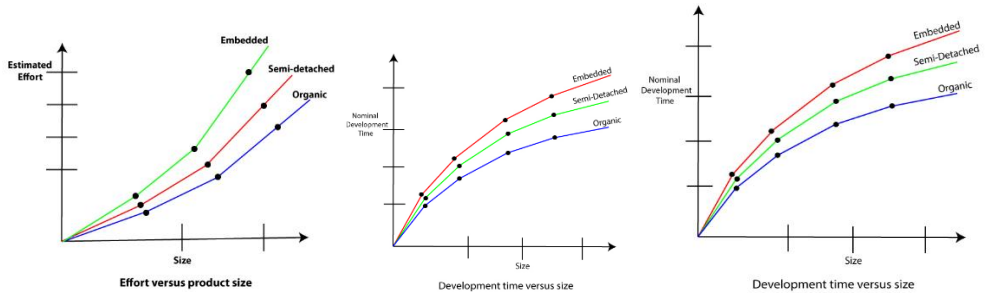
- According to Boehm, software cost estimation should be done through three stages:

1. Basic Model
2. Intermediate Model
3. Detailed Model
4. **Basic COCOMO Model:** The basic COCOMO model provide an accurate size of the project parameters. The following expressions give the basic COCOMO estimation model:
5. **Effort=a_1*(KLOC) a_2 PM**
   Tdev=b_1*(efforts)b_2 Months
6. **KLOC** is the estimated size of the software product indicate in Kilo Lines of Code,
7. a_1,a_2,b_1,b_2 are constants for each group of software products,
8. **Tdev** is the estimated time to develop the software, expressed in months,
9. **Effort** is the total effort required to develop the software product, expressed in **person months (PMs)**. **Estimation of development time**
10. For the three classes of software products, the formulas for estimating the development time based on the effort are given below:
11. **Organic:** Tdev = 2.5(Effort) 0.38 Months
12. **Semi-detached:** Tdev = 2.5(Effort) 0.35 Months
13. **Embedded:** Tdev = 2.5(Effort) 0.32 Months
14. 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.

Effort vs 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.



Example

Suppose a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three model i.e., organic, semi-detached & embedded.

- **Solution: The basic COCOMO equation takes the form:**
- **Effort=a_1*(KLOC) a_2 PM**
- **Tdev=b_1*(efforts)b_2Months**
- **Estimated Size of project= 400 KLOC**
- **(i)Organic Mode**

E = 2.4 * (400)1.05 = 1295.31 PM

D = 2.5 * (1295.31)0.38=38.07 PM

- **(ii)Semidetached Mode**
- **E = 3.0 * (400)1.12=2462.79 PM**

D = 2.5 * (2462.79)0.35=38.45 PM

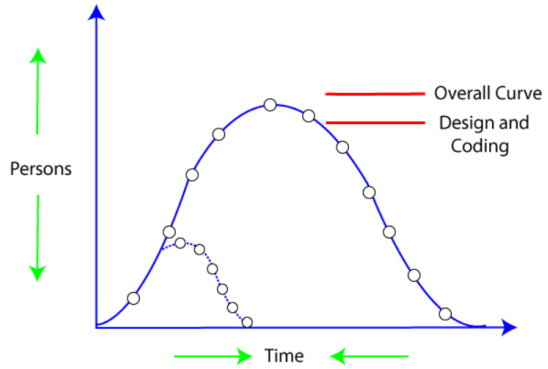
- **(iii) Embedded Mode**

- **E = 3.6 * (400)1.20 = 4772.81 PM**

D = 2.5 * (4772.8)0.32 = 38 PM

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



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** .

t_d correlate to the time of system and integration testing. Therefore, **t_d** can be relatively considered as the time required for developing the product.

C_k Is the state of technology constant and reflects requirements that impede the development of the program.

Typical values of **C_k** = 2 for poor development environment

C_k= 8 for good software development environment

C_k = 11 for an excellent environment (in addition to following software engineering principles, automated tools and techniques are used).

The exact value of **C_k** for a specific task can be computed from the historical data of the organization developing it.

Putnam proposed that optimal **staff develop on a project should follow the Rayleigh curve**. Only a small number of engineers are required at the beginning of a plan to carry out planning

and specification tasks. As the project progresses and more detailed work are necessary, the number of engineers reaches a peak. After implementation and unit testing, the number of project staff falls.

Effect of a Schedule change on Cost

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

- **Where, K is the total effort expended (in PM) in the product development**
- **L is the product size in KLOC**
- **t_d corresponds to the time of system and integration testing**
- **C_k Is the state of technology constant and reflects constraints that impede the progress of the program**
- **Now by using the above expression, it is obtained that,**

$$K = L^3 / C_k^3 t_d^4$$

$$K = C / t_d^4$$

For the same product size, $C = L^3 / C_k^3$ is a constant.

$$\text{Or} \qquad \frac{K_1}{K_2} = t_{d2}^4 / t_{d1}^4$$

$$\text{Or} \qquad K \propto 1/t_d^4$$

$$\text{Or,} \qquad \text{cost} \propto 1/t_d$$

2.11 Work Breakdown Structure

- The Work breakdown structure (WBS) is used to sub-divide the initially identified activities of the project into smaller sub-activities.
- The lowest level of sub-activities is known as the tasks in a WBS.
- WBS is a tree representation of the activities, sub-activities and tasks to be carried out in order to solve the problem.
- Each node in this tree is represented by a rectangle.

- For each of the decomposed sub activities has an arrow from its parent activity.
- The main project represents the root of the WBS.
- The work breakdown structure of a Demo Activity is shown in the figure 4.4.In this figure, the Main Activity is composed of Activities A, B, C, D and E respectively. The activity B is again decomposed into sub-activities B1 and B2. Similarly, the activity C is again decomposed into sub-activities C1 and C2. The WBS helps the project manager to monitor the overall project.

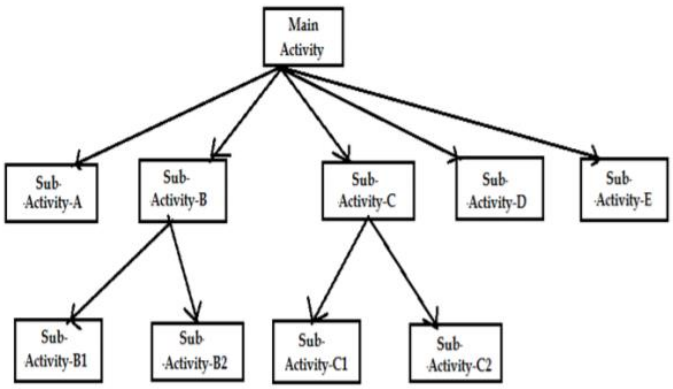


Fig 2.13 Work breakdown structure

2.11.1 Activity Network Representation

An activity network representation shows the different activities involved in the projects, the interdependencies among them and the estimated durations for each of the activities in the proposed project. In the estimation of duration for the activities, the different project manager uses their experiences and past ideas. The activity network representation of Demo Activity project is shown in the figure

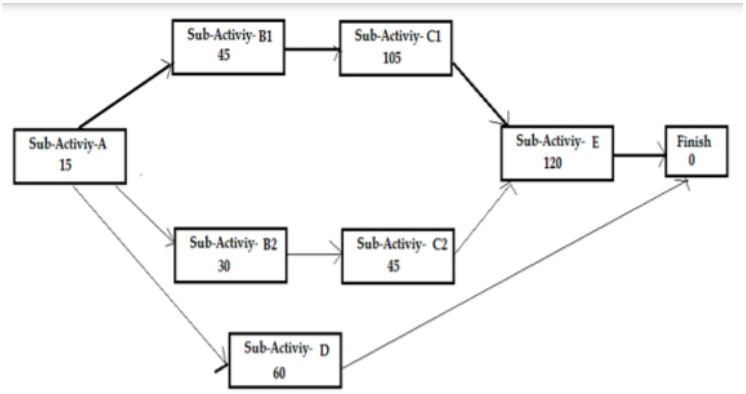


Fig 2.14 Activity Network Representation

- There are two variations in the activity network representations: Activity of Node (AoN) and Activity of Edge (AoE).

- In the AoN, activities are represented by rectangular node and duration of the activity is shown alongside the node. The inter-dependencies are denoted by using directional edges.
- In the AoE, activities are represented by edges and duration of the activity is shown alongside each respective activity edge. Among this two, the AoN representation is most popular since it is easy to understand and revise
- The following project parameters are computed from the above activity networks of Demo Activity project:

Task Number	*Task Name*	*Duration (in days)*	*Dependent on Tasks*
T1	Activity-A	15	——
T2	Sub-activity-B1	45	T1
T3	Sub-activity-B2	30	T1
T4	Sub-activity-D	60	T1
T5	Sub-activity-C1	105	T2
T6	Sub-activity-C2	45	T3
T7	Sub-activity-E	120	T5 and T6

2.11.2 Critical Path Method

- Critical path method is an operational research technique to determine slack times for tasks.
- A path in activity network representation is set of consecutive nodes and edges from start node to finish node.
- The critical path is the sequence of activities with the longest duration. A delay in any of these activities will result in a delay for the whole project. A critical task is one where slack time is zero.
- A critical path only contains the critical tasks from start node to finish node. In the computation of critical path, following parameters are needed to be calculated:
- Minimum Time (MT): maximum of all the paths from start node to finish.
- **Earliest Start Time (EST):** The earliest possible time at which an activity can be started.
- **Latest Start Time (LST):** Equal to the latest finish time of the activity minus the duration of that activity.

  • **Earliest Finish Time (EFT):** the EFT of a task is the sum of EST and duration of the task.

- **Latest Start Time (LFT):** the LFT of a task is the sum of LST and duration of the task.

  • **Slack Time (ST): I**t is the total time that a task can be delayed. ST is calculated as the difference of LST and EST or LFT and EFT.

Example 1

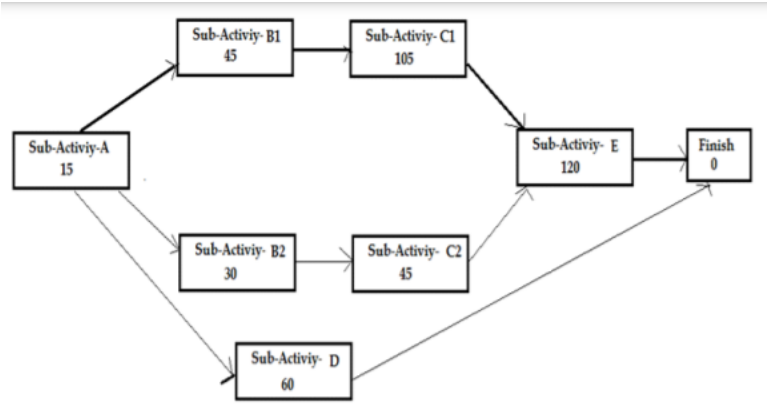


Fig 2.15 Critical Path Method

MT=285

Task Name	EST	EFT	LST	LFT	ST
Activity-A	0	15	0	15	0
Sub-activity-B1	15	60	15	60	0
Sub-activity-B2	15	45	90	120	75
Sub-activity-C1	60	165	60	165	0
Sub-activity-C2	45	90	120	165	75
Sub-activity-E	160	285	165	285	0
Sub-activity-D	15	75	225	285	210

Example 2

Activity	Predecessor Activity	Duration (Weeks)
A	-	3
B	A	5
C	A	7
D	B	10
E	C	5
F	D,E	4

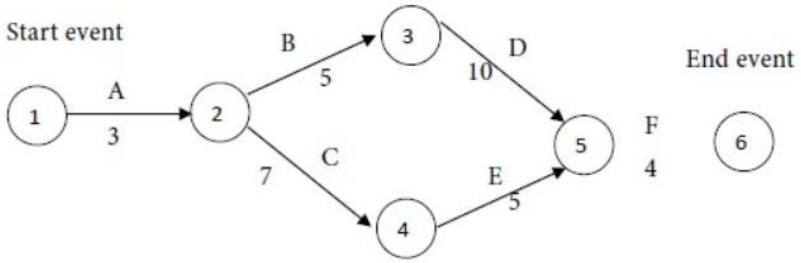


Fig 2.16. Network path for the given data

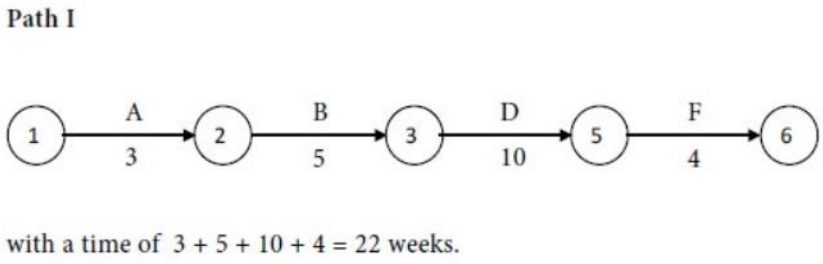


Fig 2.17 possible path from fig 2.16

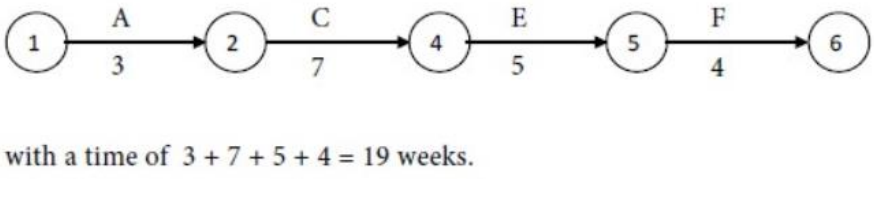


Fig 2.18 possible path from fig 2.16

PERT Chart

- PERT (Project Evaluation and Review Technique) is a representation of Activity Network where activities are represented by boxes and arrows represents their dependencies. In a PERT chart pessimistic, likely, and optimistic estimates are made for each task instead of making a single estimate. This makes critical path analysis in PERT charts very complex.
- The PERT chart representation of the above demo activity project of figure 4.5 is shown in figure 4.6. PERT charts are a more complicated form of activity chart. With the use of PERT chart monitoring the timely progress of activities is possible.

Steps in PERT

- Step 1: **Identify all the project's activities.**
- First, define all the major phases, milestones, and tasks needed to complete the project.
- Step 2**: Identify dependencies**

  If you determine some tasks or activities have dependencies, you will want to depict those tasks with directional arrows. This will ensure your team knows the sequence they need to tackle each task.

- Step 3: **Draw your chart.**

  The next step is to take the events and milestones (numbered nodes) you've identified and draw them out. Then write out the tasks and activities that the team must complete between each node, using directional arrows or divergent arrows accordingly.

- Step 4: **Establish timelines for all activities.**

  You should now set a timeframe when the team will need to complete those tasks along with all arrows. For example, in our mockup above, you can see the "Train sales" activity has a timeframe of 1 day. This can represent the estimated timeframe and/or deadline you set for the activity.

Fig. 2.19 Dependency

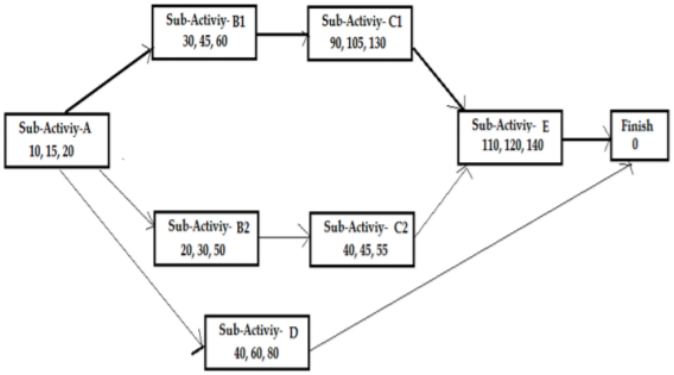


Fig 2.20 PERT chart

Gantt chart

A Gantt chart is a project management tool that illustrates a project plan. It typically includes two sections: the left side outlines a list of tasks, while the right side has a timeline with schedule bars that visualize work. The Gantt chart can also include the start and end dates of tasks, milestones, dependencies between tasks, and assignees. To keep up with the demands of modern software development, roadmap tools like Jira Software include features like a collapsible task structure and resource management panels. These roadmap tools help teams maintain a coherent project strategy despite the iterative nature of the software development process.

Origins of the Gantt chart

In the early part of the 20th century, Henry Gantt revolutionized project management with Gantt charts. At that time, they were written out on pieces of paper. With the rise of computers in the 1980s, Gantt charts became increasingly complex and elaborate. Today, Gantt charts are still one of the most widely used project management tools.

Today, Gantt chart tools are often referred to as roadmap tools. Jira includes two roadmap tools to create Gantt charts for your projects: Roadmaps, which creates plans around Jira issues assigned to a team, and Advanced Roadmaps, which does the same thing across teams and organizations.

What is a Gantt chart used for?

Project managers use Gantt charts for three main reasons:

Build and manage a comprehensive project

Gantt charts visualize the building blocks of a project and organize it into smaller, more manageable tasks. The resulting small tasks are scheduled on the Gantt chart's timeline, along with dependencies between tasks, assignees, and milestones.

Determine logistics and task dependencies

Gantt charts can be employed to keep an eye on the logistics of a project. Task dependencies ensure that a new task can only start once another task is completed. If a task is delayed (it happens to the best of us), then dependent issues are automatically rescheduled. This can be especially useful when planning in a multi-team environment.

Monitor progress of a project

As teams log time towards issues in your plan, you can monitor the health of your projects and make adjustments as necessary. Your Gantt chart can include release dates, milestones, or other important metrics to track your project’s progress.

The benefits of using a Gantt chart

There are two main reasons why Gantt charts are loved throughout the project management world. They make it easier to create complicated plans, especially those that involve multiple teams and changing deadlines. Gantt charts help teams to plan work around deadlines and properly allocate resources.

Projects planners also use Gantt charts to maintain a bird’s eye view of projects. They depict, among other things, the relationship between the start and end dates of tasks, milestones, and dependent tasks. Modern Gantt chart programs such as Jira Software with Roadmaps and Advanced Roadmaps synthesize information and illustrate how choices impact deadlines.

Disadvantages of Gantt Chart

- Require more efforts for Creating and Managing the Chart
- Updating a Chart is Very Time Consuming

- All Tasks are not visible in a single view of a Gantt
- Need to scroll and Click additional buttons to view remaining items
- Stacks represents only the time and not the hours of the work
- Not easy to re align the tasks from on section to another
- Not easy to calculate the aggregates

Question Format:

Part-A			
Q.No	**Questions**	**Competence**	**BT Level**
1.	Define activities	Knowledge	BTL 1
2.	What do you understand by work breakdown structure (WBS)?	Remember	BTL 1
3.	What is critical path?	Remember	BTL 1
4.	Write any three network diagram methods?	Analysis	BTL 4
5.	Difference between PERT and CPM.	Understand	BTL 2
6.	What is resource allocation?	Evaluate	BTL 5
7.	Identify the various approaches you would use to identify activities.	Analysis	BTL 4
11.	Compose the 3 basic objectives of organizational behavior proposed by Taylor.	Understand	BTL2
12.	List out the members in software development team	Knowledge	BTL2
13.	Explain Personality Traits	Understand	BTL2
Part-B			
Q.No	**Questions**	**Competence**	**BT Level**
8.	Describe the organizational behavior with example	Analyze	BTL 4
9.	Analyze the factors and characteristics that are involved in making a team.	Understand	BTL 2
10.	Do staff selection relate with quality of product? Justify with appropriate reason.	Evaluate	BT5
11.	Discuss the leadership models and the functions of a leader with an example.	Create	BT6

12.	Draw the critical path diagram for the tabulated activities. Identify critical path and the duration of the project. <table><tr><th>Activity</th><th>Duration (days)</th><th>Immediate Predecessor</th></tr><tr><td>A</td><td>9</td><td>–</td></tr><tr><td>B</td><td>10</td><td>–</td></tr><tr><td>C</td><td>9</td><td>A</td></tr><tr><td>D</td><td>15</td><td>B, C</td></tr><tr><td>E</td><td>12</td><td>B</td></tr><tr><td>F</td><td>4</td><td>D</td></tr><tr><td>G</td><td>8</td><td>E</td></tr><tr><td>H</td><td>10</td><td>F, G</td></tr></table>	Analyze	BT 4
13.	Identify the organizational structure where staff in different departments need to be available to work on a particular project?	Create	BT 6
14.	Describe PERT network in detail with example.	Understand	BT 6

References

1. Adams, J. Hayes, J. and Hopson, B.(eds) (1976) Transition: understanding and managing change personal change London, Martin Robertson
2. Garrett, V. (1997) Managing Change in School leadership for the 21st century Brett Davies and Linda Ellison, London, Routledge

UNIT – III - PROJECT REVIEW – SBAA3017

UNIT 3 PROJECT REVIEW

Tracking Meetings - Recovery plans - Schedule Work & Escalation Meetings. Project Engineering: Product Requirements - Understanding the Customer Problem to solve - Initial Investigation, Strategies for determining information requirements, Information gathering Tools - Product Objectives.

3.1 PROJECT REVIEW

Form is completed at the end of the Initiation *project phase* to tell the sponsor whether the project has achieved its objectives to date.

First, a Project Management Review is conducted to measure the deliverables produced by the project, then the results of the review are documented on this Project Review form which is presented to the sponsor for approval.

Project Phase reviews are conducted at the end of the Initiation, Planning and Execution phases within a project. This form helps you to complete a project review for the the 'Initiation' project phase.

The form helps you to document the results of your Project Review, by stating whether the:

- Project is currently delivering to schedule
- Budget allocated was sufficient at this point
- Deliverables have been produced and approved
- Risks have been controlled and mitigated
- Issues were identified and resolved
- Changes were properly managed
- Project is on track

The form helps you to:

- Document the results of your Project Review
- Clearly communicate the progress of your project to your sponsor
- List any risks or issues which have impacted the project
- Show sponsor the deliverables produced to date
- Seek approval to proceed to the next phase

By implementing Phase Reviews, you are putting in place the necessary "check-points" to monitor and control the project, thereby ensuring its success.

What is a Project Review?

A Project Review is an assessment of the status of a project, at a particular point in time. The first time in the project life cycle that a project review is undertaken is at the end of the first project phase, called "Initiation". During this project review, a decision is made as to whether or not the team has met the objectives and is approved to proceed to the next project phase, being the "Planning" phase. Performing a project management review at the end of each phase is critical to the success of the project, because it allows the Project Sponsor to control the progress of the project and make sure that it passes through each Project Phase smoothly.

When do I complete a Project Review?

As soon as the project team believes they have completed a particular project phase, a project review should be undertaken. There will usually be at least three project reviews during the project life cycle: at the end of the Initiation, Planning and Execution project phases. The template on this page will help you complete a project review for the "Initiation" project phase. The items included in the project review form are targeted towards this phase specifically.

3.2 THE PROJECT TRACKING MEETING

THE PROJECT TRACKING MEETING and its derivative actions serves as the primary driving force behind the project. Here are responses to some frequently asked questions about this control mechanism.

What is the primary purpose for project tracking meetings?

Project planning is about getting in control. Project tracking is about staying in control. The No. 1 reason for project tracking meetings is to identify potential project problems before they occur. The No. 2 reason is to ensure that recovery plans are put in place before unrecoverable harm occurs. Project tracking is predominately focused on being proactive, not reactive.

How often should a project formally be tracked?

Project tracking meetings should occur once a week. (The exceptions are small-duration projects that are only several weeks or less in duration, in which case, project tracking meetings could occur more frequently.) Meeting less often than each week can delay the discovery or discussion of serious problems, which can harm the successful outcome of the project. Meeting more frequently than weekly can be quite unproductive and waste scarce time, because it requires members of the project tracking meeting to spend additional time preparing more than one progress status per week. It also requires the project tracking meeting members to spend additional time in meetings rather than being free to work their plans.

Does it matter what day of the week the project tracking meeting is conducted?

Yes. Routine project tracking meetings are very important to the health of a project and require participants to attend—on time and prepared. Therefore, avoid having meetings on Mondays or Fridays; these days are often used as holidays or personal days for extended weekends. Furthermore, meeting participants use Mondays to catch up on progress that may have occurred over the weekend. This leaves Tuesdays, Wednesdays and Thursdays for the meeting. My favorites are Tuesdays and Wednesdays, because I like to reserve the day after the project tracking meeting for work and escalation meetings to address unresolved issues or new issues identified from the meeting. This means that Thursday would be used as the reserved day if the project tracking meeting were held on Wednesday.

What if project members find themselves assigned to more than one project?

If there are multiple ongoing projects across an organization and, referring to the answer to the previous question, all project tracking meetings were held on Tuesdays and Wednesdays, it seems that there would be too many meeting conflicts. However, most organizations seldom experience this problem. In those cases where it does occur, the project managers need to meet among themselves and carefully coordinate their project tracking meetings to avoid such conflicts.

Who should attend project tracking meetings?

Everyone might attend for small projects of, say, 10 or fewer members. For all larger sized projects, a representative from each organization or team would attend. Managers typically should not own activities or tasks and, therefore, are optional attendees. However, the most effective managers will attend as often as they can to support their employees who are assigned to the project.

Is it overkill for the project tracking meeting participants to meet briefly every day?

The weekly, formal project tracking meeting is a must. However, here is an additional technique that can work surprisingly well:The project manager can meet with participants of the project tracking team for 15–30 minutes at the start of each workday to ensure that the top-priority problems are receiving the attention they require. This mostly is an informal meeting that requires little preparation, if any, from the participants.

These subjects are presented at tracking meetings in the order described here.

Project High-Priority Areas. The project manager displays the top three to five problems now plaguing the project, while their "owners" report any late-breaking news. These problems are currently impacting a major project milestone (often called an "issue") or have the potential (called a "risk") to do so. The project manager tracks these high-priority areas daily, all other project progress/problems are tracked weekly.

Overview of Project Progress. The project manager presents this information on a single chart that lists the project's major milestones. The chart is first presented as a high-level view of the project plan and is updated for each tracking meeting to illustrate the "big picture" of where the project's progress is in relation to where it was planned to be. This chart has special interest to the project's sponsor and client. It is expected to be a reasonably good view of the forest without obstruction from all the trees. The chart likely will require updating by the end of the tracking meeting after all participants have presented their status.

Progress of Project Activities. Each participant of a project tracking meeting presents their status against their portion of the project plan. This status includes metrics to substantiate the progress made, identifies their top three to five priorities and their corresponding status, and a 30-day outlook of what can be expected, including whether or not help is required.

Progress of Action Items. An action item is a project problem that is logged, assigned to an owner to resolve, and then tracked until it is closed. The owners of action items present their progress. Presentations of action items can be performed at the same time a participant has the floor to present his or her progress of project activities.

Project Outlook. The project manager forecasts a 30-day outlook for the project. That is, 30 days from now, will the project be on schedule, within cost, and what is the overall likelihood (high, medium, low) that the project will complete as planned? Although this information initially should be prepared before the meeting, it is likely that it must be altered real-time based on the latest information collected at the meeting.

Schedule Work/Escalation Meetings. The project manager spends the last moments of the meeting declaring what project activities and action items require special attention over the next

two to three days. If possible, these meetings should be scheduled now, preferably for the following day. These meetings typically become priorities within the project.

THE MEETING AGENDA should be published and followed throughout the meeting. The project manager has the authority to declare that only a subset of the participants present their plan status, as well as which action items are presented. All status not presented must still be submitted to the project manager at the end of the meeting. This allows the project manager to study the status without subjecting everyone to presentations on other than the currently most important areas of the project plan and action items.

The project manager must lead these meetings so that they run on schedule and are productive. A beneficial technique is for the project manager to assign a scribe to record the meeting notes so that the project manager is free to concentrate on effectively running the meeting. This meeting and its derivative actions serve as the primary driving force behind the project. As these meetings go … so goes the project

3.3 Project Recovery

Recovering a project that is really troubled is not for the faint hearted. It is the opinion of this author that recovery is not achievable and repeatable without a project recovery methodology. The more complicated a project is, the greater the importance of a recovery methodology.

A number of project recovery methodologies exist. The majority of the project recovery methodologies have a four-step process, namely (1) audit/identification of current status, (2) analysis of the problems (fault finding), (3) negotiate the recommended actions, and (4) implement the recovery plan.

Exhibit 3 depicts the high level project recovery methodology and associated actions:

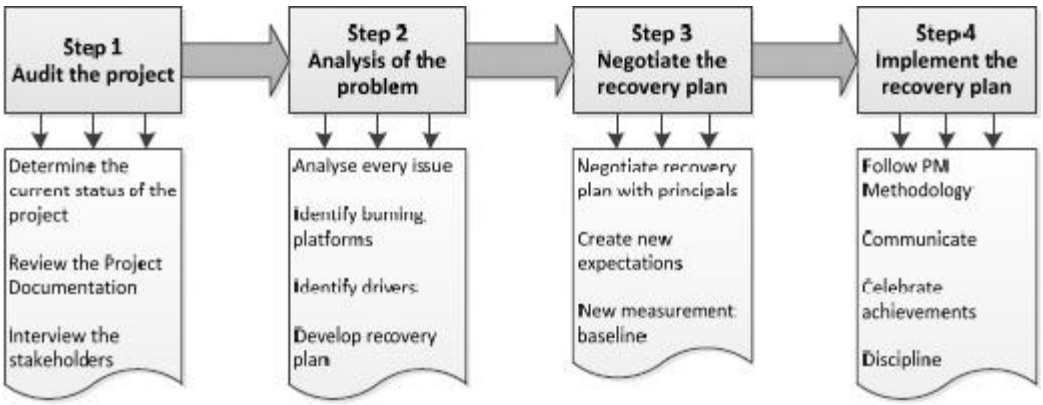


Fig. 3.1 Exhibit 3: High-level project recovery methodology.

Step 1: Audit the project. The primary aim of this step is to review all the project documentation to determine the current status. All the relevant stakeholders should be interviewed to obtain their understanding in terms of expectations, issues, and risks. The majority of the problems should be identified through the audit process.

Step 2: Analysis of the problems. Analyze every identified problem on the project. Identify the burning platforms. Also ensure that the basics are put in place, such as redefining or confirming

the scope of the project. Identify the real schedule and cost drivers. Suspend or eliminate deliverables in the scope that are not immediate requirements for achieving the project objectives and re-schedule it as far in the future as possible. Draw up a new plan aimed specifically at the recovery effort. Be realistic in terms of what is achievable with the current funds and resources.

Step 3: Negotiate the recommended actions with the major principals. The principals' support is critical to implementing the recovery plan. Present the recovery plan to them, substantiate each and every action in the plan, and negotiate where possible. This negotiation will also create new expectations and ownership from the stakeholders and ensure that the new baselines will be the ones that measurement is made against.

Step 4: Implement the recovery plan. Follow a sound project management methodology. Control the project by means of the performance measurement baseline and manage issues actively. Make sure communication channels are open to the stakeholders, keeping them informed is a critical success factor for continued support; otherwise, they might get easily disillusioned. Escalate major issues to ensure that the steering committee also takes ownership in the project. Enforce discipline — this is no time to relax! Celebrate each and every achievement, because this will motivate the team.

Saving the Project Objectives

The project recovery methodology as described above was implemented. The audit revealed some of the obvious problems that exist on a troubled project. The scope was poorly defined and did not include all of the intended deliverables of the project. The poor scope definition resulted in an incomplete schedule and budget. Resources were not leveled or poorly planned. Issue management was not done formally, resulting in a number of the warning signs not recognized. There was no project management methodology defined, thus no real project controls. The project budget was not fully developed, thus the project manager had no control over the project costs. The development of the hectares and planting of the bananas were still the major drivers of the project. This made it easier to recover. On further investigation, it also appeared that the target set for the end of 2009 was to plant 1,000 hectares and not the 750 hectares as was stipulated in the project objectives. Thus recovering the schedule might not be as difficult as thought in the beginning.

The following recommendations were made to the project steering committee:

Define the complete scope of the project; this will enable better planning and costing of the project.

Return to the project objective to plant only 750 hectares by the end of 2009 and not the 1,000 hectares. Do not introduce actions now that will accelerate the planting schedule, but might complicate the steady state farming in the future; in other words, stick to the 10 hectares per week per farm. As soon as the planting rate of 10 hectares per week per farm is stabilized, one can always launch one of the other farms earlier and ramp-up the planting rate to 30 hectares per week over three farms. This will meet the schedule requirements.

Determine which deliverables are critical for providing the fruit and determining which deliverables can be procured later. Thus, place all deliverables that are not directly related to the planting rate subordinate to those that are. One such deliverable was a major IT

infrastructure that was not critical, but a major cost driver, and should be suspended or placed on hold.

Introduce a project management methodology and execute the project according to it.

Micromanage the planting activities by having a planning meeting the day before, an instruction meeting each morning before work commences, and a control meeting before the planning meeting for the next day. Bring back discipline, which is what project management is all about anyway.

The project steering committee accepted all these recommendations and the implementation of the recovery plan commenced.

The recovery effort, with a lot of effort, started yielding the desired results within two months. The planting occurred at the desired 10 hectares per farm per week and stabilized on 20 hectares planted per week. The third farm was introduced earlier and the rate increased to the desired 30 hectares per week for the three farms. The forecasts indicated that the 750 hectare target would be met before the target date and this without compromising the quality of the fruit. Efforts were also put in place to train the project team members in project management and the project management methodology to ensure that after the consultants depart, the expertise remains. The project was under control and back on track — success!

3.4 Escalation Meetings

The Escalation Process clarifies the boundaries and channels of decision-making throughout an organization in order so solve the problem quickly and with clarity. Designed around the concept of a core project team with a clear project manager, this process outlines a path that allows the core team to make decisions at lower levels of the org chart while having a predefined path for exception management.

This might be called an escalation plan, or escalation workflow that moves a high-priority issue up to a higher level. It minimizes the time it takes to escalate decisions that are beyond their scope of authority. It is among the essential tools for an agile product development process. Though it is used extensively for customer support, customer service agents, and managing SLA timeframe, it can also be an essential tools for an agile product development process.

Escalation Management in Four Steps

A project manager creates the process (or escalation matrix) in four steps:

Project Escalation Template		
Decision Category	**Decision Type**	**Escalation Path**
Product Features	Feature	IC: FL: PM: CT/FD: BU: CMO
Staffing	Schedule, Cost	IC: FL: PM/FD: CT: BU
Change Management	Process	IC: PM: CT
Product Cost	Cost	IC: FL: PM: CT/FD: BU: CFO: COO
Legal	Legal	IC: FL: PM: GC
Customer Service	Schedule, Cost	IC: FL: PM: BU: CMO
Process Improvement	Schedule, Cost	IC: PM: PMO

Project Escalation Map Legend		
Legend	**Org. Contributor**	**Decision Authority**
IC	Individual Contributor	Functional Execution
FL	Functional Lead	Functional Delivery
PM	Project Manager	Project Delivery
CT	Cross-Functional Team	Cross-Functional Execution
FD	Functional Director (or VP)	Functional Budget
BU	Business Unit Lead	BU Delivery
GC	General Counsel	Legal Compliance
PMO	Project Mgt. Office	Cross-Project Delivery
CMO	Chief Marketing Officer	Customer Experience
COO	Chief Operating Officer	Corporate Execution
CFO	Chief Financial Officer	Financial Impact
CEO	Chief Executive Officer	Corporate Impact

Fig. 3.2 Sample project escalation map

1. Define decision categories: these can include areas such as finance, staffing, tools, and technical features/functionality. When defining the categories, the project manager should be mindful of the right balance in the number of categories based on the complexity of your organization. Project managers should to avoid too many issues to take to the next level to overburden the process, and too few will not provide a meaningful escalation path. This is especially important in **new product development**, where there is inherently more risk.
2. In each category, project managers should determine the appropriate escalation procedure by functional responsibility. Project managers should start at the lowest level in the organization, typically an individual contributor. Some decision categories can have parallel communications (functional and cross-functional) and typically this flows from the project manager.
3. Define the key organizational contributors and their decision-making authority including the project manager. This can vary based on the size and complexity of the project. In some cases, there will be dual communication paths (functional and project) to ensure rapid decision-making.
4. The project manager then reviews with management to get agreement on the categories, decision authority, and escalation procedure. It is the decision authority and the process to raise an escalation should be signed off by management – this is most important.

What Are The Benefits of the Escalation Process for Project Managers?

Minimizes delays in delivering products to market.
Drives accountability in the decision-making process.
Saves time and energy by providing a clear escalation path for decision-making.
Educates new team members on how to make decisions quickly.

In general, the escalation should be resolved in hours/days not days/weeks.

Escalation Meeting

The Executive Escalation Meeting will meet and discuss how and when a specific decision or a recommendation made at the level of the Euronext Derivatives Steering Committee is to be implemented and may ask the Euronext Derivatives Steering Committee to reconsider the decision or recommendation in question in light of constraints or difficulties linked to the implementation or new facts discussed in the Executive Escalation Meeting.

Board Meeting means a meeting of the Board;

Open meeting or "public meeting" means a meeting at which the public may be present.

Annual Meeting means the annual meeting of the stockholders of the Company.

Mid-Span Meet means an Interconnection between two (2) networks, designated by two (2) Telecommunications Carriers, whereby each provides its own cable and equipment up to the Meet Point of the cable facilities. "Mid-Span Meet POI" A Mid-Span Meet POI is a negotiated Point of Interface, limited to the Interconnection of facilities between the Qwest Serving Wire Center location and the location of the CLEC switch or other equipment located within the area served by the Qwest Serving Wire Center.

meeting of shareholders means an annual meeting of shareholders or a special meeting of shareholders;

special meeting of shareholders means a meeting of any particular class or classes of shareholders and a meeting of all shareholders entitled to vote at any annual meeting of shareholders at which special business is to be transacted.

Special Meeting means a special meeting of the holders of Voting Shares, called by the Board of Directors for the purpose of approving a supplement.

General Meeting means the annual or any special general meeting of the Association.

Shareholders Meeting

Company Meeting means the special meeting of Company Shareholders, including any adjournment or postponement of such special meeting in accordance with the terms of the Arrangement Agreement, to be called and held in accordance with the Interim Order to consider the Arrangement Resolution.

Regular Meeting means a scheduled meeting held in accordance with the approved calendar/schedule of meetings.

Shareholder Meeting means the meeting of the holders of Shares for the purpose of seeking the Shareholder Approval, including any adjournment thereof. relevant meeting means a meeting of the authority to consider whether or not to approve a proposal to dismiss a relevant officer; and Meeting Date means the date on which the Meeting is held in accordance with the Meeting Order.

relevant meeting means a meeting of the authority to consider whether or not to approve a proposal to dismiss a relevant officer.

Meeting Date means the date on which the Meeting is held in accordance with the Meeting Order.

3.5 Project Engineering

Product Requirements

To meet the new challenges - today's companies will have to change their strategies and to increase the customer experience - in order to get a better chance of succeeding in the future. Finding and maintaining a balance between operations, resources and market demands is critical, but at the same time - important to succeed with. This means that the companies should improve their product development process in order to stay competitive.

Companies that periodically introduce new products - have a bigger chance of succeeding on today's competitive market. Companies spend billions of dollars on new product development and at the same time 40 to 90 percent (depending on category) of new products fail.

Understanding the customer needs is very important step at the start of the product development, when the requirements are set for the product. If the customer needs are not properly understood in the early phase - there is a risk that changes have to be made later in the project, when it may be difficult to incorporate them. Additionally - the cost of changes increases the longer the project progresses.

Aside from the customer needs - the **product's requirements** include all functions, features and behaviors that the product must possess, so that it will be efficient, ease to use, safe, low cost, etc. In other words - any function, constraint or other property that is required in order to satisfy user's needs. User requirements are gathered from users and described from the analyst with a customer point of view.

Requirements Attributes (features)

To be considered as good - each product requirement must possess certain attributes. For example, in the software engineering there is the INVEST principle, so the requirements should be: Independent, Negotiable, Valuable, Estimable, Small, Testable.

For the general product requirements - the desired attributes include the following:

Clear - the requirement is unambiguously defined.

Complete - all that is needed - is consisted in the requirement.

Credible - the requirement must be technically possible to be achieved.

Consistent - the requirement is not in collision with other requirements.

Verifiable - the requirement must be verified and validated (so it should be expressed in physical terms and measurable attributes).

Necessary - if some requirement brings clear business value - it can be treated as more necessary than other requirements that are “nice to have” (e.g. product color).

Product Requirements Lifecycle

The product requirements lifecycle includes the following phases:

Requirements gathering,

Requirements analysis,

Requirements verification / validation,

Requirements management and tracking,

Requirements specification (documentation).

We will now briefly describe each of these phases.

Gathering of Product Requirements

In this phase - we have to determine the sources of the product requirements and the approach for their handling and management. It is important to identify all of the sources:

Objectives - also known as ‘Business concerns’ or ‘Critical Success Factors’, represent the comprehensive goal of the product. Special attention should be paid to the cost of reaching the target (through a feasibility study).

Domain knowledge - a production engineer needs to have knowledge of the product’s application domain and to be ahead of others; to resolve conflicts in the requirements and to be a mediator between all participants.

Stakeholders - some products might prove to be unsatisfactory, because of the different perspectives of the stakeholders. The product engineer needs to identify a correct representation and to manage these situations (different culture and politics).

Operating environment - product requirements come from the environment in which the product is used on daily basis.

The organizational environment - the product will have to support the business environment and the processes (conditions) that it will be used it. New product should not bring unplanned changes, so it requires careful analysis by the product creators, before making a final decision.

When all sources are identified - the product engineer can start the requirements elicitation (gathering). This area deals with the techniques for selection and extraction, which can be particularly sensitive. For example, users may have difficulty in defining actions and sometimes they omit important information. These techniques are not passive and require hard work for the articulation of all requirements. The interview represents a traditional way of collecting client’s requests. This technique has certain advantages and limitations.

It is necessary to prepare scenarios for the requirements elicitation:

Providing a framework for defining the questions: What if...? How is this done..? etc.

Using a technique known as Conceptual modeling - i.e. Case Modeling and Diagrams.

Also, holding meetings (brainstorming) and group work can be more efficient than individual techniques of observation (observing how users will use the product and interact with it).

Requirements Analysis

After the process of requirements gathering - a process for their analysis is necessary. Requirements analysis refers to the processes required in order to:

Identify and resolve the conflicts between the product requirements;

Detection of product features, and how the product will behave in real environments;

Elaboration of product requirements in the process of product development.

The traditional view of the analysis requires the use of conceptual modeling and other methods for analysis, such as the Structured Analysis and Design Technique (SADT).

The most important thing is to accurately describe the clients' requirements, in order to facilitate their validation, verification of their implementation, and the cost estimates.

Requirements analysis includes:

The primary mechanism for requirement validation is a formal technical review. The review team includes engineers, customers, and other stakeholders who are checking product specification for content errors, areas that need clarification, insufficient information, inconsistencies (a major problem when making large products or systems), conflicting demands, or unrealistic requirements (which cannot be achieved).

Although requirement validation can be done in order to uncover potential errors, it is useful to check each requirement according to a predefined list of questions. The following questions are only a small part of what can be asked:

• Are the requirements clearly stated? Can they be misinterpreted?

• Are all the aspects of the product operation (functioning) covered?

• Is the source of the request identified (e.g. person, regulation, document)? Has the final form of the requirement been verified from the original source?

• Is the requirement quantified?

• What other requirements are associated with this requirement? Are they clearly stated using a reference matrix or other mechanism?

• Does the requirement affect domain restrictions?

• Can the requirement be tested? If it can, can we determine tests (so-called validation criteria) for checking the requirement?

• Can the requirement be tracked in any product that will be created?

Product Requirements Management and Tracking

It was stated above that the product requirements are changing and that there might be a need to change requirements throughout the product lifecycle. Requirement management is a set of activities that helps the project team identify, control, and track requirements and their changes at any point in the product's life. Also in this step - requirements prioritization and re-ordering can be done.

The requirements management begins with identification. Each requirement is assigned a unique identifier of the following form:

<type requirement> <requirement Id>

where < type requirement> can receive these possible values: F = functional requirement, B = behavioral requirement, I = input requirement, O = output requirement etc. Thus, the requirement identified by F09 denotes a function requirement number 9.

Once the requirements have been identified, monitoring (tracking) tables are developed. Each tracking table links specific requirements to one or more aspects of the product or the environment. Some of the many tracking charts are the following:

Properties Tracking Table - displays how the requirements relate to the visible features of the product.

Source (reason) tracking table - identifies the source of each requirement.

Dependency Tracking Table - indicates how the requirements are related to each other.

Subsystem Tracking Table - categorizes requirements according to the subsystems they belong to.

Product Requirements Documentation

After the gathering and the analysis of the product requirements - the product analyst and the project team should specify all product requirements. Under the requirements specification - there should be a composition (drafting) of a Product Requirements Document (PRD).

The product requirements document is comprised of: full product's overview, the product features, working environment etc. It is not solely dedicated to the product's functions and features, but it should list all functional and non-functional requirements (such as usability, safety, reliability, etc.). While writing this document - we must consider the following aspects of the product:

Define the main product principles (e.g. it should be reliable and ease of use).

Define the product's purposes, i.e. the business values that it will bring for your customers (ease of use, cheap price, new features).

Define your clients / customers, your competition, and your product development team.

Define the goals of the product, and tasks that can be fulfilled by using the product.

Define the different user profiles, e.g. young adults, elderly people, wide audience etc.

The PRD document should be written in clear and concise style, so the product developer will have a clear understanding about the function of the device, and can jump straight into the design.

Product Requirement Types

There are particular types of requirements that each product should fulfill and here we list the most relevant ones.

Functionality

Each product is developed with its main purposes (goals). For example - the cars are designed to take us to the destination by driving, with certain technical characteristics, exterior and interior features etc. The product's functional requirements should include all technical details in order to provide its uninterrupted functionality.

Quality

Quality is any element (feature), measureable or not, that gives things values beyond their functionality and features. Typical quality requirements include: reliability of the product, consistency, durability, availability, customer experience, look and feel, performance, maintainability, materials / ingredients etc.

Usability

Usability requirements are created to ensure that the product will be easy to use. Usability is a broad concept, but it can be split into more elementary ones - like intuitive and easy to learn, so the users are able to flow through the tasks without being interrupted. Also - they should increase the productivity and performance of the user while using the product.

Reliability

Reliability is one of the most important non-functional (quality) requirements. Reliability is measured as time to failure, probability of failure and failure-free operation. It also defines the mean time to repair and between repairs, coefficient of availability and unavailability, failure rate etc.

Safety

Safety is often introduced as requirement in order to avoid, or reduce potential risks. In physical products (items) - safety relates to well-being, health and life protection of the users, while in the IT systems - it refers to protection of user's personal data (e.g. credit card number etc.).

Packaging

Packaging requirement refers to the product packaging and its purpose is to protect the product for transport damages, and also for marketing (design) purposes. Packaging requirements are subject to regulations in all countries worldwide. For example, European Union has introduced Packaging Regulations Directive in 1998, and it must be obeyed in all EU countries. It starts with massive containers, dangerous materials, chemicals and also includes food products and small bags.

3.6 Understanding the Customer Problem to solve

As someone responsible for handling customers, you should know when to listen and when to speak. This will enable you to understand the problem, the first step in solving any problem.

Meeting Basic Needs

These are the basic steps in understanding a customer's problems −

Listen patiently to what the customer has to say

Write down all that is being said

Don’t interrupt if you have a query; note it down

Once the customer has finished narrating his problem, summarize your understanding from your notes

Add to the notes you have taken if the customer has anything more or different to say

After understanding the problem, you should immediately decide whether you will be able to solve the problem or need to escalate. Whatever your conclusion, assure the customer politely and convincingly that his problem will be solved.

Thinking Out of the Box

Having a problem-solving attitude is essential for a customer service associate. As most customers are inarticulate, out-of-the-box thinking is crucial to solve the problems. Also, even after the problem has been defined, it is not necessary that it has a straightforward solution. You may have to approach the problem from a unique way that has not been tried earlier.

Here are some scenarios where you may need to be innovative to offer a solution −

- Not covered specifically in company or department guideline
- Information needs to be gathered from another department
- Customer is pushy and wants you to solve immediately
- Customer has already made complaints that were not resolved

Going the Extra Mile

Every customer service department has a set of laid down norms, usually written. These norms or rules are there to −

- Define a team member’s responsibilities
- Establish protocols for inter-departmental communication
- Outline what a team member is not required to do

These guidelines are for safeguarding your professional interests. However, no one will stop you from going an extra mile to help a customer. In fact, as the face of your organization, you should do everything in your power to solve a customer’s problems. Even if it means doing something you are not strictly required to do.

A satisfied customer is the best publicity any organization can have. Plus, a satisfied customer will become loyal too, buying your products every time a need arises rather than look for other options.

Section Overview

The initial investigation is the first step in the analysis phase of your project. It is a fact-finding mission in which you aim to find out as much as you can about your user, the current problem and what the user needs from a computer system in order to solve the problem. You will use a variety of methods of fact-finding in order to give you as full a picture as possible.

What you find out from your investigation will form the basis of the rest of the analysis phase and the design phase and it is therefore essential that it is thorough and well-planned. It is extremely difficult to create an appropriate and useful computer system and to write a quality project report without first carrying out a genuine, well-planned and comprehensive initial investigation with a real user or users.

Make sure that you read ahead to the analysis and design phase guidelines so that you have a clear idea of what you are aiming to find out.

As a starting point, you need your investigation to enable you to outline the following:

- Background to the problem/identify the problem
- Identify the user(s) having the problem
- Identify what the user(s) would need in order to be able to solve the problem
- Outline the limitations of the solution you propose
- Identify two or more possible ways of providing a solution
- Choose the best way of providing a solution and explain why it is the best way
- Identify the sources and destinations of the data used by the system and the processes used on the data.
- Identify all the data used by the system data requirements and draw an ER model if appropriate.
- Give a clear, comprehensive, specific list of objectives of the project (what the project will need to do in order to be successful)

Possible Fact-Finding Methods

Observation

- Describe what or who you will observe and how many times.
- Describe what you expect to gain from this.
- Carry out the observation and document what you observed. Include the full transcript of your observation(s) in an appendix and refer to where to find it in the write up of this section. The summary, analysis and conclusions of your findings will be presented in the Analysis section.

Collection of existing documentation

- Describe what documentation you expect to collect and how much of it. It is generally a good idea to collect samples of your user's current forms, reports and other documentation as evidence of the existing data structures and form of storage and of the data flows in the current system.
- Describe what you expect to gain from this.
- In this section include copies of the documentation that you collected.

Special checks (e.g. asking people to record how long it takes to carry out a particular task)

- Describe who will do the tasks and what you will ask them to do.
- Carry out the special checks and document what you found out.

Questionnaires

- Describe who you will give a questionnaire to, what questions you will you ask and how many you will give out.
- Describe what you expect to gain from this. Why are you doing this instead of an interview?
- Devise a sensible (but not complicated questionnaire) and include a blank copy of it in this section.
- Include 2 or 3 examples of the completed questionnaires in an appendix and refer to where to find them in the write up of this section. The summary, analysis and conclusions of the information given by the questionnaires will be presented in the Analysis section.

Interviews

- Describe who you will interview.
- Describe what you expect to gain from this. Why are you doing this instead of using a questionnaire?
- Produce a list of questions that you will ask in the interview. (You may need more than one set of questions if you are interviewing different people for different reasons.)
- Do the interview(s) and document what you found out. Include the full transcript of your interview(s) in an appendix and refer to where to find it in the write up of this section. The summary, analysis and conclusions of your findings from the interview(s) will be presented in the Analysis section.

3.7 Initial Investigation

Expected Contents

(Use these as sub-headings in your project report):

Introduction to the user and their problem

Remember that at this stage you will not know many details as you have not yet carried out a structured investigation so you can be brief as all you are doing in this sub-section is setting the scene.

Description of Investigation

To get a top mark for your project you will need to provide evidence that you have carried out an extensive and structured investigation into the problem.

To do this you will need to document the aims of your investigation, the methods you used, why those methods were suitable for your project and the results you gained from them. Note that lengthy transcripts of observations or interviews and completed questionnaires should be included in an appendix and referred to rather than included in the write-up of this section of the project report.

Use the guidance given in the 'Fact-finding Methods' section above to help you to decide which methods to use and how to document them.

For each method consider whether it is sensible to use in this instance.

If it is sensible, then

- Explain what you plan to do
- Explain why
- Do it
- Document what you did
- Document what you found out (i.e. the results).

If it is not sensible, then show that you made a sensible decision about what was appropriate rather than just not bothering by

- Stating that you did not use the method.
- Explaining why you did not use the method.

3.8 Strategies for determining information requirements

- The systems development process transforms the existing (as is) system into the proposed (to be) system

- Requirements determination

- The single most critical step of the entire SDLC
- Changes can be made easily in this stage
- Most (>50%) system failures are due to problems with requirements
- The iterative process of OOSAD is effective because:
- Small batches of requirements can be identified and implemented incrementally

The system will evolve over time

Requirements Determination

- Purpose: to convert high level business requirements (from the system request) into detailed requirements that can be used as inputs for creating models
- What is a requirement?
  - A statement of what the system must do or a characteristic it must have
  - Will later evolve into a technical description of how the system will be implemented
- Types:
  - Functional: relates to a process or data
  - Non-functional: relates to performance or usability

Requirements Definition

- Functional & non-functional requirements listed in outline format
- May be prioritized
- Provides information needed in subsequent workflows
- Defines the scope of the system

Sample of Requirements Definition

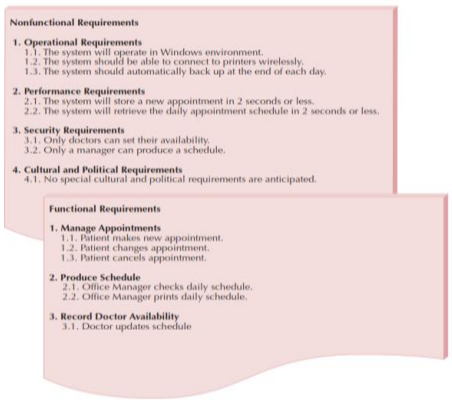


Fig. 3.3 Sample requirement definition

Creating a Requirements Definition

- Determine the types of functional and non-functional requirements applicable to the project
- Use requirements-gathering techniques to collect details
- Analysts work with users to verify, change and prioritize each requirement
- Continue this process through analysis workflow, but be careful of scope creep
- Requirements that meet a need but are not within the current scope can be added to a list of future enhancements

Determining Requirements

- Business & IT personnel need to collaborate
- Strategies for problem analysis:
    - Root cause analysis
    - Duration analysis
    - Activity-based costing
    - Informal benchmarking
    - Outcome analysis
    - Technology analysis

- Activity elimination

- Requirements Analysis Strategies Problem analysis
    - Ask users to identify problems with the current system
    - Ask users how they would solve these problems
    - Good for improving efficiency or ease-of-use
- Root cause analysis
    - Focus is on the cause of a problem, not its solution
    - Create a prioritized list of problems
    - Try to determine their causes
    - Once the causes are known, solutions can be developed
- Duration analysis
    - Determine the time required to complete each step in a business process
    - Compare this to the total time required for the entire process
    - Large differences suggest problems that might be solved by:
        - Integrating some steps together
        - Performing some steps simultaneously (in parallel)
- Activity-based costing
    - Same as duration analysis but applied to costs
- Informal benchmarking
    - Analyzes similar processes in other successful organizations
- Outcome analysis
    - What does the customer want in the end?

- Technology analysis
    - Apply new technologies to business processes & identify benefits
- Activity elimination
    - Eliminate each activity in a business process in a “force-fit” exercise

Problems in Requirements Determination

- Analyst may not have access to the correct users
- Requirements specifications may be inadequate
- Some requirements may not be known in the beginning
- Verifying and validating requirements can be difficult

3.9 Requirements Gathering Techniques

- Process is used to:
    - Uncover all requirements (those uncovered late in the process are more difficult to incorporate)
    - Build support and trust among users
- Which technique(s) to use?
    - Interviews
    - Joint Application Development (JAD)
    - Questionnaires
    - Document analysis
    - Observation

Interviews

- Most popular technique—if you need to know something, just ask
- Process:

- Select people to interview & create a schedule
- Design interview questions (Open-ended, closed-ended, & probing types of questions)
- Prepare for the interview (Unstructured vs. structured interview organized in a logical order)
- Conduct the interview (Top-down vs. bottom-up)
- Follow-up after the interview

Question Types

Types of Questions	Examples
Closed-ended questions	• How many telephone orders are received per day? • How do customers place orders? • What information is missing from the monthly sales report?
Open-ended questions	• What do you think about the current system? • What are some of the problems you face on a daily basis? • What are some of the improvements you would like to see in a new system?
Probing questions	• Why? • Can you give me an example? • Can you explain that in a bit more detail?

Fig. 3.4 Sample Questionary

Interviewing Strategies

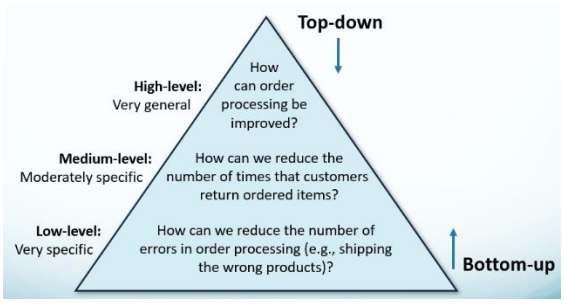
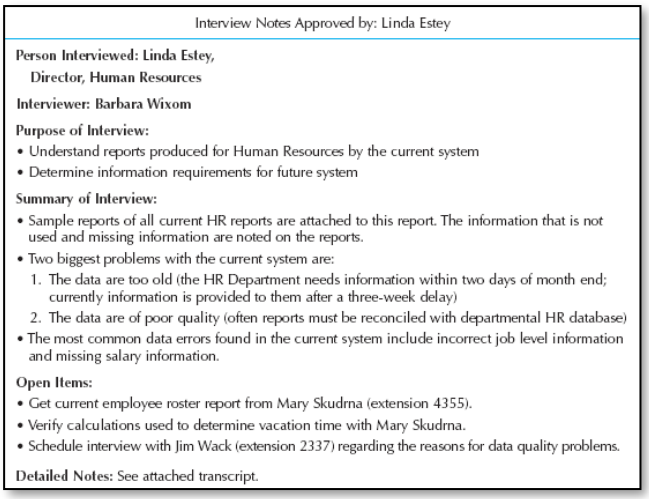


Fig. 3.5 Interview strategies

Post-Interview

- **Prepare notes and send to the interviewee for verification**



Interview Notes Approved by: Linda Estey

Person Interviewed: Linda Estey,
Director, Human Resources

Interviewer: Barbara Wixom

Purpose of Interview:

- Understand reports produced for Human Resources by the current system
- Determine information requirements for future system

Summary of Interview:

- Sample reports of all current HR reports are attached to this report. The information that is not used and missing information are noted on the reports.
- Two biggest problems with the current system are:
  1. The data are too old (the HR Department needs information within two days of month end; currently information is provided to them after a three-week delay)
  2. The data are of poor quality (often reports must be reconciled with departmental HR database)
- The most common data errors found in the current system include incorrect job level information and missing salary information.

Open Items:

- Get current employee roster report from Mary Skudrna (extension 4355).
- Verify calculations used to determine vacation time with Mary Skudrna.
- Schedule interview with Jim Wack (extension 2337) regarding the reasons for data quality problems.

Detailed Notes: See attached transcript.

Fig. 3.6 Post Interview report

Joint Application Development (JAD)

- Joint user-analyst meeting hosted by a facilitator
    - 10 to 20 users
    - 1 to 2 scribes as needed to record the session
    - Usually in a specially prepared room
- Meetings can be held electronically and anonymously
    - Reduces problems in group settings
    - Can be held remotely
- Sessions require careful planning to be successful
    - Users may need to bring documents or user manuals
    - Ground rules should be established

Questionnaires

- A set of written questions used to obtain information from individuals
- May be paper based or electronic (e.g., web based)

- Common uses:
    - Large numbers of people
    - Need both information and opinions
    - When designing for use outside the organization (customers, vendors, etc.)
- Typical response rates: < 50% (paper); < 30% (Web)

Questionnaire Steps

- Select the participants
    - Identify the population
    - Use representative samples for large populations
- Designing the questionnaire
    - Careful question selection
    - Remove ambiguities
- Administering the questionnaire
    - Working to get good response rate
    - Offer an incentive (e.g., a free pen)
- Questionnaire follow-up
    - Send results to participants

  Send a thank-you

Information gathering Tools

What kinds of Information do we need?

Information about the Firm/Organization

•Information about the User Staff

•Information about the Work Flow

Information about the Firm/Organization

- Policies
- Goals
- Objectives
- Organization Structure

Information about the User Staff

•Authority Relationships

•Job Functions

•Information Requirements

•Interpersonal Relationships

Information about the Work Flow

•Work Flow

•Methods and Procedures

•Work Schedules

•DFD or Flowchart

Where does Information Originate?

Information is gathered from two principal resources:

-Personnel or written documents from within the organization

- The organization's environment.

The Primary External sources of information are:

-Vendors

-Government Documents

-Newspaper and Professional Journals

The Primary Internal sources of information are:

- Financial Reports
- Personnel Staff
- Professional Staff (Legal Counsel, auditor etc.)
- System Documentation and Manuals
- The User or User Staff
- Reports and Transaction Documents

On Site Observation

- What kind of system is it? What does it do?
- Who runs the system? Who are the important people in it?
- What is the history of the system? How did it get to its present stage of development?
- What kind of system is it in comparison with other systems in the organization?
- Is it a fast paced or slow system to external crises?

Observation Methods
Natural- A natural observation occurs in a setting such as the employee's place of work.
Contrived- A contrived observation is set up by the observer in a place like a laboratory.

Obtrusive- An obtrusive observation takes place when the respondent knows he/she is being observed

Unobtrusive- An unobtrusive observation takes place in a contrived way such as behind a one way mirror
Direct- A direct observation takes place when the analyst actually observes the subject or the system at work

Indirect- In an indirect observation the analyst uses mechanical devices such as cameras and videotapes to capture information

Structured- In an structured observation the observer looks for and records a specific action

Unstructured- These methods place the observer in a situation to observe whatever be pertinent at that time

Problems in On Site Observation

Intruding into the user's area often results in adverse reactions by the staff, therefore adequate preparation and training are important

•Attitudes and motivations cannot be readily observed

•Observations are subject to error due to the observer's misinterpretation

Unproductive, long hours are often spent in an attempt to observe specific one time activities or events

Interviews
It is a face to face interpersonal role situation, in which a person called the interviewer, asks questions to another person, designed to gather information about a problem.

Advantages of Interview
It is a superior technique used for exploring areas

•It offers better opportunity to evaluate the validity of the information gathered.

•The interviewer can observe not only what they say and how they say.

- •It is an effective technique for eliciting information about complex subjects
- Many people enjoy being interviewed, regardless of the subject

Drawbacks of Interview

The major drawback of an interview is the long preparation time

The art of Interviewing
It is an art.

•The primary requirements for a successful interview are to create friendly atmosphere and to put the respondent at ease.

•The interview proceeds with asking questions properly, obtaining reliable responses, and recording them accurately and completely

Arranging the Interview

It should be arranged so that the physical location, time of interview, and order of interviewing assure privacy and minimal interruption.

•Appointments should be made well in advance and interviewer should adhere to time period.

Guides to a Successful Interview

Following steps should be taken:
•Set the stage for the Interview.

•Establish rapport; put the Interviewee at ease.

•Phrase questions clearly and succinctly.

•Be a good listener; avoid arguments.

•Evaluate the outcome of the interview.

Set the stage for the interview

-This is an ice breaking session, relaxed, informal phase where the analyst opens the interview.

-The analyst adjusts his/her own image to counter that of the interviewee
Establishing Rapport

A. Do not deliberately mislead the user staff about the purpose of the interview.
B. Assure interviewees confidentiality that no information will be released to unauthorized personnel
Avoid showing off your knowledge or sharing information from other sources.
D. Avoid acting like an expert consultant
E. Respect the time schedules
F. Do not promise anything you cannot deliver, such as advice

G. Dress and behave appropriately for the setting

H. Do not interrupt the interviewee
I. Avoid personal involvement in the affairs of the user's department

Asking the questions
Obtaining and recording the response- The analyst should make an effort to obtain information from the user
Data recording and the notebook- Many systems fail because of poor data recording. So special care must be taken to record the data
Questionnaires
These are the questions to which individuals respond
Advantages of Questionnaires
•It is economical and requires less skills to administer than the interview.

•A questionnaire can be administered to large number of individuals simultaneously

•Questionnaires ensure uniformity of questions

•In a questionnaire respondents give opinion without fear
Respondents have time to think the questions over and do calculations to provide more accurate data.

Types of Interviews and Questionnaires

1. The Unstructured Alternative
2. The Structured Alternative

1. The Unstructured Alternative

It is a relatively nondirective information gathering technique.

•It allows respondents to answer questions freely in their own words.

•The responses are spontaneous rather than forced.

•System analyst should encourage the respondent to talk freely

2. The Structured Alternative

The questions are presented with exactly the same wording and in the same order

•Questions may be either closed or open ended.

•An open ended question requires no response direction or specific response
Closed questions are those in which the responses are presented as a set of alternatives. There are five major varieties:

-Fill in the blanks:

-Dichotomous (yes/no type) questions:

-Ranking scales questions

-Multiple choice questions

-Rating scales questions

Part-A			
Unit 3			
Q.No	**Questions**	**Competence**	**BT Level**
1.	Define Project Review	Knowledge	BTL 1
2.	Illustrate the primary purpose for project tracking meetings?	Analyze	BTL 4
3.	List out the attendess of project tracking meetings	Understand	BTL 2
4.	Infer the 4 steps in Escalation Management	Understand	BTL 2
5.	Explain the strategies for determining information requirements	Knowledge	BTL 1
6.	Write the steps in Questionnaire creation	Create	BTL 6
7.	Where does Information Originate?	Knowledge	BTL 1
8.	Define some Information Gathering Tools	Understand	BTL 2
9.	Define Questionnaires	Knowledge	BTL 1
10.	List out the guidelines to Successful Interview	Understand	BTL2
Part-B			
Q.No	**Questions**	**Competence**	**BT Level**
1.	Develop a plan for Recover a project from failiure	Create	BTL 6
2.	"software Quality Assurance is an umbrella activity" justify	Evaluate	BTL 5
3.	Do staff selection relate with quality of product? Justify with appropriate reason.	Evaluate	BT5
4.	Understanding the Customer Problem to solve	Create	BT6
5.	Requirements Gathering Technique	Analyze	BT 4
6.	Interviewing Strategies	Create	BT 6
7.	Product Requirement Types	Understand	BT 6

UNIT – IV -Problem Solving- SBAA3017

UNIT 4 PROBLEM SOLVING

Product Specifications - Defining the Final Product - Data Flow Diagram, Data Dictionary, Structured English, Decision Trees, Decision Tables - Feasibility Study. Software Testing : Test Plan - Development Testing : Verification and Validation - General Testing Methods : White Box and Black Box Testing - Unit Testing - System Integration Testing - Validation Testing - System testing.

4.3 Data Flow Diagram (DFD)

Although the *data flow diagram* (DFD) and related diagrams and information are not a formal part of UML, they can be used to complement UML diagrams and provide additional insight into system requirements and flow. The DFD takes an input-process-output view of a system. That is, data objects flow into the software, are transformed by processing elements, and resultant data objects flow out of the software. Data objects are represented by labeled arrows, and transformations are represented by circles (also called bubbles). The DFD is presented in a hierarchical fashion. That is, the first data flow model (sometimes called a level 0 DFD or *context diagram*) represents the system as a whole. Subsequent data flow diagrams refine the context diagram, providing increasing detail with each subsequent level.

What are the steps? Flow-oriented modeling provides an indication of how data objects are transformed by processing functions. Behavioral modeling depicts the states of the system and its classes and the impact of events on these states. Pattern-based modeling makes use of existing domain knowledge to facilitate requirements analysis. WebApp requirements models are especially adapted for the representation of content, interaction, function, and configurationrelated requirements.

What is the work product? A wide array of textbased and diagrammatic forms may be chosen for the requirements model. Each of these representations provides a view of one or more of the model elements.

How do I ensure that I've done it right? Requirements modeling work products must be reviewed for correctness, completeness, and consistency. They must reflect the needs of all stakeholders and establish a foundation from which design can be conducted.

Creating a Data Flow Model

A few simple guidelines can aid immeasurably during the derivation of a data flow diagram: (1) the level 0 data flow diagram should depict the software/system as a single bubble; (2) primary input and output should be carefully noted; (3) refinement should begin by isolating candidate processes, data objects, and data stores to be represented at the next level; (4) all arrows and bubbles should be labeled with meaningful names; (5) *information flow continuity* must be maintained from level to level,2 and (6) one bubble at a time should be refined. There is a natural tendency to overcomplicate the data flow diagram. This occurs when you attempt to show too much detail too early or represent procedural aspects of the software in lieu of

information flow. To illustrate the use of the DFD and related notation, we again consider the *SafeHome* security function. A level 0 DFD for the security function is shown in Figure 7.1. The primary *external entities* (boxes) produce information for use by the system and consume information generated by the system. The labeled arrows represent data objects or data object hierarchies. For example, **user commands and data** encompasses all configuration commands, all activation/deactivation commands, all miscellaneous interactions, and all data that are entered to qualify or expand a command.

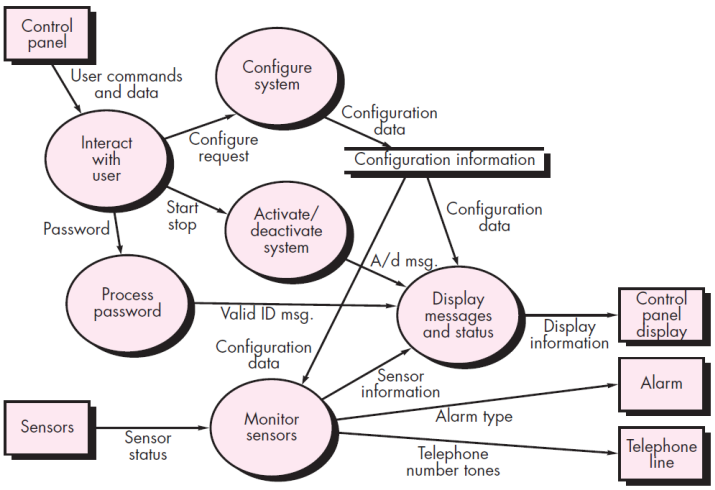


Fig. 4.1 Level 1 DFT

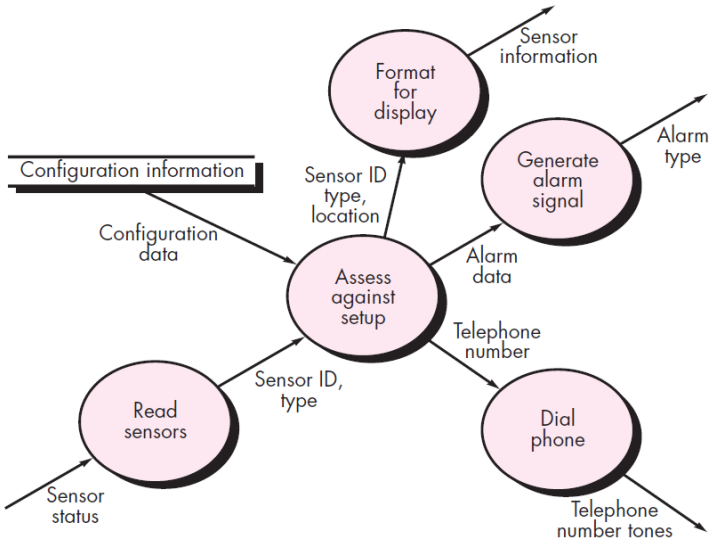


Fig. 4.2 Level 2 DFT

4.4 Data Dictionaries

Overview :

Data Dictionary is the major component in the structured analysis model of the system. A data dictionary in Software Engineering means a file or a set of files that includes a database's metadata (hold records about other objects in the database), like data ownership, relationships of the data to another object, and some other data.

Components of Data Dictionary :

In Software Engineering, the data dictionary contains the following information as follows.

- Name of the item – It can be your choice.
- Aliases – It represents another name.
- Description – Description of what actual text is all about.
- Related data items – Relationship with other data items.
- Range of values – It will represent all possible answers.

Data Dictionary Notations tables :

The Notations used within the data dictionary are given in the table below as follows.

Notations	Meaning
X = a+b	X consists data elements a and b.
X = [a/b]	X consists of either elements a or b.
X = a X	X consists of optimal data elements a.
X = y[a]	X consists of y or more events of data element a
X = [a] z	X consists of z or less events of data element a
X = y [a] z	X consists of some events of data elements between y and z.

Fig. 4.3 data dictionary sample

Features of Data Dictionary :

Here, we will discuss some features of the data dictionary as follows.

It helps in designing test cases and designing the software.

It is very important for creating an order list from a subset of the items list.

It is very important for creating an order list from a complete items list.

The data dictionary is also important to find the specific data item object from the list.

Uses of Data Dictionary :

Here, we will discuss some use cases of the data dictionary as follows.

Used for creating the ordered list of data items

Used for creating the ordered list of a subset of the data items

Used for Designing and testing of software in Software Engineering

Used for finding data items from a description in Software Engineering

4.5 Structured English

Program design language (PDL), also called *structured English* or *pseudocode*, incorporates the logical structure of a programming language with the free-form expressive of a natural language (e.g., English). Narrative text (e.g., English) is embedded within a programming language-like syntax. Automated tools can be used to enhance the application of PDL. A basic PDL syntax should include constructs for component definition, interface description, data declaration, block structuring, condition constructs, repetition constructs, and input-output (I/O) constructs. It should be noted that PDL can be extended to include keywords for multitasking and/or concurrent processing, interrupt handling, interprocess synchronization, and many other features. The application design for which PDL is to be used should dictate the final form for the design language. The format and semantics for some of these PDL constructs are presented in the example that follows.

To illustrate the use of PDL, consider a procedural design for the *SafeHome* security function discussed in earlier chapters. The system monitors alarms for fire, smoke, burglar, water, and temperature (e.g., the heating system fails while the
homeowner is away during winter) and produces an alarm bell and calls a monitoring service, generating a voice-synthesized message.Recall that PDL is *not* a programming language. You can adapt as required without worry about syntax errors. However, the design for the monitoring software would have to be reviewed (do you see any problems?) and further refined before code could be written. The following PDL8 provides an elaboration of the procedural design for an early version of an alarm management component.

```
The intent of this component is to manage control panel switches and input from sensors by
type and to act on any alarm condition that is encountered.
    set default values for systemStatus (returned value), all data items
    initialize all system ports and reset all hardware
    check controlPanelSwitches (cps)
        if cps = "test" then invoke alarm set to "on"
        if cps = "alarmOff" then invoke alarm set to "off"
        if cps = "newBoundingValue" then invoke keyboardInput
        if cps = "burglarAlarmOff" invoke deactivateAlarm;
        •
        •
        •
        default for cps = none
    reset all signalValues and switches
    do for all sensors
        invoke checkSensor procedure returning signalValue
        if signalValue > bound [alarmType]
            then phoneMessage = message [alarmType]
                set alarmBell to "on" for alarmTimeSeconds
                set system status = "alarmCondition"
                parbegin
                    invoke alarm procedure with "on", alarmTimeSeconds;
                    invoke phone procedure set to alarmType, phoneNumber
                endpar
            else skip
        endif
    enddofor
end alarmManagement
```

Note that the designer for the **alarmManagement** component has used the construct parbegin … parend that specifies a parallel block. All tasks specified within the parbegin block are executed in parallel. In this case, implementation details are not considered.

4.6 Decision Tree

In many software application areas, it is often more cost effective to acquire rather than develop computer software. Software engineering managers are faced with a make/ buy decision that can be further complicated by a number of acquisition options:
(1) software may be purchased (or licensed) off-the-shelf
(2) "full-experience" or"partial-experience" software components may be acquired and then modified and integrated to meet specific needs,
(3) software may be custom
built by an outside contractor to meet the purchaser's specifications.
The steps involved in the acquisition of software are defined by the criticality of the software to be purchased and the end cost. In some cases (e.g., low-cost PC software), it is less expensive to purchase and experiment than to conduct a lengthy evaluation of potential software packages. In the final analysis, the make/buy decision is made
based on the following conditions:
(1) Will the delivery date of the software product be
sooner than that for internally developed software?
(2) Will the cost of acquisition plus
the cost of customization be less than the cost of developing the software internally?
(3) Will the cost of outside support (e.g., a maintenance contract) be less than the cost of internal support? These conditions apply for each of the acquisition options.

Creating a Decision Tree

The steps just described can be augmented using statistical techniques such as decision tree analysis.16 For example, Figure 4.5 depicts a decision tree for a softwarebased system X. In this case, the software engineering organization can
(1) build system X from scratch,
(2) reuse existing partial-experience components to construct the system,
(3) buy an available software product and modify it to meet local needs, or
(4) contract the software development to an outside vendor.

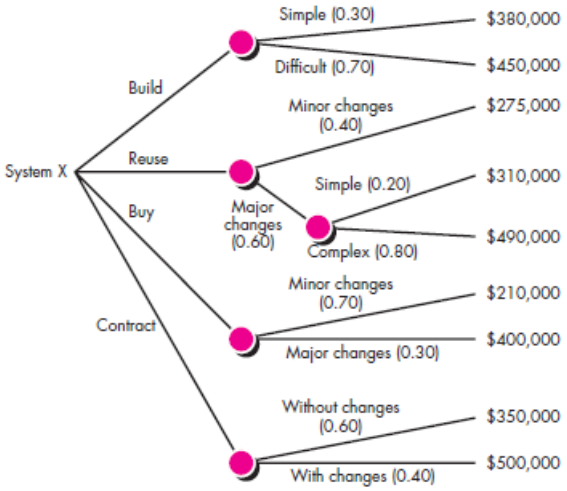


Fig. 4.5 sample decision tree for system X

If the system is to be built from scratch, there is a 70 percent probability that the job will be difficult. Using the estimation techniques discussed earlier in this chapter, the project planner estimates that a difficult development effort will cost $450,000. A "simple" development effort is estimated to cost $380,000. The expected value for cost, computed along any branch of the decision tree, is

$$\text{Expected cost} = \Sigma \text{ (path probability)}_i \times \text{(estimated path cost)}_i$$

where i is the decision tree path. For the build path,

$$\text{Expected cost}_{\text{build}} = 0.30 \text{ (\$380K)} + 0.70 \text{ (\$450K)} = \$429\text{K}$$

Following other paths of the decision tree, the projected costs for reuse, purchase, and contract, under a variety of circumstances, are also shown. The expected costs for these paths are

$$\text{Expected cost}_{\text{reuse}} = 0.40 \text{ (\$275K)} + 0.60 \text{ [0.20 (\$310K)} + 0.80 \text{ (\$490K)]} = \$382\text{K}$$
$$\text{Expected cost}_{\text{buy}} = 0.70 \text{ (\$210K)} + 0.30 \text{ (\$400K)} = \$267\text{K}$$
$$\text{Expected cost}_{\text{contract}} = 0.60 \text{ (\$350K)} + 0.40 \text{ (\$500K)} = \$410\text{K}$$

Based on the probability and projected costs that have been noted in Figure 26.8, the lowest expected cost is the "buy" option. It is important to note, however, that many criteria—not just cost— must be considered during the decision-making process. Availability, experience of the developer/ vendor/contractor, conformance to requirements, local "politics," and the likelihood of change are but a few of the criteria that may affect the ultimate decision to build, reuse, buy, or contract.

4.7 Decision Tables

In many software applications, a module may be required to evaluate a complecombination of conditions and select appropriate actions based on these conditions. *Decision tables* provide a notation that translates actions and conditions
(described in a processing narrative or a use case) into a tabular form. The table isdifficult to misinterpret and may even be used as a machine-readable input to a table-driven algorithm. Decision table organization is illustrated in Figure Referring to the figure, the table is divided into four sections. The upper left-hand quadrant contains a list of all conditions. The lower left-hand quadrant contains a list of all actions that are possible based on combinations of conditions. The right-hand quadrants form a matrix that indicates condition combinations and the corresponding actions that will occur for a specific combination. Therefore, each column of the matrix may be interpreted as a *processing rule.* The following steps are applied to develop a decision table:

1. List all actions that can be associated with a specific procedure (or component).
2. List all conditions (or decisions made) during execution of the procedure.
3. Associate specific sets of conditions with specific actions, eliminating impossible combinations of conditions; alternatively, develop every possible permutation of conditions.
4. Define rules by indicating what actions occur for a set of conditions.

	Rules					
Conditions	**1**	**2**	**3**	**4**	**5**	**6**
Regular customer	T	T				
Silver customer			T	T		
Gold customer					T	T
Special discount	F	T	F	T	F	T
Actions						
No discount	✓					
Apply 8 percent discount			✓	✓		
Apply 15 percent discount					✓	✓
Apply additional *x* percent discount		✓		✓		✓

Fig. 4.6 Decision table

To illustrate the use of a decision table, consider the following excerpt from a informal use case that has just been proposed for the print shop system: Three types of customers are defined: a regular customer, a silver customer, and a gold customer (these types are assigned by the amount of business the customer does with the print shop over a 12 month period). A regular customer receives normal print rates and delivery. A silver customer gets an 8 percent discount on all quotes and is placed ahead of all regular customers in the job queue. A gold customer gets a 15 percent reduction in quoted prices and is placed ahead of both regular and silver customers in the job queue. A special discount of *x* percent in addition to other discounts can be applied to any customer's quote at the discretion of management.

4.8 Feasibility Study

Feasibility Study in Software Engineering is a study to evaluate feasibility of proposed project or system. Feasibility study is one of stage among important four stages of Software Project Management Process. As name suggests feasibility study is the feasibility analysis or it is a measure of the software product in terms of how much beneficial product development will be for the organization in a practical point of view. Feasibility study is carried out based

on many purposes to analyze whether software product will be right in terms of development, implantation, contribution of project to the organization etc.

Types of Feasibility Study :

The feasibility study mainly concentrates on below five mentioned areas. Among these Economic Feasibility Study is most important part of the feasibility analysis and Legal Feasibility Study is less considered feasibility analysis.

Technical Feasibility

In Technical Feasibility current resources both hardware software along with required technology are analyzed/assessed to develop project. This technical feasibility study gives report whether there exists correct required resources and technologies which will be used for project development. Along with this, feasibility study also analyzes technical skills and capabilities of technical team, existing technology can be used or not, maintenance and up-gradation is easy or not for chosen technology etc.

Operational Feasibility

In Operational Feasibility degree of providing service to requirements is analyzed along with how much easy product will be to operate and maintenance after deployment. Along with this other operational scopes are determining usability of product, Determining suggested solution by software development team is acceptable or not etc.

Economic Feasibility

In Economic Feasibility study cost and benefit of the project is analyzed. Means under this feasibility study a detail analysis is carried out what will be cost of the project for development which includes all required cost for final development like hardware and software resource required, design and development cost and operational cost and so on. After that it is analyzed whether project will be beneficial in terms of finance for organization or not.

Legal Feasibility

In Legal Feasibility study project is analyzed in legality point of view. This includes analyzing barriers of legal implementation of project, data protection acts or social media laws, project certificate, license, copyright etc. Overall it can be said that Legal Feasibility Study is study to know if proposed project conform legal and ethical requirements.

Schedule Feasibility

In Schedule Feasibility Study mainly timelines/deadlines is analyzed for proposed project which includes how many times teams will take to complete final project which has a great impact on the organization as purpose of project may fail if it can’t be completed on time.

Feasibility Study Process :

The below steps are carried out during entire feasibility analysis.

1. Information assessment
2. Information collection
3. Report writing
4. General information

Need of Feasibility Study :

Feasibility study is so important stage of Software Project Management Process as after completion of feasibility study it gives a conclusion of whether to go ahead with proposed project as it is practically feasible or to stop proposed project here as it is not right/feasible to develop or to think/analyze about proposed project again.
Along with this Feasibility study helps in identifying risk factors involved in developing and deploying system and planning for risk analysis also narrows the business alternatives and enhance success rate analyzing different parameters associated with proposed project development.

4.9 Software Testing

4.9.1 Test Planning

The use of the word *planning* (in any context) is anathema to some Web developers.These developers don't plan; they just start—hoping that a killer WebApp will emerge. A more disciplined approach recognizes that planning establishes a road map for all work that follows. It's worth the effort. In their book on WebApp testing, Except for the simplest of Web sites, it quickly becomes apparent that some sort of test planning is needed. All too often, the initial number of bugs found from ad hoc testing is large enough that not all of them are fixed the first time they're detected. This puts an additional burden on people who test Web sites and applications. Not only must they conjure up imaginative new tests, but they must also remember how previous tests were executed in order to reliably re-test the Web site/application, and ensure that known bugs have been removed and that no new bugs have been introduced.

The questions you should ask are: How do we "conjure up imaginative new tests," and what should those tests focus on? The answers to these questions are contained within a test plan. A WebApp test plan identifies

(1) the task set2 to be applied as testing commences,
(2) the work products to be produced as each testing task is executed, and
(3) the manner in which the results of testing are evaluated, recorded, and reused when regression testing is conducted. In some cases, the test plan is integrated with the project plan. In others, the test plan is a separate document.

4.9.2 Development Testing

It is a method of applying testing practices consistently throughout the software development life cycle process. This testing ensures the detection of bugs or errors at the right time which further ensures delay of any kind of risk in terms of time and cost. Development Testing aims

to establish a framework to verify whether the requirements of a given project are met in accordance with the rules of the mission to be accomplished. This testing is performed by the software developers or other engineers during the construction phase of the software development lifecycle (SDLC). Development Testing is a continuous or a running process in the development of a product in the entire software development life cycle. This testing is done only once as compared to other testings which can be performed many times. To meet the deadline date, development testing is performed during the development phase of a software product,
In Development Testing, the phases are more tightly integrated so that code that is being written and checked in is automatically tested. In this way, the problems can be more quickly discovered and can be addressed.

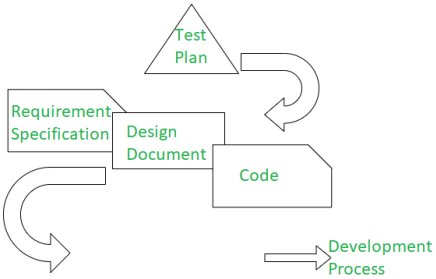


Fig. 4.7 development testing

When writing new code or building a new software product.

When development cost is low, the client should perform development Testing so that the client doesn't have to face the debugging and another testing cost.

Development testing requires some metric depending upon the organization to organization, and these may include the following :

1. Static code Analysis : Static code analysis is a technique of debugging by analyzing the source code before running a program. It is carried out by analyzing a set of code against a set or multiple sets of coding rules. This involves analyzing the source code, without actually executing the program.
By performing static code analysis, the developers will know early on if there are any problems in their code and by this, it will be easier to fix those problems.

2. **Data Flow Analysis :** This concept uses the Control flow Graph mechanism to check the flow of the program, at different levels. Data Flow Testing is a type of structural testing. It is a method that is used to find the test paths of a program according to the locations of definitions and uses of variables in the program. It has nothing to do with data flow diagrams. This testing uses the control flow graph to check the anomalies in the code which can interrupt the flow of the program.

3. **Metric Analysis :** Metric is a synonym for measurement. To calculate the efficiency of a program, various software metrics like calculating cyclomatic complexity, counting Lines of code (LOC), function points, etc are used in that case. In metric analysis, Test metrics are used in taking the decision for the next phase of activities like cost estimating & future

projects, recognizing the type of improvement required to succeed the project, or in taking a decision on the process or technology to be modified, etc.

4. Code review : The source code is inspected and is checked for any flaws in it. It can be used to find and remove flaws in the code such as memory leaks and buffer overflows. It is very important to do a code review in the early phase like a peer review, carry out this step earlier than you send your code to be tested for development. Also, do some functionality testing of your code so that it becomes easy in code review. There are various approaches to do code reviews such as The Email thread, Pair programming, Over shoulder, and Tool-assisted.

5. Benefits of Development Testing :

- Helps in increasing software development life cycle (SDLC) efficiency
- Helps in reducing software errors
- Speed up the delivery process
- It gives high-quality code at any time as the code is being tested continuously
- It requires less time to deploy in the market for new features

Disadvantages of Development Testing :

- This testing can be time-consuming as it is done during the entire phase of a software product

Tools required for Development Testing :

For conducting testing of a software product, there are many automated testing tools are available in the market and some of them I have given below. The most commonly used tools for software testing are:

- Selenium
- TestingWhiz
- HPE Unified Functional Testing
- Watir
- Katalon Studio

4.9.3 Verification and Validation

Software testing is one element of a broader topic that is often referred to as verification and validation (V&V). *Verification* refers to the set of tasks that ensure that software correctly implements a specific function. *Validation* refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements. Boehm [Boe81] states this another way:

Verification: "Are we building the product right?"
Validation: "Are we building the right product?"

Verification and validation includes a wide array of SQA activities: technical reviews, quality and configuration audits, performance monitoring, simulation, feasibility study, documentation review, database review, algorithm analysis, development testing, usability testing, qualification testing, acceptance testing, and installation testing. Although testing plays an extremely important role in V&V, many other activities are also necessary. Testing does provide the last bastion from which quality can be assessed and, more pragmatically, errors can be uncovered. But testing should not be viewed as a safety net. As they say, "You can't test in quality. If it's not there before you begin testing, it won't be there when you're finished testing."

Quality is incorporated into software throughout the process of software engineering. Proper application of methods and tools, effective technical reviews, and solid management and measurement all lead to quality that is confirmed during testing. Miller [Mil77] relates software testing to quality assurance by stating that "the underlying motivation of program testing is to affirm software quality with methods that can be economically and effectively applied to both large-scale and small-scale systems."

4.10 General Testing Methods

4.10.1 White Box Testing

White Box Testing is software testing technique in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design, usability and security. In white box testing, code is visible to testers so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing and Glass box testing.
It is one of two parts of the Box Testing approach to software testing. Its counterpart, Blackbox testing, involves testing from an external or end-user type perspective. On the other hand, White box testing in software engineering is based on the inner workings of an application and revolves around internal testing.

The term "WhiteBox" was used because of the see-through box concept. The clear box or WhiteBox name symbolizes the ability to see through the software's outer shell (or "box") into its inner workings. Likewise, the "black box" in "Black Box Testing" symbolizes not being able to see the inner workings of the software so that only the end-user experience can be tested.

What do you verify in White Box Testing?

White box testing involves the testing of the software code for the following:

- Internal security holes
- Broken or poorly structured paths in the coding processes
- The flow of specific inputs through the code
- Expected output
- The functionality of conditional loops
- Testing of each statement, object, and function on an individual basis

The testing can be done at system, integration and unit levels of software development. One of the basic goals of whitebox testing is to verify a working flow for an application. It involves testing a series of predefined inputs against expected or desired outputs so that when a specific input does not result in the expected output, you have encountered a bug.

How do you perform White Box Testing?

To give you a simplified explanation of white box testing, we have divided it into **two basic steps**. This is what testers do when testing an application using the white box testing technique:

Step 1) UNDERSTAND THE SOURCE CODE

The first thing a tester will often do is learn and understand the source code of the application. Since white box testing involves the testing of the inner workings of an application, the tester must be very knowledgeable in the programming languages used in the applications they are testing. Also, the testing person must be highly aware of secure coding practices. Security is often one of the primary objectives of testing software. The tester should be able to find security issues and prevent attacks from hackers and naive users who might inject malicious code into the application either knowingly or unknowingly.

Step 2) CREATE TEST CASES AND EXECUTE

The second basic step to white box testing involves testing the application's source code for proper flow and structure. One way is by writing more code to test the application's source code. The tester will develop little tests for each process or series of processes in the application. This method requires that the tester must have intimate knowledge of the code and is often done by the developer. Other methods include Manual Testing, trial, and error testing and the use of testing tools as we will explain further on in this article.

White Box Testing Techniques

A major White box testing technique is Code Coverage analysis. Code Coverage analysis eliminates gaps in a Test Case suite. It identifies areas of a program that are not exercised by a set of test cases. Once gaps are identified, you create test cases to verify untested parts of the code, thereby increasing the quality of the software product

There are automated tools available to perform Code coverage analysis. Below are a few coverage analysis techniques a box tester can use:

Statement Coverage:- This technique requires every possible statement in the code to be tested at least once during the testing process of software engineering.

Branch Coverage – This technique checks every possible path (if-else and other conditional loops) of a software application.

Apart from above, there are numerous coverage types such as Condition Coverage, Multiple Condition Coverage, Path Coverage, Function Coverage etc. Each technique has its own merits and attempts to test (cover) all parts of software code. **Using Statement and Branch coverage you generally attain 80-90% code coverage which is sufficient.**

Following are important WhiteBox Testing Techniques:

- Statement Coverage
- Decision Coverage
- Branch Coverage
- Condition Coverage
- Multiple Condition Coverage
- Finite State Machine Coverage
- Path Coverage
- Control flow testing
- Data flow testing

Types of White Box Testing

White box testing encompasses several testing types used to evaluate the usability of an application, block of code or specific software package. There are listed below —

- **Unit Testing:** It is often the first type of testing done on an application. Unit Testing is performed on each unit or block of code as it is developed. Unit Testing is essentially done by the programmer. As a software developer, you develop a few lines of code, a single function or an object and test it to make sure it works before continuing Unit Testing helps identify a majority of bugs, early in the software development lifecycle. Bugs identified in this stage are cheaper and easy to fix.
- **Testing for Memory Leaks**: Memory leaks are leading causes of slower running applications. A QA specialist who is experienced at detecting memory leaks is essential in cases where you have a slow running software application.

Apart from above, a few testing types are part of both black box and white box testing. They are listed as below

- **White Box Penetration Testing:** In this testing, the tester/developer has full information of the application's source code, detailed network information, IP addresses involved and all server information the application runs on. The aim is to attack the code from several angles to expose security threats
- **White Box Mutation Testing**: Mutation testing is often used to discover the best coding techniques to use for expanding a software solution.

White Box Testing Tools

Below is a list of top white box testing tools.

- Parasoft Jtest
- EclEmma
- NUnit
- PyUnit
- HTMLUnit
- CppUnit

Advantages of White Box Testing

- Code optimization by finding hidden errors.
- White box tests cases can be easily automated.
- Testing is more thorough as all code paths are usually covered.
- Testing can start early in SDLC even if GUI is not available.

Disadvantages of WhiteBox Testing

- White box testing can be quite complex and expensive.
- Developers who usually execute white box test cases detest it. The white box testing by developers is not detailed can lead to production errors.
- White box testing requires professional resources, with a detailed understanding of programming and implementation.
- White-box testing is time-consuming, bigger programming applications take the time to test fully.

4.10.2 Black box testing

Black box testing is a type of software testing in which the functionality of the software is not known. The testing is done without the internal knowledge of the products.

Black box testing can be done in following ways:

1. Syntax Driven Testing – This type of testing is applied to systems that can be syntactically represented by some language. For example- compilers,language that can be represented by context free grammar. In this, the test cases are generated so that each grammar rule is used at least once.

2. Equivalence partitioning – It is often seen that many type of inputs work similarly so instead of giving all of them separately we can group them together and test only one input of each group. The idea is to partition the input domain of the system into a number of equivalence classes such that each member of class works in a similar way, i.e., if a test case in one class results in some error, other members of class would also result into same error.
The technique involves two steps:

1. **Identification of equivalence class –** Partition any input domain into minimum two sets: **valid values** and **invalid values**. For example, if the valid range is 0 to 100 then select one valid input like 49 and one invalid like 104.
2. **Generating test cases –**
   (i) To each valid and invalid class of input assign unique identification number.
   (ii) Write test case covering all valid and invalid test case considering that no two invalid inputs mask each other.

   To calculate the square root of a number, the equivalence classes will be:
   (a) Valid inputs:
   - Whole number which is a perfect square- output will be an integer.
   - Whole number which is not a perfect square- output will be decimal number.
   - Positive decimals

   (b) Invalid inputs:
   - Negative numbers(integer or decimal).
   - Characters other that numbers like “a”,”!”,”;”,etc.

3. Boundary value analysis – Boundaries are very good places for errors to occur. Hence if test cases are designed for boundary values of input domain then the efficiency of testing improves and probability of finding errors also increase. For example – If valid range is 10 to 100 then test for 10,100 also apart from valid and invalid inputs.

4. Cause effect Graphing – This technique establishes relationship between logical input called causes with corresponding actions called effect. The causes and effects are represented using Boolean graphs. The following steps are followed:

1. Identify inputs (causes) and outputs (effect).
2. Develop cause effect graph.
3. Transform the graph into decision table.
4. Convert decision table rules to test cases.

For example, in the following cause effect graph:

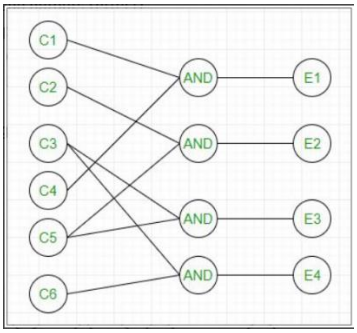


Fig. 4.8 Cause effect graph

It can be converted into decision table like:

		1	2	3	4
CAUSES	C1	1	0	0	0
	C2	0	1	0	0
	C3	0	0	1	1
	C4	1	0	0	0
	C5	0	1	1	0
	C6	0	0	0	1
EFFECTS	E1	X	-	-	-
	E2	-	X	-	-
	E3	-	-	X	-
	E4	-	-	-	X

Fig.9 Decision table

Each column corresponds to a rule which will become a test case for testing. So there will be 4 test cases.

5. Requirement based testing – It includes validating the requirements given in SRS of software system.

6. Compatibility testing – The test case result not only depend on product but also infrastructure for delivering functionality. When the infrastructure parameters are changed it is still expected to work properly. Some parameters that generally affect compatibility of software are:

1. Processor (Pentium 3,Pentium 4) and number of processors.
2. Architecture and characteristic of machine (32 bit or 64 bit).
3. Back-end components such as database servers.
4. Operating System (Windows, Linux, etc).

4.10.3 Unit Testing

Unit Testing is a software testing technique by means of which individual units of software i.e. group of computer program modules, usage procedures and operating procedures are tested to determine whether they are suitable for use or not. It is a testing method using which every independent modules are tested to determine if there are any issue by the developer himself. It is correlated with functional correctness of the independent modules.
Unit Testing is defined as a type of software testing where individual components of a software aretested.
Unit Testing of software product is carried out during the development of an application. An individual component may be either an individual function or a procedure. Unit Testing is typically performed by the developer.
In SDLC or V Model, Unit testing is first level of testing done before integration testing. Unit testing is such type of testing technique that is usually performed by the developers. Although due to reluctance of developers to tests, quality assurance engineers also do unit testing.

Objective of Unit Testing:

The objective of Unit Testing is:

1. To isolate a section of code.
2. To verify the correctness of code.
3. To test every function and procedure.
4. To fix bug early in development cycle and to save costs.
5. To help the developers to understand the code base and enable them to make changes quickly.
6. To help for code reuse

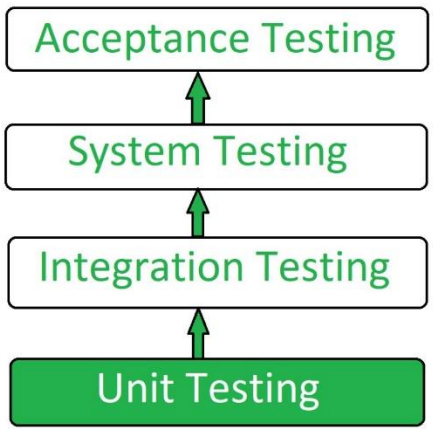


Fig. 4.10 Unit testing

Types of Unit Testing:
There are 2 type of Unit Testing: **Manual**, and **Automated**.

Workflow of Unit Testing:

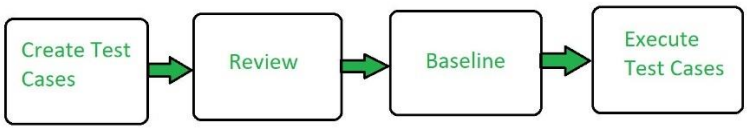


Fig. 4.11 Workflow of Unit Testing

Unit Testing Tools:

Here are some commonly used Unit Testing tools:

1. Jtest
2. Junit
3. NUnit
4. EMMA
5. PHPUnit

Advantages of Unit Testing:

- Unit Testing allows developers to learn what functionality is provided by a unit and how to use it to gain a basic understanding of the unit API.
- Unit testing allows the programmer to refine code and make sure the module works properly.
- Unit testing enables to test parts of the project without waiting for others to be completed.

4.10.4 Integration Testing

Because object-oriented software does not have a hierarchical control structure, conventional top-down and bottom-up integration strategies have little meaning. In addition, integrating operations one at a time into a class (the conventional incremental integration approach) is often impossible because of the "direct and indirect interactions of the components that make up the class" [Ber93]. There are two different strategies for integration testing of OO systems. The first, *thread-based testing,* integrates the set of classes required to respond to one input or event for the system. Each thread is integrated and tested individually. Regression testing is applied to ensure that no side effects occur. The second integration approach, *use-based testing,* begins the construction of the system by testing those classes (called *independent classes*) that use very few (if any) of server classes. After the independent classes are tested, the next layer of classes, called *dependent classes,* that use the independent classes are tested. This sequence of testing layers of dependent classes continues until the entire system is constructed. Unlike conventional integration, the use of driver and stubs (Chapter 18) as replacement operations is to be avoided, when possible. *Cluster testing* is one step in the integration testing of OO software. Here, a cluster of collaborating classes (determined by examining the CRC and objectrelationship model) is exercised by designing test cases that attempt to uncover errors in the collaborations.

4.10.5 Validation Testing

The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements.

Validation Testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment.

It answers to the question, Are we building the right product?

Validation Testing - Workflow:

Validation testing can be best demonstrated using V-Model. The Software/product under test is evaluated during this type of testing.

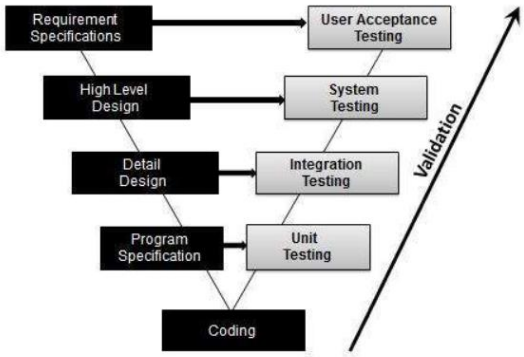


Fig. 4.12 Work flow of validation testing

Activities:

- Unit Testing
- Integration Testing
- System Testing
- User Acceptance Testing

4.10.6 System testing

System Testing is a type of software testing that is performed on a complete integrated system to evaluate the compliance of the system with the corresponding requirements.
In system testing, integration testing passed components are taken as input. The goal of integration testing is to detect any irregularity between the units that are integrated together. System testing detects defects within both the integrated units and the whole system. The result of system testing is the observed behavior of a component or a system when it is tested.

System Testing is carried out on the whole system in the context of either system requirement specifications or functional requirement specifications or in the context of both. System testing tests the design and behavior of the system and also the expectations of the customer. It is performed to test the system beyond the bounds mentioned in the software requirements specification (SRS).

System Testing is basically performed by a testing team that is independent of the development team that helps to test the quality of the system impartial. It has both functional and non-functional testing.

System Testing is a black-box testing.

System Testing is performed after the integration testing and before the acceptance testing.

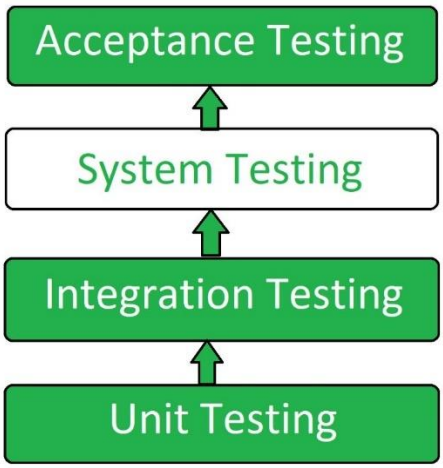


Fig 4.13 System tesing in blackbox testing

System Testing Process:
System Testing is performed in the following steps:

- **Test Environment Setup:**
  Create testing environment for the better quality testing.
- **Create Test Case:**
  Generate test case for the testing process.
- **Create Test Data:**
  Generate the data that is to be tested.
- **Execute Test Case:**
  After the generation of the test case and the test data, test cases are executed.
- **Defect Reporting:**
  Defects in the system are detected.
- **Regression Testing:**
  It is carried out to test the side effects of the testing process.
- **Log Defects:**
  Defects are fixed in this step.
- **Retest:**
  If the test is not successful, then again test is performed.

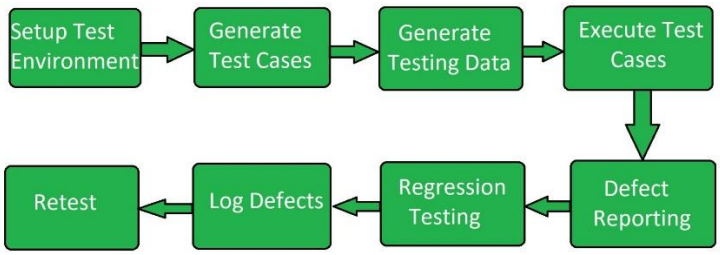


Fig. 4.14 Test evaluation

Types of System Testing:

- **Performance Testing:**
  Performance Testing is a type of software testing that is carried out to test the speed, scalability, stability and reliability of the software product or application.
- **Load Testing:**
  Load Testing is a type of software Testing which is carried out to determine the behavior of a system or software product under extreme load.
- **Stress Testing:**
  Stress Testing is a type of software testing performed to check the robustness of the system under the varying loads.
- **Scalability Testing:**
  Scalability Testing is a type of software testing which is carried out to check the performance of a software application or system in terms of its capability to scale up or scale down the number of user request load.

Part-A unit 4			
Q.No	**Questions**	**Competence**	**BT Level**
1.	**Define Data Dictionaries**	Knowledge	BTL 1
2.	Classify the types of Feasibility study	Understanding	BTL 2
3.	Describe development testing	Knowledge	BTL 1
4.	Identify the testing method which improves the usability and the code is visible to the user.	Analysis	BTL 4
5.	Write down few types of whitebox testing	Understand	BTL 2
6.	Do you agree that decision trees are helpful in risk handling? Compose your views.	Evaluate	BTL 5
7.	Classify few steps in Cause effect Graphing	Analysis	BTL 2
8.	Justify the advantages of Unit testing	Understand	BTL2

9.	Define Structured English	Knowledge	BTL 1
10.	Contrast verification and validation	Analyze	BTL 4
Part-B			
Q.No	**Questions**	**Competence**	**BT Level**
1.	Design your own suitable case study and explain the decision tree method of risk analysis.	Create	BTL 6
2.	Design and explain a Dataflow diagram for safe home	Create	BTL 6
1.	Explain the need for testing in software and describe any two testing methods.	Knowledge	BT 1
2.	Write a short note on Software test plan	Create	BT 6
3.	Write a note on blackbox testing	Analyze	BT 4
4.	**Illustrate Decision tree technique with example and suitable diagram**	Analyze	BT 4
5.	Explain about feasibility Study	Knowledge	BT 1

UNIT – V - SOFTWARE QUALITY – SBAA3017

UNIT 5 SOFTWARE QUALITY

Software Quality - Quality Measures - FURPS - Software Quality Assurance - Software Reviews - Format Technical Review (FTR) Formal Approaches to SQA - Software Reliability - Introduction to SQA - The Software Quality Assurance Plan - Formal approaches to SQA - Clean room Methodology.

5.1 Software Quality

Software quality product is defined in term of its fitness of purpose. That is, a quality product does precisely what the users want it to do. For software products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document. Although "fitness of purpose" is a satisfactory interpretation of quality for many devices such as a car, a table fan, a grinding machine, etc. For software products, "fitness of purpose" is not a wholly satisfactory definition of quality.

Example: Consider a functionally correct software product. That is, it performs all tasks as specified in the SRS document. But, has an almost unusable user interface. Even though it may be functionally right, we cannot consider it to be a quality product.

The modern view of a quality associated with a software product several quality methods such as the following:

Portability: A software device is said to be portable, if it can be freely made to work in various operating system environments, in multiple machines, with other software products, etc.

Software quality product is defined in term of its fitness of purpose. That is, a quality product does precisely what the users want it to do. For software products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document. Although "fitness of purpose" is a satisfactory interpretation of quality for many devices such as a car, a table fan, a grinding machine, etc. For software products, "fitness of purpose" is not a wholly satisfactory definition of quality.

Example: Consider a functionally correct software product. That is, it performs all tasks as specified in the SRS document. But, has an almost unusable user interface. Even though it may be functionally right, we cannot consider it to be a quality product.

The modern view of a quality associated with a software product several quality methods such as the following:

Portability: A software device is said to be portable, if it can be freely made to work in various operating system environments, in multiple machines, with other software products, etc.

Auditing of projects

Review of the quality system

Development of standards, methods, and guidelines, etc.

Production of documents for the top management summarizing the effectiveness of the quality system in the organization.

Evolution of Quality Management System

Quality systems have increasingly evolved over the last five decades. Before World War II, the usual function to produce quality products was to inspect the finished products to remove defective devices. Since that time, quality systems of organizations have undergone through four steps of evolution, as shown in the fig. The first product inspection task gave method to quality control (QC).

Quality control target not only on detecting the defective devices and removes them but also on determining the causes behind the defects. Thus, quality control aims at correcting the reasons for bugs and not just rejecting the products. The next breakthrough in quality methods was the development of quality assurance methods.

The primary premise of modern quality assurance is that if an organization's processes are proper and are followed rigorously, then the products are obligated to be of good quality. The new quality functions include guidance for recognizing, defining, analyzing, and improving the production process.

Total quality management (TQM) advocates that the procedure followed by an organization must be continuously improved through process measurements. TQM goes stages further than quality assurance and aims at frequently process improvement. TQM goes beyond documenting steps to optimizing them through a redesign. A term linked to TQM is Business Process Reengineering (BPR).

BPR aims at reengineering the method business is carried out in an organization. From the above conversation, it can be stated that over the years, the quality paradigm has changed from product assurance to process assurance, as shown in fig.

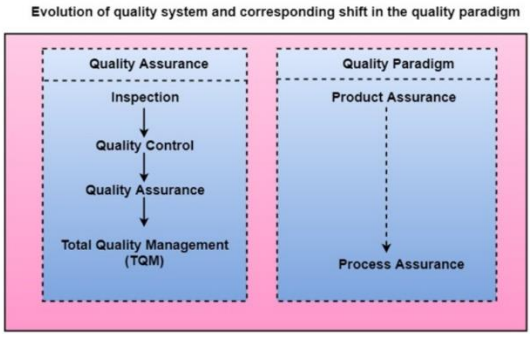


Fig 5.1 Evalution of Quality Management System

5.2 Quality Measures

In Software Engineering, Software Measurement is done based on some Software Metrics where these software metrics are referred to as the measure of various characteristics of a Software.

In Software engineering Software Quality Assurance (SAQ) assures the quality of the software. Set of activities in SAQ are continuously applied throughout the software process. Software Quality is measured based on some software quality metrics.

There is a number of metrics available based on which software quality is measured. But among them, there are few most useful metrics which are most essential in software quality measurement. They are –

1. Code Quality
2. Reliability
3. Performance
4. Usability
5. Correctness
6. Maintainability
7. Integrity
8. Security

Now let's understand each quality metric **in detail –**

1. Code Quality – Code quality metrics measure the quality of code used for the software project development. Maintaining the software code quality by writing Bug-free and semantically correct code is very important for good software project development. In code quality both Quantitative metrics like the number of lines, complexity, functions, rate of bugs generation, etc, and Qualitative metrics like readability, code clarity, efficiency, maintainability, etc are measured.

2. Reliability – Reliability metrics express the reliability of software in different conditions. The software is able to provide exact service at the right time or not is checked. Reliability can be checked using Mean Time Between Failure (MTBF) and Mean Time To Repair (MTTR).

3. Performance – Performance metrics are used to measure the performance of the software. Each software has been developed for some specific purposes. Performance metrics measure the performance of the software by determining whether the software is fulfilling the user requirements or not, by analyzing how much time and resource it is utilizing for providing the service.

4. Usability – Usability metrics check whether the program is user-friendly or not. Each software is used by the end-user. So it is important to measure that the end-user is happy or not by using this software.
5. Correctness – Correctness is one of the important software quality metrics as this checks whether the system or software is working correctly without any error by satisfying the user. Correctness gives the degree of service each function provides as per developed.
6. Maintainability – Each software product requires maintenance and up-gradation. Maintenance is an expensive and time-consuming process. So if the software product provides easy maintainability then we can say software quality is up to mark. Maintainability metrics include time requires to adapt to new features/functionality, Mean Time to Change (MTTC), performance in changing environments, etc.
7. Integrity – Software integrity is important in terms of how much it is easy to integrate with other required software's which increases software functionality and what is the control on integration from unauthorized software's which increases the chances of cyberattacks.
8. Security – Security metrics measure how much secure the software is? In the age of cyber terrorism, security is the most essential part of every software. Security assures that there are no unauthorized changes, no fear of cyber attacks, etc when the software product is in use by the end-user.

5.3 FURPS

FURPS is an acronym representing a model for classifying software quality attributes (functional and non-functional requirements):

- Functionality - Capability (Size & Generality of Feature Set), Reusability (Compatibility, Interoperability, Portability), Security (Safety & Exploitability)
- Usability (UX) - Human Factors, Aesthetics, Consistency, Documentation, Responsiveness
- Reliability - Availability (Failure Frequency (Robustness/Durability/Resilience), Failure Extent & Time-Length (Recoverability/Survivability)), Predictability (Stability), Accuracy (Frequency/Severity of Error)
- Performance - Speed, Efficiency, Resource Consumption (power, ram, cache, etc.), Throughput, Capacity, Scalability
- Supportability (Serviceability, Maintainability, Sustainability, Repair Speed) - Testability, Flexibility (Modifiability, Configurability, Adaptability, Extensibility, Modularity), Installability, Localizability

The model, developed at Hewlett-Packard was first publicly elaborated by Grady and Caswell. *FURPS+* is now widely used in the software industry. The + was later added to the model after various campaigns at HP to extend the acronym to emphasize various attributes.

5.4 Software quality assurance

Software quality assurance (often called *quality management*) is an umbrella activity that is applied throughout the software process. Software quality assurance (SQA) encompasses

(1) an SQA process,
(2) specific quality assurance and quality control tasks (including technical reviews and a multitiered testing strategy)
(3) effective software engineering practice (methods and tools)
(4) control of all software work products and the changes made to them

(5) a procedure to ensure compliance with software development standards
(when applicable)
(6) measurement and reporting mechanisms.

Elements of Software Quality Assurance

Software quality assurance encompasses a broad range of concerns and activities that focus on the management of software quality. These can be summarized in the following manner:
Standards. The IEEE, ISO, and other standards organizations have produced a broad array of software engineering standards and related documents. Standards may be adopted voluntarily by a software engineering organization or imposed by the customer or other stakeholders. The job of SQA is to ensure that standards that have been adopted are followed and that all work products conform to them.
Reviews and audits. Technical reviews are a quality control activity performed by software engineers for software engineers. Their intent is to uncover errors. Audits are a type of review performed by SQA personnel with the intent of ensuring that quality guidelines are being followed for software engineering work. For example, an audit of the review process might be conducted to ensure that reviews are being performed in a manner that will lead to the highest likelihood of uncovering errors.
Testing. Software testing is a quality control function that has one primary goal—to find errors. The job of SQA is to ensure
that testing is properly planned and efficiently conducted so that it has the highest likelihood of achieving its primary goal.
Error/defect collection and analysis. The only way to improve is to measure how you're doing. SQA collects and analyzes error and defect data to better understand how errors are introduced and what software engineering activities are best suited to eliminating them.
Change management. Change is one of the most disruptive aspects of any software project. If it is not properly managed, change can lead to confusion, and confusion almost always leads to poor quality. SQA ensures that
adequate change management practices have been instituted.

Education. Every software organization wants to improve its software engineering practices. A key contributor to improvement is education of software engineers, their managers, and other stakeholders. The SQA organization takes the lead in software process improvement and is a key proponent and sponsor of educational programs.

Vendor management. Three categories of software are acquired from external software vendors—*shrink-wrapped packages* (e.g., Microsoft *Office*), a *tailored shell* [Hor03] that provides a basic skeletal structure that is custom tailored to the needs of a purchaser, and *contracted software* that is custom designed and constructed from specifications provided by the customer
organization. The job of the SQA organization is to ensure that high-quality software results by suggesting specific quality practices that the vendor should follow (when possible), and incorporating quality mandates as part of any contract with an external vendor.
Security management. With the increase in cyber crime and new government regulations regarding privacy, every software organization should institute policies that protect data at all levels, establish firewall protection for WebApps, and ensure that software has not been tampered with internally. SQA ensures that appropriate process and technology are used to achieve software security.

Safety. Because software is almost always a pivotal component of humanrated systems (e.g., automotive or aircraft applications), the impact of hidden defects can be catastrophic. SQA may be responsible for assessing the impact of software failure and for initiating those steps required to reduce risk.
Risk management. Although the analysis and mitigation of risk is the concern of software engineers, the SQA organization ensures that risk management activities are properly conducted and that risk-related contingency plans have been established. In addition to each of these concerns and activities, SQA works to ensure that software support activities (e.g., maintenance, help lines, documentation, and manuals) are conducted or produced with quality as a dominant concern.

5.5 Software Reviews

Software Review is systematic inspection of a software by one or more individuals who work together to find and resolve errors and defects in the software during the early stages of Software Development Life Cycle (SDLC). Software review is an essential part of Software Development Life Cycle (SDLC) that helps software engineers in validating the quality, functionality and other vital features and components of the software. It is a whole process that includes testing the software product and it makes sure that it meets the requirements stated by the client.
Usually performed manually, software review is used to verify various documents like requirements, system designs, codes, test plans and test cases.

Objectives of Software Review:
The objective of software review is:

1. To improve the productivity of the development team.
2. To make the testing process time and cost effective.
3. To make the final software with fewer defects.
4. To eliminate the inadequacies.

Process of Software Review:

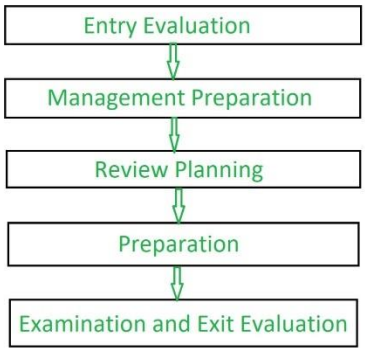


Fig 5.2 Process of software review

Types of Software Reviews:
There are mainly 3 types of software reviews:

1. **Software Peer Review:**
   Peer review is the process of assessing the technical content and quality of the product and it is usually conducted by the author of the work product along with some other developers.
   Peer review is performed in order to examine or resolve the defects in the software, whose quality is also checked by other members of the team.
   Peer Review has following types:

   - **(i) Code Review:**
     Computer source code is examined in a systematic way.

   - **(ii) Pair Programming:**
     It is a code review where two developers develop code together at the same platform.

   - **(iii) Walkthrough:**
     Members of the development team is guided by author and other interested parties and the participants ask questions and make comments about defects.

   - **(iv) Technical Review:**
     A team of highly qualified individuals examines the software product for its client's use and identifies technical defects from specifications and standards.

   - **(v) Inspection:**
     In inspection the reviewers follow a well-defined process to find defects.

2. **Software Management Review:**
   Software Management Review evaluates the work status. In this section decisions regarding downstream activities are taken.

3. **Software Audit Review:**
   Software Audit Review is a type of external review in which one or more critics, who are not a part of the development team, organize an independent inspection of the software product and its processes to assess their compliance with stated specifications and standards. This is done by managerial level people.

Advantages of Software Review:

- Defects can be identified earlier stage of development (especially in formal review).
- Earlier inspection also reduces the maintenance cost of software.
- It can be used to train technical authors.
- It can be used to remove process inadequacies that encourage defects.

5.6 Formal Technical Review (FTR)

Formal Technical Review (FTR) is a software quality control activity performed by software engineers.

Objectives of formal technical review (FTR):

Some of these are:

- Useful to uncover error in logic, function and implementation for any representation of the software.
- The purpose of FTR is to verify that the software meets specified requirements.
- To ensure that software is represented according to predefined standards.
- It helps to review the uniformity in software that is development in a uniform manner.
- To makes the project more manageable.

In addition, the purpose of FTR is to enable junior engineer to observer the analysis, design, coding and testing approach more closely. FTR also works to promote back up and continuity become familiar with parts of software they might not have seen otherwise.

Actually, FTR is a class of reviews that include walkthroughs, inspections, round robin reviews and other small group technical assessments of software. Each FTR is conducted as meeting and is considered successfully only if it is properly planned, controlled and attended.

The review meeting:

Each review meeting should be held considering the following constraints-

Involvement of people:

1. Between 3, 4 and 5 people should be involve in the review.
2. Advance preparation should occur but it should be very short that is at the most 2 hours of work for every person.
3. The short duration of the review meeting should be less than two hour. Gives these constraints, it should be clear that an FTR focuses on specific (and small) part of the overall software.

At the end of the review, all attendees of FTR must decide what to do.

1. Accept the product without any modification.
2. Reject the project due to serious error (Once corrected, another app need to be reviewed), or
3. Accept the product provisional (minor errors are encountered and are should be corrected, but no additional review will be required).

The decision was made, with all FTR attendees completing a sign-of indicating their participation in the review and their agreement with the findings of the review team.

Review reporting and record keeping :-

1. During the FTR, the reviewer actively records all issues that have been raised.
2. At the end of the meeting all these issues raised are consolidated and a review list is prepared.
3. Finally, a formal technical review summary report is prepared.

It answers three questions :-

1. What was reviewed ?
2. Who reviewed it ?
3. What were the findings and conclusions ?

Review guidelines :-
Guidelines for the conducting of formal technical reviews should be established in advance. These guidelines must be distributed to all reviewers, agreed upon, and then followed. A review that is unregistered can often be worse than a review that does not minimum set of guidelines for FTR.

1. Review the product, not the manufacture (producer).
2. Take written notes (record purpose)
3. Limit the number of participants and insists upon advance preparation.
4. Develop a checklist for each product that is likely to be reviewed.
5. Allocate resources and time schedule for FTRs in order to maintain time schedule.
6. Conduct meaningful training for all reviewers in order to make reviews effective.
7. Reviews earlier reviews which serve as the base for the current review being conducted.
8. Set an agenda and maintain it.
9. Separate the problem areas, but do not attempt to solve every problem notes.
10. Limit debate and rebuttal.

5.7 Formal Approaches to SQA

1. **Prepares an SQA plan for a project:** The program is developed during project planning and is reviewed by all stakeholders. The plan governs quality assurance activities performed by the software engineering team and the SQA group. The plan identifies calculation to be performed, audits and reviews to be performed, standards that apply to the project, techniques for error reporting and tracking, documents to be produced by the SQA team, and amount of feedback provided to the software project team.
2. **Participates in the development of the project's software process description:** The software team selects a process for the work to be performed. The SQA group reviews the process description for compliance with organizational policy, internal software standards, externally imposed standards (e.g., ISO-9001), and other parts of the software project plan.
3. **Reviews software engineering activities to verify compliance with the defined software process:** The SQA group identifies, reports, and tracks deviations from the process and verifies that corrections have been made.
4. **Audits designated software work products to verify compliance with those defined as a part of the software process:** The SQA group reviews selected work products, identifies, documents and tracks deviations, verify that corrections have been made, and periodically reports the results of its work to the project manager.
5. **Ensures that deviations in software work and work products are documented and handled according to a documented procedure:** Deviations may be encountered in

the project method, process description, applicable standards, or technical work products.

6. **Records any noncompliance and reports to senior management:** Non- compliance items are tracked until they are resolved.

5.8 Software Reliability

Software Reliability means **Operational reliability**. It is described as the ability of a system or component to perform its required functions under static conditions for a specific period.

Software reliability is also defined as the probability that a software system fulfills its assigned task in a given environment for a predefined number of input cases, assuming that the hardware and the input are free of error.

Software Reliability is an essential connect of software quality, composed with functionality, usability, performance, serviceability, capability, installability, maintainability, and documentation. Software Reliability is hard to achieve because the complexity of software turn to be high. While any system with a high degree of complexity, containing software, will be hard to reach a certain level of reliability, system developers tend to push complexity into the software layer, with the speedy growth of system size and ease of doing so by upgrading the software.

For example, large next-generation aircraft will have over 1 million source lines of software on-board; next-generation air traffic control systems will contain between one and two million lines; the upcoming International Space Station will have over two million lines on-board and over 10 million lines of ground support software; several significant life-critical defense systems will have over 5 million source lines of software. While the complexity of software is inversely associated with software reliability, it is directly related to other vital factors in software quality, especially functionality, capability, etc.

Software Reliability Models

A software reliability model indicates the form of a random process that defines the behavior of software failures to time.

Software reliability models have appeared as people try to understand the features of how and why software fails and attempt to quantify software reliability.

Over 200 models have been established since the early 1970s, but how to quantify software reliability remains mostly unsolved.

There is no individual model that can be used in all situations. No model is complete or even representative.

Most software models contain the following parts:

- Assumptions

- Factors

A mathematical function that includes the reliability with the elements. The mathematical function is generally higher-order exponential or logarithmic.

Software Reliability Modeling Techniques

Both kinds of modeling methods are based on observing and accumulating failure data and analyzing with statistical inference.

Differentiate between software reliability prediction models and software reliability estimation models

Basics	Prediction Models	Estimation Models
Data Reference	Uses historical information	Uses data from the current software development effort.
When used in development cycle	Usually made before development or test phases; can be used as early as concept phase.	Usually made later in the life cycle (after some data have been collected); not typically used in concept or development phases.
Time Frame	Predict reliability at some future time.	Estimate reliability at either present or some next time.

5.9 Software Quality Assurance Plan

Abbreviated as SQAP, the software quality assurance plan comprises of the procedures, techniques, and tools that are employed to make sure that a product or service aligns with the requirements defined in the SRS(software requirement specification).

Fig 5.3 Software quality assurance plan

SQA Activities
Given below is the list of SQA activities:

#1) Creating an SQA Management Plan:
The foremost activity includes laying down a proper plan regarding how the SQA will be carried out in your project.

Along with what SQA approach you are going to follow, what engineering activities will be carried out, and it also includes ensuring that you have a right talent mix in your team.

#2) Setting the Checkpoints:
The SQA team sets up different checkpoints according to which it evaluates the quality of the project activities at each checkpoint/project stage. This ensures regular quality inspection and working as per the schedule.

#3) Apply software Engineering Techniques:

Applying some software engineering techniques aids a software designer in achieving high-quality specification. For gathering information, a designer may use techniques such as interviews and FAST (Functional Analysis System Technique).

Later, based on the information gathered, the software designer can prepare the project estimation using techniques like WBS (work breakdown structure), SLOC (source line of codes), and FP(functional point) estimation.

#4) Executing Formal Technical Reviews:
An FTR is done to evaluate the quality and design of the prototype.

In this process, a meeting is conducted with the technical staff to discuss regarding the actual quality requirements of the software and the design quality of the prototype. This activity helps in detecting errors in the early phase of SDLC and reduces rework effort in the later phases.

#5) Having a Multi- Testing Strategy:
By multi-testing strategy, we mean that one should not rely on any single testing approach, instead, multiple types of testing should be performed so that the software product can be tested well from all angles to ensure better quality.

Applying some software engineering techniques aids a software designer in achieving high-quality specification. For gathering information, a designer may use techniques such as interviews and FAST (Functional Analysis System Technique).

Later, based on the information gathered, the software designer can prepare the project estimation using techniques like WBS (work breakdown structure), SLOC (source line of codes), and FP(functional point) estimation.

#4) Executing Formal Technical Reviews:
An FTR is done to evaluate the quality and design of the prototype.

In this process, a meeting is conducted with the technical staff to discuss regarding the actual quality requirements of the software and the design quality of the prototype. This activity

helps in detecting errors in the early phase of SDLC and reduces rework effort in the later phases.

#5) Having a Multi- Testing Strategy:
By multi-testing strategy, we mean that one should not rely on any single testing approach, instead, multiple types of testing should be performed so that the software product can be tested well from all angles to ensure better quality.

By validating the change requests, evaluating the nature of change and controlling the change effect, it is ensured that the software quality is maintained during the development and maintenance phases.

#8) Measure Change Impact:
If any defect is reported by the QA team, then the concerned team fixes the defect.

After this, the QA team should determine the impact of the change which is brought by this defect fix. They need to test not only if the change has fixed the defect, but also if the change is compatible with the whole project.

For this purpose, we use software quality metrics which allows managers and developers to observe the activities and proposed changes from the beginning till the end of SDLC and initiate corrective action wherever required.

#9) Performing SQA Audits:
The SQA audit inspects the entire actual SDLC process followed by comparing it against the established process.

It also checks whatever reported by the team in the status reports were actually performed or not. This activity also exposes any non-compliance issues.

#10) Maintaining Records and Reports:
It is crucial to keep the necessary documentation related to SQA and share the required SQA information with the stakeholders. The test results, audit results, review reports, change requests documentation, etc. should be kept for future reference.

#11) Manage Good Relations:
In fact, it is very important to maintain harmony between the QA and the development team.

We often hear that testers and developers often feel superior to each other. This should be avoided as it can affect the overall project quality.

5.10 Clean room Methodology

The **cleanroom software engineering** process is a software development process intended to produce software with a certifiable level of reliability. The cleanroom process was originally developed by Harlan Mills and several of his colleagues including Alan Hevner at IBM. The focus of the cleanroom process is on defect prevention, rather than defect removal. The name "cleanroom" was chosen to evoke the cleanrooms used in the electronics industry to prevent the introduction of defects during the fabrication of semiconductors. The

cleanroom process first saw use in the mid to late 1980s. Demonstration projects within the military began in the early 1990s. Recent work on the cleanroom process has examined fusing cleanroom with the automated verification capabilities provided by specifications expressed in CSP.

The basic principles of the cleanroom process are

Software development based on formal methods

Software tool support based on some mathematical formalism includes model checking, process algebras, and Petri nets. The Box Structure Method might be one such means of specifying and designing a software product.[4] Verification that the design correctly implements the specification is performed through team review, often with software tool support.

Incremental implementation under statistical quality control

Cleanroom development uses an iterative approach, in which the product is developed in increments that gradually increase the implemented functionality. The quality of each increment is measured against pre-established standards to verify that the development process is proceeding acceptably. A failure to meet quality standards results in the cessation of testing for the current increment, and a return to the design phase.

Statistically sound testing

Software testing in the cleanroom process is carried out as a statistical experiment. Based on the formal specification, a representative subset of software input/output trajectories is selected and tested. This sample is then statistically analyzed to produce an estimate of the reliability of the software, and a level of confidence in that estimate.

Part-A unit 5			
Q.No	**Questions**	**Competence**	**BT Level**
1.	Explain the need for quality measurement in software project	Knowledge	BTL 1
2.	Illustrate the condition for portability in software	Remember	BTL 1
3.	Define the constraints to be met during the review meeting	Remember	BTL 1
4.	Write a note on process of Software Review	Analysis	BTL 4
5.	Define the objective of software review	Remember	BTL 1
6.	Define Audit Review	Remember	BTL 1
7.	Write three categories of software are acquired from external software vendor	Analysis	BTL 4
8.	Describe the end decision of attandees in review meeting	Understand	BTL2
9.	What is risk management	Knowledge	BTL 1

10	Appraise the reason for quality control in software management	Understand	BTL2

Part-B

Q.No	**Questions**	**Competence**	**BT Level**
1.	Ilustrate the of elements of Software Quality Assurance	Analyze	BTL 4
2.	Identify the software control activity for formating the project developed by software engineers to uncover logical errors	Analyze	BTL 4
3.	Explain the types of software reviews	Knowledge	BT5
4.	Define the quality mesures for evaluate a project	Knowledge	BT 1
5.	Illustrate quality assurance plain in detail	Analyze	BT 4
6.	Write a short note on FURPS	Create	BT 6
7.	Criticize the growth of Quality Management System	Evaluate	BT 5