Introduction to Project Management: Introduction, Project and Importance of Project Management, Contract Management, Activities Covered by Software Project Management, Plans, Methods and Methodologies, Some ways of categorizing Software Projects, Stakeholders, Setting Objectives, Business Case, Project Success and Failure, Management and Management Control, Project Management life cycle, Traditional versus Modern Project Management Practices.

Introduction to Software Project Management and Importance

1. Definition:

- Software Project Management is both an art and a science, involving the planning and leading of software projects from initial ideas to final reality.

2. Software Project:

- A software project encompasses the complete process of software development, from requirement gathering through to testing and maintenance. It follows specific execution methodologies and is bound by a defined time frame to achieve the intended software product.

3. Project Management:

- Project management is the discipline of defining and achieving targets while optimizing resources (time, money, people, materials, energy, space, etc.) over the course of a project. This involves planning, monitoring, and controlling people, processes, and events during software development.

4. Importance of Management:

- Effective management is crucial due to the complexity and long duration of software projects. Successful project management requires focusing on four key areas: people, product, process, project

5. Project Plan:

- A project plan is a document that defines the four key areas to ensure a cost-effective and high-quality software product. The success of the project plan can be measured by the timely delivery of a high-quality product within budget.

Why is Software Project Management Important?

1. Significant Investment in ICT:

- Large sums are invested in ICT (Information and Communication Technology). For example, the UK government in 2003-04 spent €2.3 billion on ICT contracts, compared to €1.4 billion on road building.

2. High Failure Rates:

- Many projects fail; according to the Standish Group, only a third of ICT projects are successful, with 82% being late and 43% exceeding their budget. Poor project management is a significant factor in these failures.

3. Methodology Criticism:

- Although the Standish Group's methodology has been criticized, there is a general perception that ICT project failures are prevalent.

Software Development Life Cycle (SDLC)

1. Definition:

- The SDLC is a methodology that provides a framework for planning, controlling, and delivering an information system. It includes stages such as requirement gathering, design, development, testing, and maintenance.

2. Framework Application:

- The SDLC acts as a foundation for various development and delivery methodologies. It can be applied to hardware and software development, with each life cycle focusing on its specific components.

Four Project Dimensions:

- People
- Process
- Product
- Technology

The Five Variables of Project Control

1. **Time:**
    - The amount of time required to complete the project.
2. **Cost:**
    - Calculated based on the time required for project completion.
3. **Quality:**
    - Determined by the time and effort invested in individual tasks.
4. **Scope:**
    - The specified requirements for the end result.
5. **Risk:**
    - Potential points of failure within the project.

Trade-Off Triangle

The triangle illustrates the relationship between three primary forces in a project:

- **Time:** The available time to deliver the project.
- **Cost:** The amount of money or resources available.
- **Quality:** The fit-to-purpose that the project must achieve to be successful.

Complexity Management in Software Projects

1. **Complexity Management:**
    - Software projects often involve intricate systems and interdependencies. Effective management ensures the project remains coherent and manageable.
2. **Requirement Management:**
    - Clear and precise requirements management is essential to meet user needs and expectations. Mismanagement can lead to scope creep and project failure.
3. **Time and Budget Control:**
    - Monitoring and controlling the project timeline and budget prevents overruns.
4. **Risk Management:**
    - Identifying, assessing, and mitigating risks prevent unforeseen issues from derailing a project.
5. **Quality Assurance:**

- Ensuring the project meets quality standards is crucial for user satisfaction and reducing post-release defects. Continuous testing and validation are key practices.

6. **Team Coordination:**
   - Effective communication and coordination among team members ensure alignment with project goals.
7. **Stakeholder Management:**
   - Engaging and managing stakeholders helps in gaining support and addressing concerns, critical for project acceptance and success.
8. **Scope Management:**
   - Defining and controlling the project scope prevents scope creep and ensures necessary features are delivered.
9. **Process Improvement:**
   - Continuously improving processes ensures the project uses the most efficient methods and practices.
10. **Resource Allocation:**
    - Efficient allocation and management of resources ensure the project has what it needs to succeed without wastage.

Statistics Highlighting the Importance of Efficient Project Management

1. **32% of Projects Fail Due to Poor Management:**
   - This statistic emphasizes the critical impact of project management on overall project success.
2. **68% of Projects Fail to Meet Deadlines, Budgets, and Quality Targets:**
   - A significant majority of projects struggle with these aspects, highlighting the need for effective project management practices.
3. **97% of Businesses Believe Project Management is Essential for Success:**
   - This shows the recognized value of project management among businesses, underscoring its importance for achieving business objectives.
4. **80% of High-Performing Projects are Led by a Qualified Project Manager:**
   - This indicates a correlation between project manager qualifications and project performance.
5. **On Average, a Large IT Project Runs 45% Over Budget:**

- This points to common budget overruns in large IT projects, emphasizing the need for rigorous budget control and efficient resource management.

What is a Project?

Definition:

- A project involves planning and carrying out tasks to achieve specific objectives within a finite duration. Even exploratory or uncertain projects benefit from careful planning, accepting that some plans may have provisional elements.

Dictionary Definitions of 'Project':

- A specific plan or design.
- A planned undertaking.
- A large undertaking, e.g., a public works scheme.

Key Points:

- Planning and the size of the task are crucial.

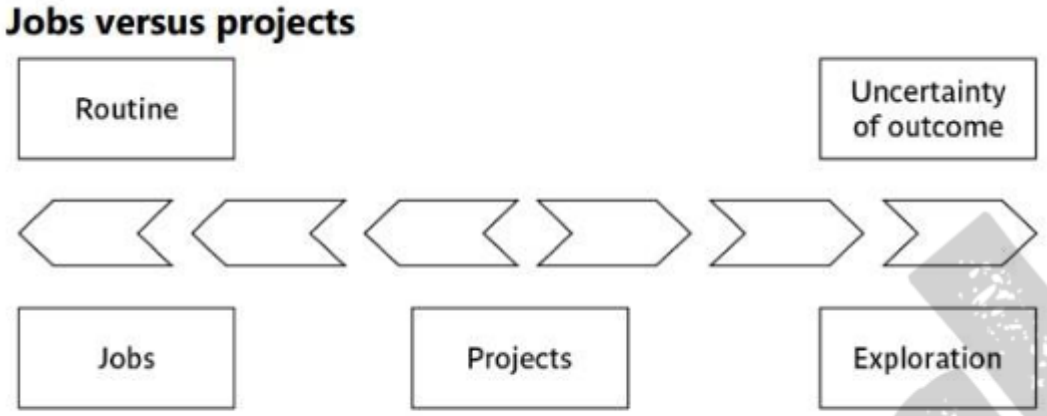


Fig:1.1 Activities most likely to benefit from project management.

Project Characteristics:

- Non-routine tasks.
- Planning is required.

- Specific objectives or products.
- Pre-determined time span.
- Work is for someone other than yourself.
- Involves several specializations.
- Carried out in several phases.
- Resource constraints.
- Large or complex.

Software Projects versus Other Types of Projects

Invisibility:

- Unlike physical artifacts, the progress of software development is not immediately visible.

Complexity:

- Software products are more complex than other engineered artifacts.

Conformity:

- Traditional engineering deals with physical systems governed by consistent physical laws, whereas software developers must conform to the varying requirements of human clients.

Flexibility:

- Software's ease of change means it often adapts to physical or organizational systems, leading to a high degree of change.

Example:

- An infrastructure project example is the construction of a flyover.
- A software project example is the development of a payroll management system using Oracle 10g and Oracle Forms 10G.

Contract Management

Overview: Contract management is a critical process that involves overseeing the creation, execution, and analysis of contracts to ensure optimal operational and financial performance while minimizing risks. This process becomes particularly crucial when organizations decide to outsource ICT development to external vendors.

Stages of Contract Management:

1. **Request and Creation:**
   - **Request:** This stage begins with identifying the need for a contract. It involves gathering all necessary information and requirements that the contract must address.
   - **Creation:** Once the requirements are gathered, the contract terms and conditions are drafted. These terms must align closely with the objectives and expectations of all parties involved.

   Example: Suppose a software company needs to engage a third-party developer to create a new module for their existing software suite. The project manager initiates the process by collecting detailed specifications, timelines, payment terms, and other relevant details.

2. **Negotiation:**
   - During this stage, the parties involved discuss and negotiate the terms laid out in the contract. Negotiation often involves revisions and adjustments to reach a mutually agreeable arrangement.

   Example: The software company negotiates with the third-party developer regarding project milestones, deliverables, pricing, and any special requirements. Both parties aim to find common ground that ensures the project's success and aligns with their respective interests.

3. **Approval and Execution:**
   - **Approval:** Once the terms are finalized through negotiation, the contract undergoes review and approval processes within the respective legal and stakeholder frameworks.
   - **Execution:** Signing the contract by authorized representatives of both parties makes it legally binding.

   Example: After legal teams review and approve the contract terms, the software company and the third-party developer sign the contract, formalizing their commitments and responsibilities.

4. **Obligations and Performance:**
   - This stage involves ensuring that all parties fulfill their obligations as per the agreed terms and conditions outlined in the contract. It includes monitoring performance and compliance throughout the contract lifecycle.

   Example: The third-party developer begins work according to the project schedule and milestones agreed upon in the contract. Simultaneously, the software company provides necessary support, resources, and periodic payments as per the contract terms.

5. **Modification and Renewal:**
    - Contracts are dynamic documents. If circumstances change during the contract period, amendments may be necessary to accommodate new requirements or adjust existing terms. Additionally, contracts may be reviewed for renewal if the project extends beyond the initial agreement.

    Example: Midway through the development phase, the software company decides to add new features to the software module, not originally included in the contract. Both parties negotiate and amend the contract to include these changes, addressing any corresponding adjustments in timelines and compensation.

6. **Closure:**
    - The final stage involves completing all contractual obligations, ensuring that all deliverables are satisfactorily met, and formally closing the contract. It includes conducting final reviews, resolving any outstanding issues, and making the last payments.

    Example: After successful completion of the development project, including thorough testing and acceptance of the software module, the software company conducts a final review. Once all parties agree that the deliverables meet the agreed-upon standards, the contract is closed, and any remaining payments are processed.

Benefits of Effective Contract Management:

- **Risk Mitigation:** Identifies and addresses potential risks early in the contract lifecycle.
- **Improved Compliance:** Ensures all parties adhere to legal and regulatory requirements.
- **Cost Savings:** Avoids unnecessary expenses and penalties through efficient contract management.
- **Performance Tracking:** Monitors and evaluates performance against agreed-upon metrics and milestones.
- **Relationship Management:** Fosters positive relationships between contracting parties through clear communication and mutual understanding.
- **Speed to Market:** Utilizes vendor expertise and resources effectively to accelerate project timelines and delivery.

Example: XYZ Tech (Detailed Scenario)

1. **Identifying Needs:** XYZ Tech identifies the need to develop a mobile app to complement its existing software suite, aiming to enhance user experience and expand market reach.

2. **Selecting a Vendor:** After evaluating several firms known for their expertise in mobile app development, XYZ Tech shortlists a reputable vendor based in India.
3. **Defining Requirements:** XYZ Tech collaborates closely with its internal stakeholders to define comprehensive requirements for the mobile app. This includes specifying features, user interface design preferences, performance benchmarks, and compatibility with existing systems.
4. **Contract Negotiation:** Negotiations between XYZ Tech and the selected vendor focus on outlining clear deliverables, establishing project timelines, defining payment structures, and incorporating confidentiality clauses to protect intellectual property.
5. **Project Management:** XYZ Tech appoints a dedicated project manager to oversee the development process. The manager serves as the primary point of contact for the vendor, ensuring regular updates, milestone reviews, and alignment with project objectives.
6. **Delivery and Integration:** Upon completion of development, the vendor delivers the mobile app, thoroughly tested to ensure functionality and performance. XYZ Tech's internal teams coordinate seamless integration of the app with their existing software suite.
7. **Post-Delivery Support:** The vendor provides ongoing maintenance and support services post-launch, addressing any issues or enhancements requested by XYZ Tech. This ensures the app's continued reliability and responsiveness to user needs.

Activities Covered by Software Project Management

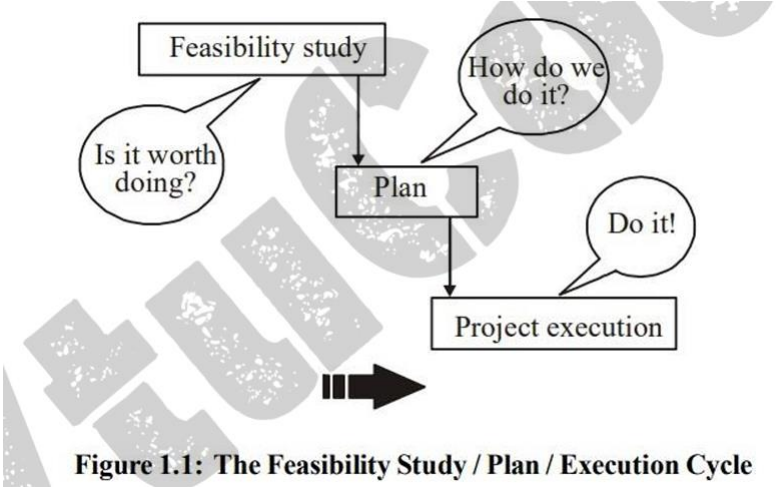


Figure 1.1: The Feasibility Study / Plan / Execution Cycle

Feasibility Study

Purpose: Determine if a prospective project is worth starting.

Activities:

- **Requirement Gathering**:
  - Identify and document what the proposed application needs to do.

  - Engage stakeholders to understand their needs and expectations.
  - Perform initial analysis to outline the scope of the project.
- **Cost and Benefit Estimation**:
  - Estimate the developmental costs, including labor, technology, and infrastructure.
  - Estimate operational costs, such as maintenance, support, and training.
  - Analyze the potential benefits, including increased efficiency, cost savings, and revenue generation.
- **Strategic Planning**:
  - Evaluate how the project aligns with the organization's strategic goals.
  - Prioritize the project among other potential software developments.
  - Assess the risks and uncertainties associated with the project.

Planning

Purpose: Develop a plan if the feasibility study shows the project is viable.

Activities:

- **Outline Planning**:
  - Create a high-level plan covering the entire project lifecycle.
  - Define project objectives, deliverables, and milestones.
  - Establish the project timeline and key deadlines.
- **Detailed Planning for Initial Stage**:
  - Develop a comprehensive plan for the first stage of the project.
  - Identify specific tasks, resources, and responsibilities.
  - Establish a detailed schedule and allocate resources.
- **Incremental Detailed Planning**:
  - As the project progresses, develop detailed plans for subsequent stages.
  - Update plans based on new information and project developments.
  - Ensure continuous alignment with project objectives and stakeholder expectations.

Project Execution

Purpose: Execute the project, which includes design and implementation.

Activities:

- **Requirements Analysis**:

- Conduct in-depth requirement elicitation with stakeholders.
- Use techniques such as interviews, surveys, and workshops to gather requirements.
- Document and validate requirements, ensuring they are clear, complete, and feasible.

- **Specification**:
  - Create detailed documentation of the system's functional and non-functional requirements.
  - Develop use cases, user stories, and process models to describe the system's behavior.
  - Ensure all requirements are traceable and testable.
- **Design**:
  - **Architecture Design**:
    - Define the high-level structure of the system.
    - Determine how the system's components will interact.
    - Decide on the division of tasks between user actions and automated processes.
    - Develop architectural diagrams and design documents.
  - **Detailed Design**:
    - Break down the system into smaller, manageable units.
    - Design the internal structure and algorithms for each unit.
    - Create detailed design specifications for coding and testing.
- **Coding**:
  - Write code based on the detailed design specifications.
  - Use appropriate programming languages and development tools.
  - Follow coding standards and best practices.
  - Perform code reviews and peer reviews to ensure quality.
- **Testing (Verification and Validation)**:
  - **Integration**:
    - Combine individual software components and test them as a unified system.
    - Ensure components interact correctly and meet overall requirements.
    - Perform integration testing to identify and fix issues.
  - **Qualification Testing**:
    - Conduct comprehensive testing of the entire system.
    - Validate that the system meets all specified requirements.
    - Perform user acceptance testing (UAT) with stakeholders to confirm the system's readiness.
- **Implementation/Installation**:
  - Deploy the system to the production environment.
  - Ensure all components are correctly installed and configured.
  - Conduct final testing to verify successful implementation.

    - Provide training and support to users during the transition.
- **Acceptance Support**:
    - Offer ongoing maintenance and support to address issues and make improvements.
    - Monitor the system's performance and make necessary adjustments.
    - Plan and execute updates and enhancements as needed.
    - Document all maintenance activities and changes.

Plans Methods and Methodologies

1. **Method**: A method refers to a general approach or technique for carrying out a type of activity. It outlines the steps needed to achieve a certain goal. For example, in software testing, a method might include steps such as analyzing requirements, writing test cases, and executing tests.
2. **Plan**: A plan converts the method into actionable steps. It details specific activities, timelines, responsible individuals, and required resources. It makes the method practical and executable.
3. **Methodology**: A methodology is a structured combination of methods and techniques used to achieve a broader objective. It often includes multiple methods that are applied in sequence or in parallel to complete a complex process.

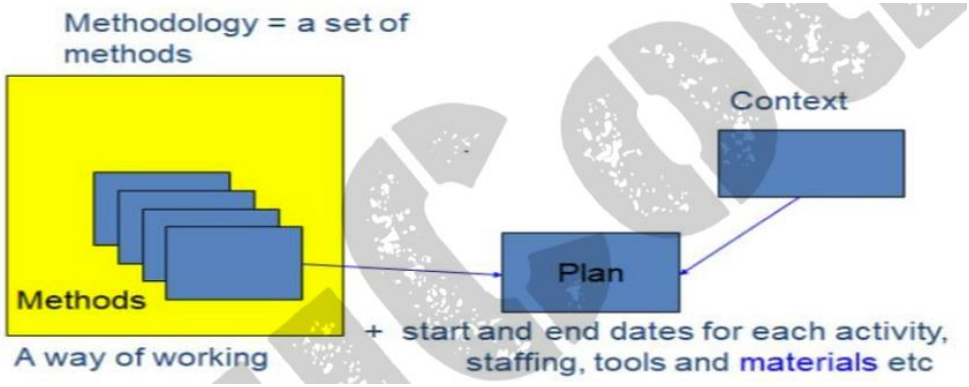
A plan for an activity must be based on some idea of a method of work. To take a simple example, if you were asked to test some software, even though you do not know anything about the software to be tested, you could assume that you would need to:

- Analyze the requirements for the software
- Devise and write test cases that will check that each requirement has been satisfied
- Create test scripts and expected results for each test case
- Compare the actual results and the expected results and identify discrepancies

While a method relates to a type of activity in general, a plan takes that method (and perhaps others) and converts it to real activities, identifying for each activity:

- Its start and end dates
- Who will carry it out?
- What tools and materials will be used?

'Materials' in this context could include information, for example a requirements document. With complex procedures, several methods may be deployed, in sequence or in parallel. The output from one method might be the input to another. Groups of methods or techniques are often referred to as methodologies.



Some ways of categorizing Software Projects

1. Changes to the Characteristics of Software Projects

- **Historical vs. Modern Development**:
  - **Historical**: Software was written from scratch with no code reusability.
  - **Modern**: Most programming languages support code reusability, dynamic linking, and frameworks.
- **Project Duration**:
  - **Historical**: Multi-year projects.
  - **Modern**: Projects often completed in a few months.
- **Customer Participation**:
  - **Historical**: Limited to initial requirements gathering and final delivery.
  - **Modern**: Customers are involved in almost every aspect of the project.

2. Compulsory vs. Voluntary Users

- **Compulsory Systems**:
  - Example: Order processing systems in organizations.
  - Users are required to use these systems to perform their job functions.
- **Voluntary Systems**:
  - Example: Computer games.
  - Users choose to use these systems.
  - Requirements are less precise and depend more on developer creativity and user feedback (e.g., market surveys, focus groups, prototype evaluation).

3. Information Systems vs. Embedded Systems

- **Information Systems**:
  - Enable staff to carry out office processes.
  - Example: Stock control systems.
- **Embedded Systems**:
  - Control machines or equipment.
  - Example: Air conditioning control systems.
- **Hybrid Systems**:
  - Combine elements of both.
  - Example: A stock control system that also controls an automated warehouse.

4. Software Products vs. Services

- **Software Products**:
  - Developed for general customers.
  - Sold off-the-shelf to a large number of customers.
  - Examples: Microsoft Windows, Oracle 8i.
  - Domain-specific products target specific customer segments (e.g., BANCS from TCS, FINACLE from Infosys).
- **Software Services**:
  - Include customization, outsourcing, maintenance, testing, and consultancy.
  - Aim to meet specific objectives rather than producing a product.

5. Outsourced Projects

- **Outsourcing**:
  - Companies outsource parts of their work to other companies.
  - Reasons for outsourcing:
    - Lack of expertise in specific areas.
    - Cost-effectiveness.

6. Object-driven Development

- **Object-driven Projects**:
  - Aim to meet certain objectives or identify the need for a new software system.
  - Example: A project that results in recommendations for a new software system.
- **Product Development Projects**:

- Aim to create a software product based on identified needs or recommendations.

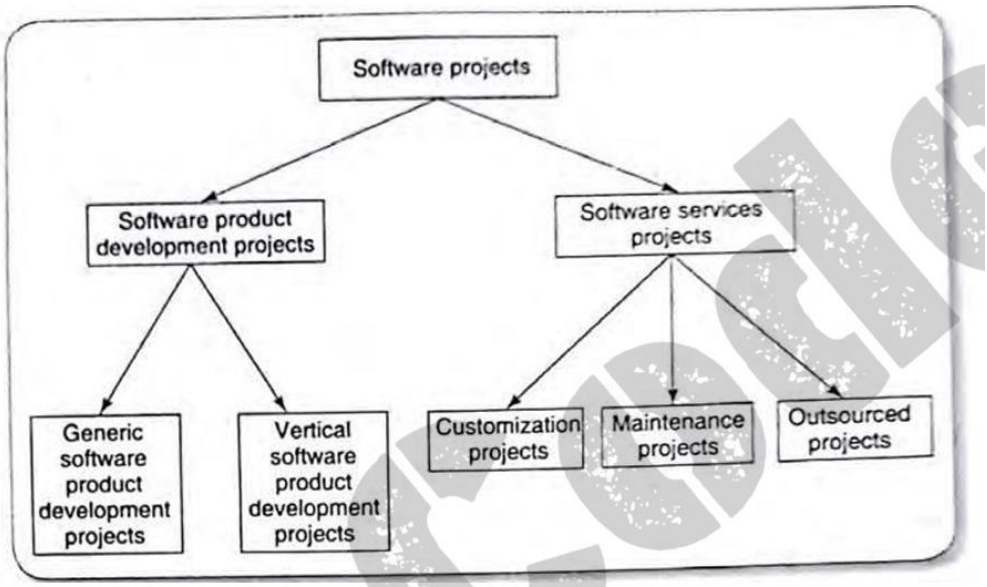


Fig: 1.7 A Classification of software projects

STAKEHOLDERS

Stakeholders in a Project

1. **Importance of Identifying Stakeholders Early:**
   - **Reason:** Establish adequate communication channels from the start.
   - **Diverse Interests:** Stakeholders have different motivations and objectives.
     - **Example:** End-users focus on ease of use, while managers look at cost savings.
2. **Theory W (Win-Win) by Boehm and Ross:**
   - **Concept:** Software project management theory where the manager aims to create scenarios where all parties benefit.
   - **Objective:** Foster a shared interest in the project's success.
3. **Types of Stakeholders:**
   - **Internal to the Project Team:**
     - **Characteristics:** Direct managerial control by the project leader.
     - **Example:** Team members working directly on the project.
   - **External to the Project Team but within the Same Organization:**
     - **Characteristics:** Require assistance from other departments within the same organization.
     - **Example:** Needing help from the information management group for database updates or from users for system testing.

        - **Commitment:** Needs to be negotiated.
    - **External to Both the Project Team and the Organization:**
        - **Characteristics:** Include customers or contractors.
        - **Example:** Users who will benefit from the system or contractors performing project-related work.
        - **Relationships:** Often based on legally binding contracts.
4. **Managing Different Stakeholder Objectives:**
    - **Project Leader's Role:**
        - Recognize varying interests among stakeholders.
        - Reconcile these interests effectively.
        - **Skills Needed:** Good communication and negotiation skills.

Setting Objectives

Defining Objectives:

- Objectives define what the project team must achieve for project success.
- They focus on desired outcomes (post-conditions) rather than tasks within the project.
- Objectives can be framed as statements starting with “the project will be a success if...”.

Importance of Clear Objectives:

- The project manager and team must know what constitutes success to focus on essential aspects.
- Multiple user groups and specialist teams involved in a project require well-defined, universally accepted objectives.
- A project authority, often a steering committee, oversees and authorizes any changes to objectives and resources.

Project Steering Committee:

- Holds overall responsibility for setting, monitoring, and modifying objectives.
- Project manager runs the project day-to-day but reports to the steering committee.
- Only the steering committee can authorize changes to project objectives and resources.

Sub-objectives and Goals

Guidance and Motivation:

- Objectives guide and motivate individuals and groups.
- Effective objectives for individuals must be within their control.
- For example, a business objective might be reducing staff costs, but a software developer's objective should be keeping development costs within budget.

Breaking Down Objectives:

- Objectives are broken down into goals or sub-objectives.
- Goals are steps toward achieving the main objective, similar to scoring goals in a football match to win.

SMART Objectives

SMART Mnemonic:

- **Specific:** Objectives should be concrete and well-defined. Vague aspirations are unsatisfactory.
- **Measurable:** There should be measures of effectiveness to gauge success. For example, "reduce customer complaints" is more measurable than "improve customer relations."
- **Achievable:** Objectives must be within the power of the individual or group to achieve.
- **Relevant:** Objectives must be relevant to the true purpose of the project.
- **Time-constrained:** Objectives should have a defined timeline for achievement.

Measures of Effectiveness

Practical Methods:

- Measures of effectiveness ascertain whether an objective has been met.
- Example: "Mean time between failures" (MTBF) is used to measure reliability.
- Measures are usually related to the installed operational system.

Business Case

In software engineering, the justification or business case for a project is crucial to ensure that the effort, time, and expense invested will yield worthwhile benefits. A well-defined business case helps in aligning the project with organizational goals and securing stakeholder support. Here are key considerations:

Quantification of Benefits

To substantiate the project's value, it is often necessary to develop a business model that outlines how the new application will generate the anticipated benefits. This model should include:

- **Revenue Generation:** How the application will increase revenue through new customers, higher sales, or improved market share.
- **Cost Savings:** How the application will reduce costs by streamlining operations, reducing errors, or improving efficiency.
- **Improved Service Quality:** How the application will enhance customer satisfaction, leading to increased loyalty and repeat business.
- **Competitive Advantage:** How the application will provide a strategic advantage over competitors through unique features or better performance.

Maintaining the Business Case

Throughout the project lifecycle, it is essential to ensure that the business case remains viable. This involves managing development costs, project scope, and delivery timelines:

1. **Development Costs**
    - **Budget Management:** Ensuring that development costs do not escalate to a point where they outweigh the projected benefits.
    - **Resource Allocation:** Efficient allocation of resources to avoid unnecessary expenditures and optimize productivity.
2. **System Features**
    - **Scope Management:** Avoiding excessive scope reduction that might compromise the system's ability to deliver the expected benefits.
    - **Feature Prioritization:** Ensuring that critical features are prioritized to meet key business objectives.
3. **Delivery Date**
    - **Schedule Management:** Adhering to project timelines to avoid delays that could result in lost opportunities or benefits.
    - **Risk Management:** Identifying and mitigating risks that could impact the project schedule, such as resource shortages or technical challenges.

Traditional vs. Modern Management Practices in Software Development

Over the last two decades, the software industry has seen significant changes in its approach to software development. The shift from traditional to modern management practices reflects the evolving nature of software projects and the increasing complexity and dynamism of the industry. Here are some key differences between traditional and modern management practices:

1. Planning and Incremental Delivery

- **Traditional:** Projects were simpler and more predictable. Detailed project plans were created before execution, with a focus on adhering to the plan throughout the project lifecycle.
- **Modern:** Projects require rapid application development and deployment. Planning focuses on incremental deliveries with evolving functionalities, accommodating client feedback and participation. This approach, often called extreme project management, is highly flexible and human-centric.

2. Quality Management

- **Traditional:** Quality was primarily assessed at the final stages of the project, with a focus on meeting predefined specifications.
- **Modern:** Quality management is continuous and proactive. Project managers assess progress and track the quality of all intermediate artifacts, reflecting increased customer awareness and expectations regarding product quality.

3. Change Management

- **Traditional:** Once requirements were signed off by the customer, changes were rarely entertained. The focus was on adhering to the original plan.
- **Modern:** Customer suggestions and feedback are actively solicited and incorporated throughout the development process. Incremental delivery models and version control are used to manage changes, making change management (or configuration management) a crucial responsibility of the project manager.

4. Requirement Management

- **Traditional:** Requirements were identified upfront, signed off by the customer, and frozen before development started.
- **Modern:** Requirements frequently change during the development cycle. Requirement management has become a systematic process involving controlling changes, documenting, analyzing, tracing, and prioritizing requirements, and communicating changes to stakeholders.

5. Release Management

- **Traditional:** Software was typically released as a single, complete product at the end of the development cycle.
- **Modern:** Frequent and regular releases are made throughout the development process. Agile methodologies advocate for the release of basic functionalities first, followed by incremental releases of more complete functionalities. Effective release management is crucial for maintaining customer engagement and satisfaction.

6. Risk Management

- **Traditional:** Risk management was often reactive, dealing with issues as they arose.
- **Modern:** Effective risk management is proactive and considered essential to project success. It involves identifying risks, assessing their impacts, prioritizing them, and preparing risk-containment plans.

7. Scope Management

- **Traditional:** The project scope was fixed early in the project lifecycle, with minimal changes allowed.
- **Modern:** Customers are encouraged to submit change requests. Scope management involves balancing scope, schedule, and project cost, as these parameters are interdependent and affect each other.

Project Management Life Cycle

The project management life cycle is a comprehensive framework that guides project managers through the stages of a project, from initiation to closure. This cycle ensures that projects are completed successfully, meeting their objectives and delivering the intended benefits.

Phases of the Project Management Life Cycle

1. **Project Initiation**
    - **Concept Development:** Understand the characteristics, scope, constraints, costs, and benefits of the software to be developed.
    - **Feasibility Study:** Determine if the project is financially and technically feasible.
    - **Business Case:** Develop a business case to justify the project.
    - **Project Charter:** Create a document outlining the project's objectives, stakeholders, and authority of the project manager.

- **Team Formation:** Assemble the project team.

2. **Project Planning**
   - **Project Plan:** Identify tasks and schedule resources and timeframes.
   - **Resource Plan:** List required resources, including manpower and equipment.
   - **Functional Plan:** Document plans for manpower, equipment, and other costs.
   - **Quality Plan:** Outline quality targets and control plans.
   - **Risk Plan:** Identify potential risks, prioritize them, and plan actions to mitigate them.
   - **Detailed Planning:** Estimate cost, duration, and effort. Develop schedules and staffing plans, and address risk management.
3. **Project Execution**
   - **Task Execution:** Execute tasks as per the project plan.
   - **Process Quality:** Ensure quality through proper processes.
   - **Deliverables Production:** Produce and deliver the project's outputs.
4. **Project Monitoring and Controlling**
   - **Progress Monitoring:** Track the project's progress against the plan.
   - **Control Activities:** Address any deviations from the plan to keep the project on track.
   - **Plan Adjustment:** Revise the project plan as necessary to accommodate new information and constraints.
5. **Project Closure**
   - **Deliverables Release:** Release all required deliverables to the customer.
   - **Documentation:** Complete necessary documentation.
   - **Resource Release:** Release project resources and terminate vendor agreements.
   - **Payments Completion:** Complete all pending payments.
   - **Post-Implementation Review:** Analyze project performance and document lessons learned for future projects.

Key Principles and Practices

W5HH Principle

Barry Boehm's W5HH principle outlines key questions to understand project characteristics:

- **Why** is the software being built?
- **What** will be done?
- **When** will it be done?
- **Who** is responsible?
- **Where** are they organizationally located?

- **How** will the job be done technically and managerially?
- **How much** of each resource is needed?

Project Bidding

For some projects, a formal bidding process is necessary to select suitable vendors:

- **Request for Quotation (RFQ):** Used when the project and solutions are well understood.
- **Request for Proposal (RFP):** Used when the project is understood but solutions are not clear.
- **Request for Information (RFI):** Used to gather information about vendor capabilities.