Summary

Requirements Validation & Verification

Requirements Validation

It checks that the right product is being built. Ensures that the software being developed (or changed) will satisfy its stakeholders. Compares the software requirements specification against stakeholders goals and requirements.

Requirements Verification

It checks that the product is being built right. Ensures that each step followed in the process of building the software yields the right products. Checks consistency of the software requirements specification, design and implementation against the specification.

Requirement Validation and Verification Techniques

Simple Checks:

Various checks can be done using traceability techniques. Given the requirements document, verify all elicitation notes are covered. Tracing between different levels of requirements.

Involves developing a traceability matrix. Ensures that requirements have been taken into consideration (if not there should be a reason). Ensures that everything in the specification is justified.

Verify that the requirements are well written.

Prototyping:

Excellent for validation by users and customers, particularly for user interface requirements. Come in all different shapes and sizes (paper prototype of computerized system, formal executable model, ...). The prototype will not be used in the final system, the code developed is for demonstration purposes.

Functional Test Design:

Functional tests at the system level must be developed (sooner or later). They can (and should) be derived from the requirements specification. Each

functional requirement should have an associated test, non-functional requirements are harder to validate with testing.

They are based on the use cases initially. Each requirements test case must be traced to its requirements. Creating requirements tests is an effective validation technique. Designing these tests may reveal errors in the specification. Missing or ambiguous information in the requirements description may make it difficult to formulate tests.

User Manual Development:

Forces a detailed look at requirements. Particularly useful if the application is rich in user interfaces / for usability requirements. Typical information in a user manual is description of the functionality, how to get out of trouble and how to install and get started with the system.

Reviews and Inspections:

A group of people read and analyse requirements, look for potential problems, meet to discuss the problems, and agree on a list of action items needed to address these problems. It is a widely used requirements validation technique, lots of evidence of effectiveness of the technique. It can be expensive as it needs careful planning and preparation, pre-review checking and needs appropriate checklists that must be developed and if necessary maintained.

Requirements - Functional and Non-functional

A requirement is something the product must do or a quality that the product must have. It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.

Functional Requirements

What the system needs to do. They describe the **functionality** and depend on the type of software, expected users and the type of system where the software is used.

Functional user requirements may be high-level statements of what the system should do.

Functional system requirements should describe the system services in detail.

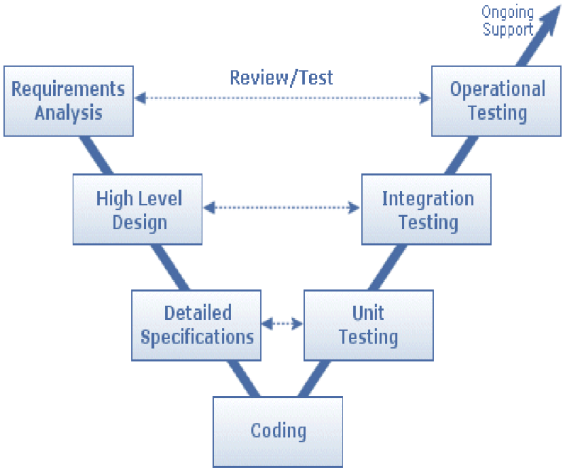
Non-functional Requirements

How the system needs to behave. These define **system properties** and constraints such as reliability, response time and storage requirements.

Non-functional requirements may be more **critical** than functional requirements, as if these are not met, the system is useless.

V Model

The V-model is an extension of the waterfall model. It is based on the association of a testing phase for each corresponding development stage. For each single phase in the Software Development Life Cycle (SDLC), there is a directly associated testing phase. Highly-disciplined model, the next phase only starts after completion of the previous phase.



There is an emphasis on testing. Testing is planned at every stage an directly mapped to development activities. Testing ensures the system is built correctly

Best Practices of the Rational Unified Process (RUP)

The Rational Unified Process (RUP) identifies six best practices for software development, particularly suited for object-oriented programming.

1. **Develop Software Iteratively:** this practice supports dividing the project into smaller increments, each delivering a subset of the final system. By focusing on delivering the highest priority early, the team ensures customer feedback is incorporated frequently. Iterative development also helps to identify and address risks early in the project lifecycle.
2. **Manage Requirements:** explicit documentation and continuous tracking of requirements ensure alignment with customer needs. This practice involves

analysing the impact of the requirements before accepting them, maintaining project stability. It ensures evolving customer demands are accommodated without jeopardizing project timelines or goals.

3. **Use Component Based Architectures:** this approach organizes the system into reusable and loosely coupled components. By promoting reuse, this practice enhances scalability and simplifies future modifications. Components interact via well-defined interfaces, ensuring a clear separation of concerns and reducing integration issues.
4. **Visually Model Software:** graphical models, particularly UML ones, provide clear static and dynamic views of the system. Visual models improve communication among stakeholders by offering a shared understanding of the system's structure and behaviour. These models also serve as blueprints, ensuring consistency across design and implementation.
5. **Verify Software Quality:** RUP emphasizes rigorous testing to ensure that the software meets defined quality standards. This includes automated tests, manual reviews and performance assessments to detect defects early. Ensuring high quality reduces post-deployment failures and enhances user satisfaction.
6. **Control Changes to Software:** a robust change management system handles modifications systematically. This includes maintaining version control, tracking changes and assessing their impact before implementation. Change control minimizes disruptions and ensures all stakeholders are informed of updates.

Phases of the Rational Unified Process

The Rational Unified Process (RUP) organizes software development into four distinct phases. Each phase consists of iterative workflows covering essential activities such as business modelling, requirements gathering, design, implementation, testing and deployment. The RUP's structure emphasizes iterative development and continuous improvement.

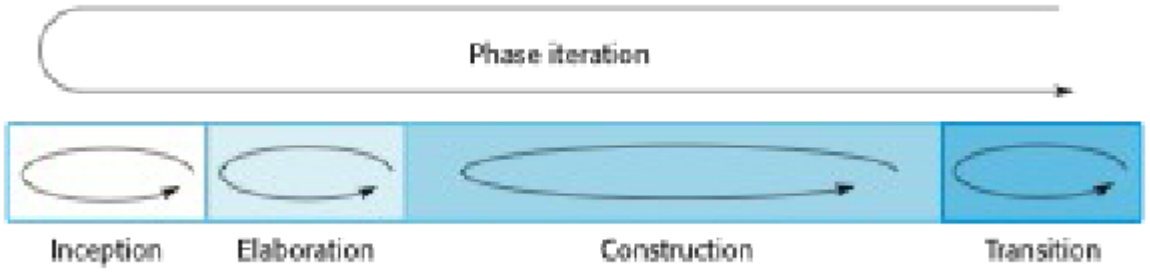
1. **Inception Phase:** the goal is to establish a business case and gain stakeholder buy-in. Key workflows include capturing initial requirements, cost-benefit analysis, project scope definition and risk assessment. Deliverables include an initial use-case model and a disposable prototype.

Effort is lower compared to other phases as it focuses on conceptualizing the project.

2. **Elaboration Phase:** this phase refines requirements and develops the architectural foundation. Key activities include analyzing use cases (80% of them completed), creating a domain model, and producing architectural documents. Risk assessments are revised and prototype user interfaces are developed. The effort increases significantly as this phase involves deeper exploration of the system's structure and scope.
3. **Construction Phase:** this phase focuses on implementing and testing the system's functionality. Activities include fleshing out stubs, completing detailed coding and ensuring functional stability. Testing becomes more comprehensive and iterative builds are generated to integrate components. Effort is the highest here since it involves intensive development and frequent iterations.
4. **Transition Phase:** this phase transitions the system to users. Activities include deploying the system, providing training and ensuring technical support. The development teams reduces in size as the maintenance team takes over. Effort decreases as the focus shifts to user adoption and system stabilization.

Workflow Activities:

- **Business Modeling**: Models the business processes using use cases.
- **Requirements**: Defines system requirements and identifies actors.
- **Analysis and Design**: Produces architectural, component, and object models.
- **Implementation**: Develops and organizes system components.
- **Testing**: Iteratively verifies system quality and validates functionality.
- **Deployment**: Releases the product and ensures successful installation.



Role of Software Architecture

Software architecture plays a critical role in the design and development of a software system. It defines the system's fundamental structure by identifying its major components and their interactions. As an early design activity, it provides the framework for connecting system requirements to implementation. Architectural decisions influence critical non-functional aspects like performance, scalability, and security, ensuring that the system meets its operational objectives.

Architecture acts as a communication tool among stakeholders, offering an abstract view of the system for discussions and planning. It facilitates system analysis, allowing teams to evaluate whether the design satisfies non-functional requirements like maintainability and reliability. Architectural patterns and styles promote reuse, reducing development costs and enabling efficient adaptation to similar projects. Furthermore, it organizes the system into manageable modules, supporting incremental development and simplifying testing and maintenance.

By serving as the foundation for detailed design and implementation, software architecture ensures system consistency and adaptability. It governs the integration of components, enabling seamless collaboration across development teams and technologies.

Advantages of using an explicit Architecture

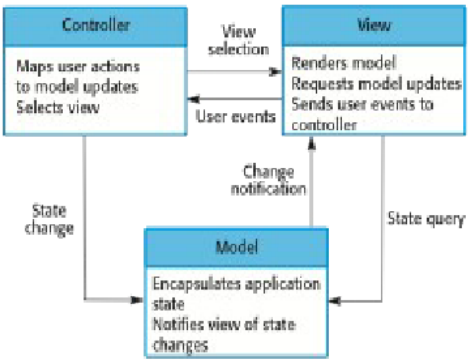
1. **Enhances Stakeholder Communication:** explicit architecture serves as a shared blueprint that stakeholders can use to discuss the system. By providing a high-level view of the system, it facilitates alignment among stakeholders, ensuring everyone has a common understanding of the system’s goals and design. This reduces misunderstandings and helps in gathering and validating requirements effectively.

2. **Facilitates System Analysis:** an explicit architecture allows teams to analyze the system for compliance with non-functional requirements such as performance, scalability and maintainability. It helps identify potential bottlenecks or vulnerabilities early in the design process, enabling proactive resolution. This analytical advantage ensures that the system's architecture supports both current and future needs, reducing the risk of costly rework.
3. **Supports Large-Scale Reuse:** explicit architectures often include reusable components or patterns, which can be applied to multiple projects within the same domain. This reuse accelerates development and enhances consistency across projects, improving overall productivity.

Architectures

Model View Controller (MVC) Pattern

- **Description:** separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on the data. The View component defines and manages how the data is presented to the user. The Controller component manages user interaction (key presses, mouse click, ...) and passes these interactions to the View and the Model.



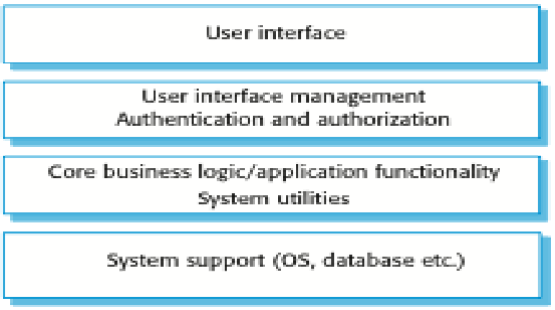
- **Example:** Database as model component, GUI as view component and logic as controller.
- **When used:** when there are multiple ways to view and interact with data and when the future requirements for interaction and presentation of data are unknown.
- **Advantages:** allows the data to change independently of its presentation and vice versa. Supports presentation of the same data in different ways

with changes made in one representation shown in all of them.

- **Disadvantages:** can involve additional code and code complexity when the data model and interactions are simple.

Layered Architecture

Used to model the interfacing of sub-systems. Organises the system into a set of layers (abstract machines) each of which provide a set of services. Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected. Often it is artificial to structure systems this way.

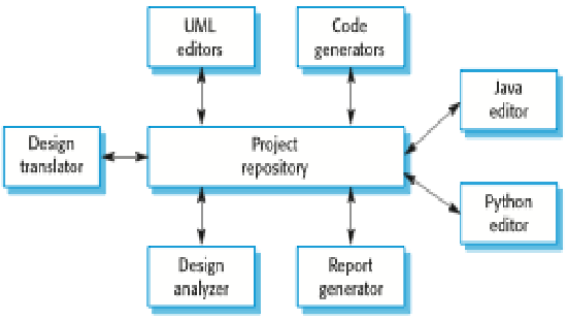


- **Description:** organizes the system into layers with related functionality associated with each layer. A layer provides services to the layer above it so the lowest-level layers represent core services that are likely to be used through the system.
- **Example:** a layered model of a system for sharing copyright documents held in different libraries.
- **When used:** when building new facilities on top of existing systems, when the development is spread across several teams with each team responsibility for a layer of functionality, when there is a requirement for multi-level security.
- **Advantages:** allows replacement of entire layers so long as the interface is maintained. redundant facilities (e.g. authentication) can be provided in each layer to increase the dependability of the system.
- **Disadvantages:** providing a clean separation between layers is often difficult and a high-level layer may have to interact directly with lower-level layers rather than through the layer immediately below it. Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer.

Repository Architecture

Sub-systems must exchange data. This may be done in two ways:

- Shared data is held in a central database or repository and may be accessed by all sub-systems.
- Each sub-system maintains its own database and passes data explicitly to other sub-systems.

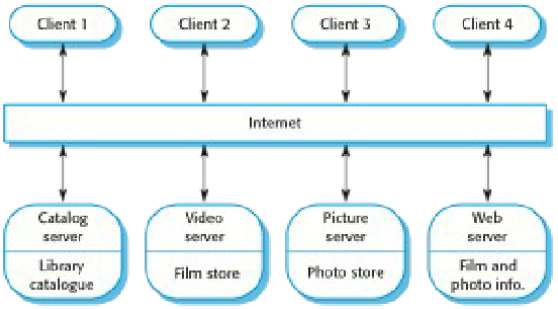


When large amounts of data are to be shared, the repository model of sharing is most commonly used as this is an efficient data sharing mechanism.

- **Description:** all data in a system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through the repository.
- **Example:** IDE where the components use a repository of system design information. Each software tool generates information which is then available for use by other tools.
- **When used:** when you have a system in which large volumes of information are generated that has to be stored for a long time. You may also use it in data-driven systems where the inclusion of data in the repository triggers an action or tool.
- **Advantages:** components can be independent, they do not need to know of the existence of other components. Changes made by one component can be propagated to all components. All data can be managed consistently as it is all in one place.
- **Disadvantages:** the repository is a single point of failure so problems in the repository affect the whole system. May be inefficient in organizing all communication through the repository. Distributing the repository across several computers may be difficult.

Client-server Architecture

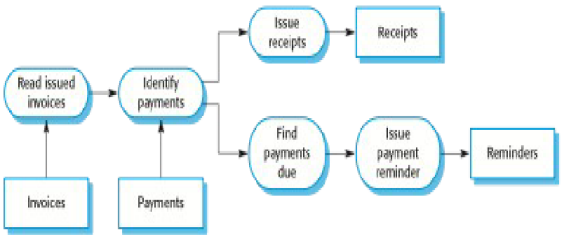
Distributed system model which shows how data and processing is distributed across a range of components. It can be implemented on a single computer. Set of stand-alone servers which provide specific services such as printing, data managements, ... Set of clients which call in these services. Network which allows clients to access servers.



- **Description:** in a client-server architecture, the functionality of the system is organized into services, with each service delivered from a separate server. Clients are users of these services and access servers to make user of them.
- **Example:** film and video/DVD library organized as a client-server system.
- **When used:** when data in a shared database has to be accessed from a range of locations. Because servers can be replicated, may also be used when the load on a system is variable.
- **Advantages:** servers can be distributed across a network. General functionality can be available to all clients and does not need to be implemented by all services.
- **Disadvantages:** each service is a single point of failure so susceptible to denial of service attacks or server failure. Performance may be unpredictable because it depends on the network as well as the system. There may be management problems if servers are owned by different organizations.

Pipe and Filter Architecture

Functional transformations process their inputs to produce outputs. May be referred to as a pipe and filter model (as in UNIX shell). Variants of this approach are very common. When transformations are sequential,



this is a batch sequential model which is extensively used in data processing systems. Not really suitable for interactive systems.

- **Description:** the processing of the data in a system is organized so that each processing component (filter) is discrete and carries out one type of data transformation. The data flows (as in a pipe) from one component to another for processing.
- **Example:** pipe and filter system used for processing invoices.
- **When used:** in data processing applications (both batch- and transaction-based) where inputs are processed in separate stages to generate related outputs.
- **Advantages:** easy to understand and supports transformation reuse. Workflow style matches the structure of many business processes. Evolution by adding transformation is straightforward.
- **Disadvantages:** the format for data transfer has to be agreed upon between communicating transformations. Each transformation must parse its input and unparse its output to the agreed form. This increases system overhead and may mean that it is impossible to reuse functional transformation that use incompatible data structures.

Project Management

Key Components of a Successful Project Plan

1. **Strategic Planning:** defines the project's objectives by analyzing the marketplace or organizational context. This involves understanding the industry, competition and target audience to ensure the project meets the needs. A well-structured strategy align the project's goals with organizational priorities and external demands.
2. **Developing the Product or Service:** the product or service must be tailored to achieve the defined business goals. It is essential to document all specifications early to prevent scope creep or unnecessary changes during the project. This component ensures that efforts remain focused on outcomes that drive real business value.

3. **Marketing:** marketing focuses in gaining buy-in from stakeholders and end-users by effectively communicating the project's benefits. Even for internal projects, marketing plays a crucial role in convincing users to accept the change. A successful marketing effort helps overcome resistance to change and ensure smooth project implementation.
4. **Support Mechanisms:** projects require robust support infrastructure, including IT tools, skilled staff and adequate facilities. This ensures the team has all the necessary resources to execute tasks effectively and address challenges. Support also extends post-implementation phases, ensuring the system operates efficiently.
5. **People and Stakeholders:** stakeholders, from sponsors to end-users, play distinct roles in project success. Each category of people involved must be committed and aligned with the project's vision. Effective stakeholder management ensures seamless collaboration and collective problem-solving.

Role of Key Stakeholders in Project Plan Development

1. **Project Sponsor:** the sponsor is the driving force behind the project's initiation and direction. They define the business objectives and ensure sufficient resources, such as budget and personnel, are allocated. Sponsors also establish priorities and influence the project's timeline by making strategic decisions. While they provide a high-level overview, they avoid micromanagement, allowing the team to handle detailed planning. Their support is crucial for securing organizational commitment and ensuring the project aligns with broader goals.
2. **Project Manager:** the project manager translates the sponsor's objectives into a detailed project plan. They define the scope, budget and schedule while managing risks and changes throughout the project. Acting as a bridge between stakeholders, the manager ensures communication is clear and expectations are aligned. They also motivate the team, resolve conflicts and monitor progress to ensure the project stays on track. A skilled project manager anticipates challenges and adapts plans to maintain momentum toward project completion.
3. **Team Members:** team members execute the tasks necessary to deliver the project's objectives, contributing their specialized skills. They are responsible for meeting deadlines, resolving issues related to their works and ensuring deliverable quality. Experienced team members often mentor

junior staff, promoting collective productivity and knowledge sharing. Their feedback during the planning phase is essential to identify potential risks and practical solutions. Engaged and cooperative team members are critical to the overall success of the project.

4. **Client, customer:** it is the person or company paying for the project. This could be internal or external to the company. The client or customer will only fully approve payment in acceptance that the requirements have been achieved.
5. **End User:** the end-user is the key final user of the system. Dissatisfied end-users can collapse a project. End user modelling is essential to meeting unstated requirements. There are novice users, regular users and power users.

Strategies for Managing Risk

In project management, risk planning involves three primary strategies for managing risks: avoidance, minimization and contingency planning.

1. **Avoidance:** this strategy focuses on eliminating potential risks before they occur. By taking proactive measures, the likelihood of the risk materializing is greatly reduced. For example, a software development team might avoid compatibility issues by standardizing the technology stack across all components of the project.
2. **Minimization:** this approach seeks to reduce the impact of risks that cannot be entirely avoided. Preventive measures are implemented to soften the consequences if the risk occurs. For example, a project team might cross-train members to ensure that tasks can still progress even if a key team member becomes unavailable.
3. **Contingency Planning:** this strategy prepares for the aftermath of risks that do occur. It involves developing a detailed, well-researched plan to manage and recover from the effects of risks. For example, in a scenario where funding cuts are a possibility, a team might prepare a comprehensive report highlighting the project's critical contributions to organizational goals to secure continued support.

Cost Benefit Analysis and Use of Payback Period

Cost-Benefit Analysis (CBA) is a systematic process in project planning used to evaluate whether a project is a sound investment and to compare multiple projects. It quantifies both the costs and benefits of a project in monetary terms, adjusted for the time value of money. This allows organizations to determine if the benefits outweigh the costs and by how much, ensuring informed decision-making.

The Payback Period is a simple tool within CBA that calculates the time required to recover the initial investment. A shorter payback period is generally preferable, as it indicated quicker cost recovery. For example, a project with a cost of 100,000€ and annual cash inflows of 25,000€ would have a payback period of four years.

However, the payback period has limitations. It does not consider the time value of money, making it less accurate for long-term projects. Additionally, it assumes uniform cash flows and ignores benefits or costs beyond the payback period, which can lead to an incomplete evaluation of a project's overall profitability.

The payback period is often complemented by more sophisticated methods like Net Present Value (NPV) to ensure a comprehensive assessment of project viability.

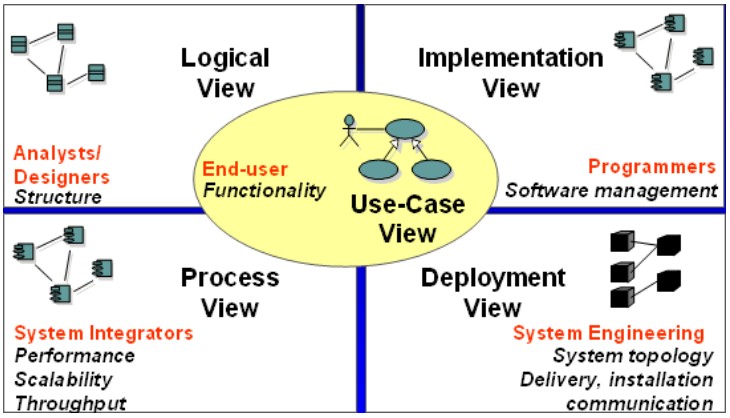
4 + 1 View of the Rational Unified Process

The 4 + 1 view model in the Rational unified Process is a framework for representing a system's architecture from multiple perspectives to effectively communicate with stakeholders. Each view serves for specific concerns and provides a comprehensive understanding of the system architecture. The five views are:

1. **Logical View:** focuses on the functional requirements of the system and identifies the major design components of the system.
2. **Processes View:** captures the system's dynamic behaviour, emphasizing interactions between processes and addressing runtime performance and scalability.
3. **Development View:** concentrates on the organization of the software in the development environment, highlighting modules, libraries and components.

This view is crucial to manage version control and collaboration.

4. **Physical View:** describes the system's deployment in hardware environments, showing how software components are mapped to physical nodes.
5. **Use Case/Scenario View (+1):** serves as a unifying element that ties together the other four view using specific use cases or scenarios.



Work Breakdown Structure (WBS)

The Work Breakdown Structure (WBS) is a hierarchical decomposition of a project into smaller, manageable units called work packages. It begins with the high-level project concept and breaks it down into successively finer levels of detail, including tasks, subtasks and activities. Each work package represents a distinct deliverable or task within the project.

WBS facilitates detailed project planning by clearly defining all the elements needed for project execution. It provides clarity on the scope of work and helps allocate resources effectively. Each work package is assigned a budget and timeline, aiding in cost estimation and scheduling. It also improves communication among stakeholders by providing a clear and visual representation of project components.

Additionally, the WBS allows for easier tracking of project progress, as smaller, discrete tasks can be monitored more effectively. It servers as the foundation for creating an action plan and setting milestones. Overall, the WBS ensures that no critical components or activities are overlooked during project planning.

Role of the Action Plan

The Action Plan outlines the specific steps needed to execute the tasks identified in the WBS. It includes essential details such as what needs to be done, when it must be completed, and who is responsible for each task. By organizing activities into a clear and structured format, the Action Plan ensures all team members understand their roles and responsibilities.

The Action Plan provides a timeline for activities, ensuring tasks are started and completed in a logical sequence. It addresses dependencies between activities, such as tasks that must be completed before others can begin.

This plan is vital for resource allocation, as it specifies the tools, personnel, and materials required for each task. It also serves as a baseline for monitoring and controlling project progress, as managers can track tasks against deadlines. Adjustments can be made to the plan as needed to address delays or unforeseen challenges.

Moreover, the Action Plan facilitates effective communication and coordination among team members and stakeholders. It enables consistent progress reviews and helps ensure that the project stays aligned with its goals. In summary, the Action Plan is the operational blueprint that transforms high-level planning into actionable steps.