

UNIT-III

WORKFLOWS AND CHECKPOINTS OF PROCESS

WORKFLOWS OF THE PROCESS:

A software process consists of different **phases and iterations**. Each phase contains a number of activities that are performed to develop the required software artifacts.

A **workflow** is a group of related activities that are performed together in a mostly sequential manner to produce and improve software artifacts.

There are **seven top-level workflows**.

1. Management Workflow

It controls the software process and ensures that the project is properly managed for all stakeholders.

2. Environment Workflow

It automates the software process and develops and maintains the environment needed for software development.

3. Requirements Workflow

It analyzes the problem and develops and improves the requirements of the software.

4. Design Workflow

It models the solution and develops and improves the software architecture and design.

5. Implementation Workflow

It programs the software components and develops the implementation and deployment artifacts.

6. Assessment Workflow

It checks the progress and quality of the software process and product.

7. Deployment Workflow

It transfers the completed software product to the user.

IMPORTANT CHARACTERISTICS OF THE PROCESS

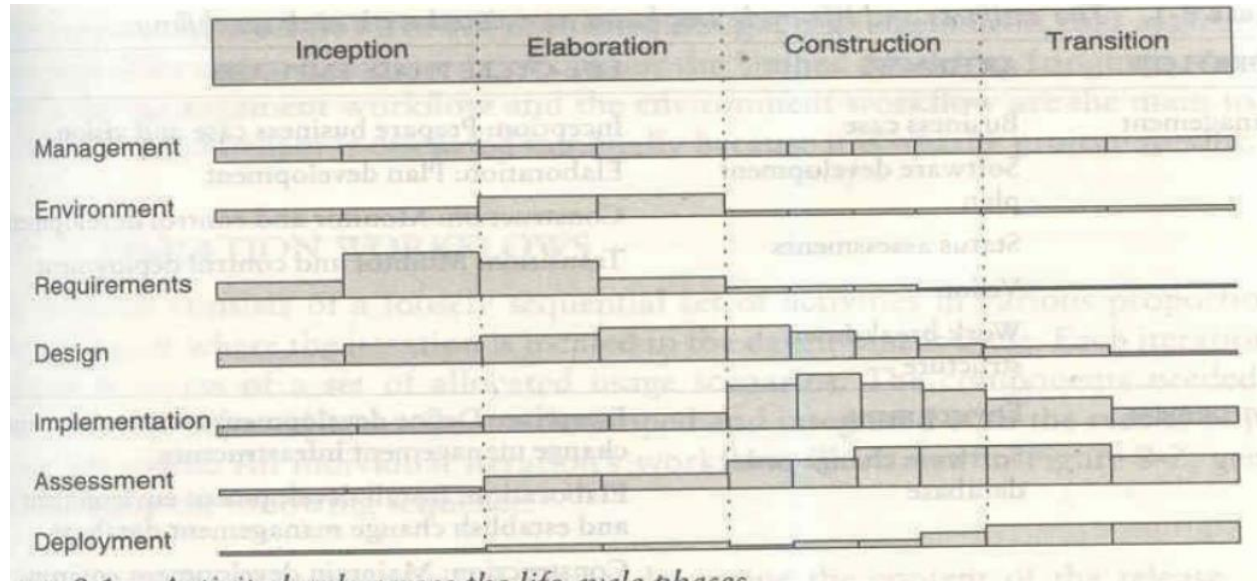


Fig--Activities levels across the life cycle

Architecture-first approach:

- Important requirements, design, implementation and assessment activities are performed before full-scale Construction.
- The architecture is developed, implemented and tested early.
- The purpose is to make sure that the architecture is suitable before developing the complete product.

Iterative life-cycle process:

- Software development is performed through repeated iterations.
- A workflow may need to be performed more than once to achieve the required result.
- The number of iterations depends on the project.

Round-trip Engineering:

- The environment is treated as an important workflow.
- It provides the tools, methods and notations needed to produce and maintain software artifacts.

Demonstration-based Approach:

- Implementation and assessment begin early in the life cycle.
- Executable parts of the evolving architecture are constructed and assessed early.

Artifacts and Life-Cycle Emphases Associated with Each Workflow

The modern software process organizes development activities into different workflows. Each workflow produces specific artifacts and has different emphasis during the four life-cycle phases: Inception, Elaboration, Construction, and Transition.

WORKFLOW	ARTIFACTS	LIFE-CYCLE PHASE EMPHASIS
Management	Business case	Inception: Prepare business case and vision
	Software development plan	Elaboration: Plan development
	Status assessments	Construction: Monitor and control development
	Vision	Transition: Monitor and control deployment
	Work breakdown structure	
Environment	Environment	Inception: Define development environment and change management infrastructure
	Software change order database	Elaboration: Install development environment and establish change management database
		Construction: Maintain development environment and software change order database
		Transition: Transition maintenance environment and software change order database
Requirements	Requirements set	Inception: Define operational concept
	Release specifications	Elaboration: Define architecture objectives
	Vision	Construction: Define iteration objectives
		Transition: Refine release objectives
Design	Design set	Inception: Formulate architecture concept
	Architecture description	Elaboration: Achieve architecture baseline
		Construction: Design components
		Transition: Refine architecture and components
Implementation	Implementation set	Inception: Support architecture prototypes
	Deployment set	Elaboration: Produce architecture baseline
		Construction: Produce complete componentry
		Transition: Maintain components
Assessment	Release specifications	Inception: Assess plans, vision, prototypes
	Release descriptions	Elaboration: Assess architecture
	User manual	Construction: Assess interim releases
	Deployment set	Transition: Assess product releases
Deployment	Deployment set	Inception: Analyze user community
		Elaboration: Define user manual
		Construction: Prepare transition materials
		Transition: Transition product to user

FIG-The artifacts and life-cycle emphases associated with each workflow

Some themes of the conventional process are not carried forward as separate workflows in the modern process:

1. Documentation
 - Documentation is not treated as a separate activity.
 - Most documentation is produced as a secondary by-product of other activities.
2. Quality Assurance
 - Quality assurance is not treated as a separate workflow.
 - Quality is incorporated into the engineering and management activities.

ITERATION WORKFLOWS

An **iteration** is a loosely sequential set of activities performed in different proportions depending on its position in the development cycle.

Each iteration is defined based on a set of **allocated usage scenarios**. The components required to implement the selected scenarios are developed and integrated with the results of previous iterations.

An individual iteration generally consists of the following workflow:
Management → Requirements → Design → Implementation → Assessment → Deployment

Workflow of an Iteration

An iteration mainly receives:

1. **Allocated Usage Scenarios** – The usage scenarios selected for development in the iteration.
2. **Results from Previous Iteration** – Results and baselines produced by earlier iterations.

Management is the planning stage of the iteration. It decides what should be included in the release, prepares a detailed plan for the iteration, and assigns the required work and tasks to the development team.

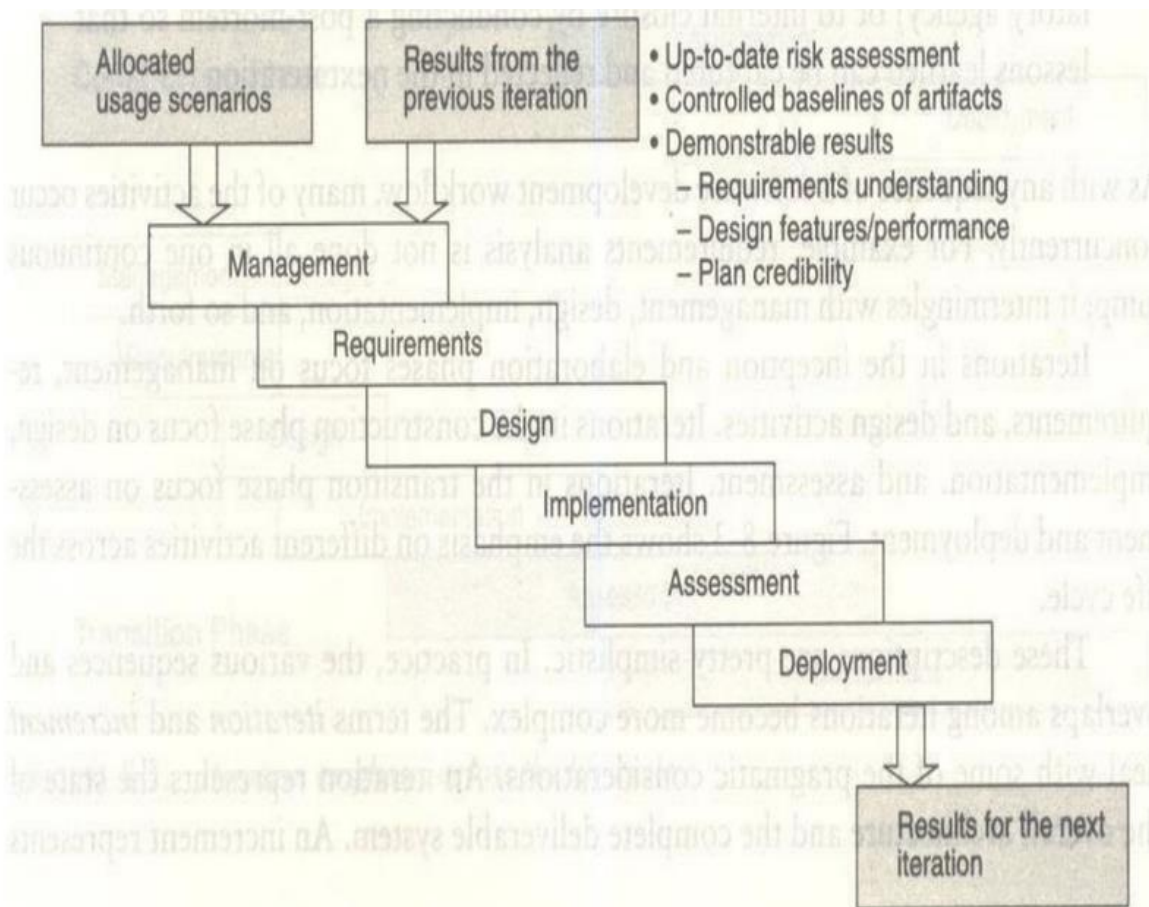


Figure :The Workflow of an Iteration

Environment deals with maintaining the software change-order database. It records new baselines and changes made to existing baselines for the product, testing, and development environment.

Requirements involves studying the existing plan, architecture, and requirements. The requirements are refined so that the required use cases can be demonstrated at the end of the iteration. The requirements are also updated whenever changes occur during development.

Design involves improving the existing architecture and design. The design model and test model are developed further so that the required features can be demonstrated and tested. The design documents are updated whenever changes are made during development.

Implementation is the development stage. New components are developed or acquired, and existing components are modified or improved. All new and modified components are integrated with the existing system and tested.

Assessment checks the results of the iteration. It verifies whether the required evaluation criteria have been satisfied and whether the current system baselines have the required quality. If any rework is required, it is decided whether it should be completed before deployment or moved to the next release.

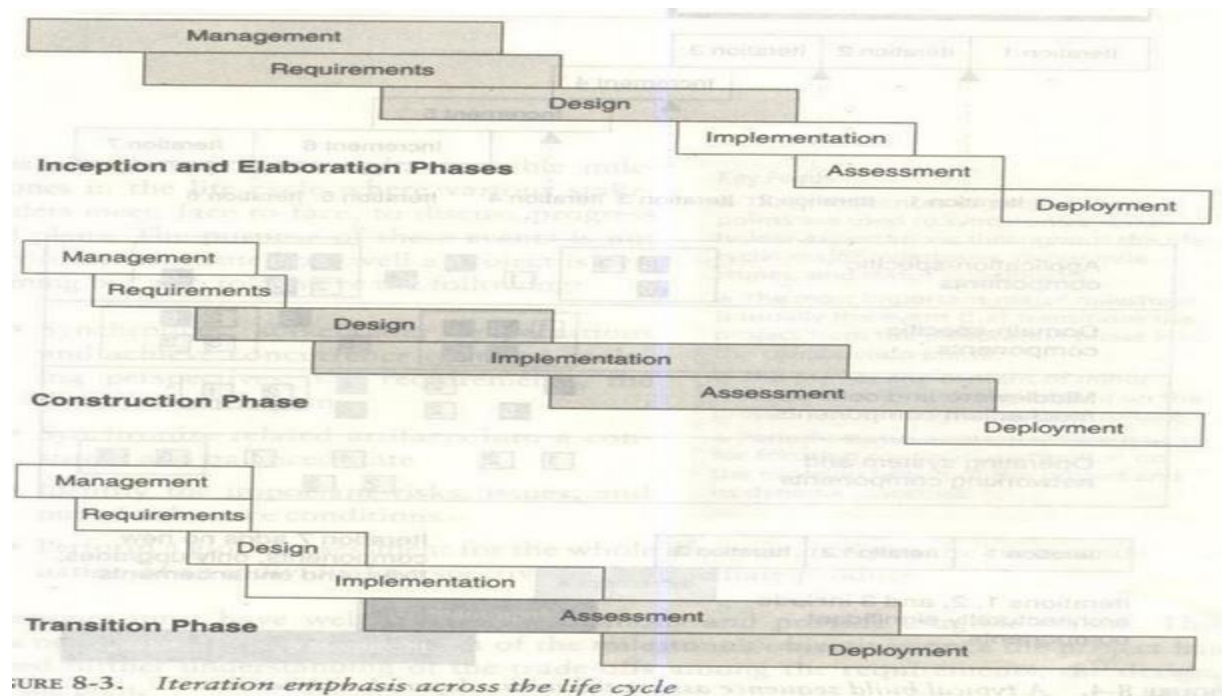
Deployment is the final stage of the iteration. The completed release is delivered to the required external organization or the iteration is closed internally through a postmortem. The lessons learned are recorded and used to improve the next iteration.

In a software development process, all activities are not performed one after another. Many activities can happen at the same time and may overlap with each other. For example, requirements analysis does not finish completely before design or implementation starts. Requirements work may continue while management, design, and implementation activities are also taking place.

The importance of different activities changes in each phase of the software life cycle. In the Inception and Elaboration phases, the main focus is on management, requirements, and design. In the Construction phase, the focus is mainly on design, implementation, and assessment. In the Transition phase, the main focus is on assessment and deployment. Therefore, the same activities may continue throughout development, but their importance changes from one phase to another.

In practice, the sequence of activities and their overlap can be more complex. Two important terms used here are iteration and increment. An iteration represents the current state of the overall architecture and the complete deliverable system. An increment represents the work that is currently in progress. This work is combined with the previous iteration to produce the next iteration.

The figure 8-3 shows how the importance of different activities changes during the four life-cycle phases. During **Inception and Elaboration**, management, requirements, and design are given more attention. During **Construction**, design, implementation, and assessment receive more attention. During **Transition**, assessment and deployment become the main focus.



Typical Build Sequence

Figure 8-4 shows a typical way in which a software system is developed using iterations and increments. The progress of development can be measured using things such as the percentage of components that are under configuration control and the percentage of demonstrable use cases.

The figure also shows a layered architecture. The layers include application-specific components, domain-specific components, middleware and common mechanism components, and operating system and networking components.

During the early iterations, important architectural components are developed. In particular, Iterations 1, 2, and 3 contain architecturally significant components.

As development continues, new increments are added to the existing system. These increments help improve and complete the system. Finally, Iteration 7 does not add new components. Instead, it mainly contains upgrades, fixes, and enhancements to the existing system.

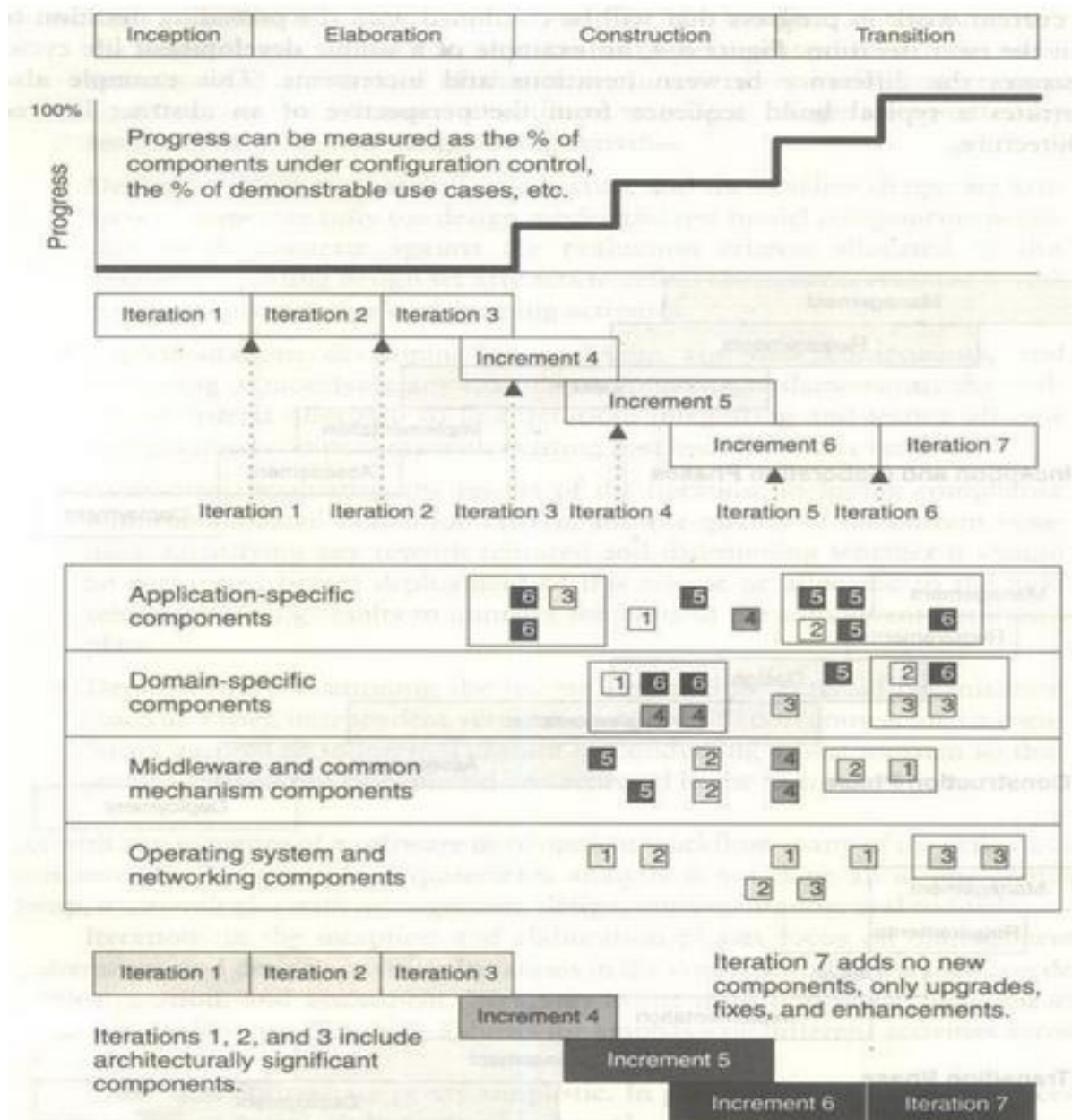


FIGURE 8-4. A typical build sequence associated with a layered architecture

CHECKPOINTS OF THE PROCESS

- In a software project, it is important to have checkpoints where the project team and stakeholders meet and review the progress of the project. These checkpoints help everyone understand what has been completed, what is currently happening, and what needs to be done next. They also help the team agree on the requirements, design, and project plan.
- Checkpoints are also useful for keeping project documents and activities consistent with each other. They help the team identify important risks, problems, and situations that are outside the planned limits. They also provide an overall view of the project instead of looking at only one activity or one product.
- A checkpoint should have clear expectations and clear results.

Three types of joint management reviews are conducted throughout the process:

Major Milestones

A major milestone is a major review conducted at the end of a development phase. It gives management and the development team an overall view of the project. It is used to identify major issues, make sure that management and technical teams have the same understanding, and check whether the objectives of that phase have been achieved.

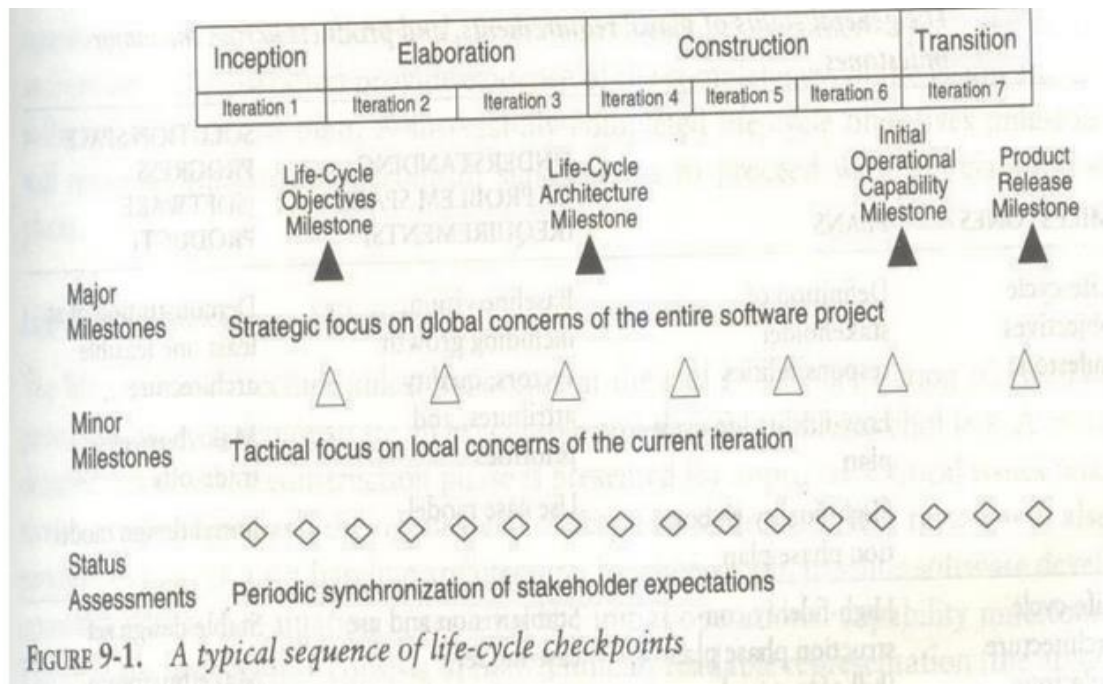
Minor Milestones

A minor milestone is a review conducted during an iteration. It focuses on the detailed work completed during that iteration. It is also used to decide whether the team can continue with the next work.

Status Assessments

A status assessment is a regular review of the project. It gives management frequent information about the progress of the project. It helps management understand how the project is progressing and whether there are any issues that need attention.

MAJOR MILESTONES



Different stakeholders have very different concerns:

- **Customers:** Schedule and budget estimates, feasibility, risk assessment, requirements understanding, progress, product line compatibility
- **Users:** Consistency with requirements and usage scenarios, potential for accommodating growth, quality attributes
- **Architects and systems engineers:** Product line compatibility, requirements changes, trade-off analyses, completeness and consistency, balance among risk, quality, and usability
- **Developers:** Sufficiency of requirements detail and usage scenario descriptions, . Frameworks for component selection or development, resolution of development risk, product line compatibility, sufficiency of the development environment
- **Maintainers:** Sufficiency of product and documentation artifacts, understandability, interoperability with existing systems, sufficiency of maintenance environment

- **Others:** Possibly many other perspectives by stakeholders such as regulatory agencies, independent verification and validation contractors, venture capital investors, subcontractors, associate contractors, and sales and marketing teams

Table 9-1: The General Status of Plans, Requirements, and Products Across the Major Milestones

MILESTONES	PLANS	UNDERSTANDING OF PROBLEM SPACE (REQUIREMENTS)	SOLUTION SPACE PROGRESS (SOFTWARE PRODUCT)
Life-cycle objectives milestone	Definition of stakeholder responsibilities Low-fidelity life-cycle plan High-fidelity elaboration phase plan	Baseline vision, including growth vectors, quality attributes, and priorities Use case model	Demonstration of at least one feasible architecture Make/buy/reuse trade-offs Initial design model
Life-cycle architecture milestone	High-fidelity construction phase plan (bill of materials, labor allocation) Low-fidelity transition phase plan	Stable vision and use case model Evaluation criteria for construction releases, initial operational capability Draft user manual	Stable design set Make/buy/reuse decisions Critical component prototypes
Initial operational capability milestone	High-fidelity transition phase plan	Acceptance criteria for product release Releasable user manual	Stable implementation set Critical features and core capabilities Objective insight into product qualities
Product release milestone	Next-generation product plan	Final user manual	Stable deployment set Full features Compliant quality

Life-Cycle Objectives Milestone:

At the **Life-Cycle Objectives Milestone**, stakeholder responsibilities and the initial life-cycle plans are defined. The baseline vision, use-case model, and at least one feasible architecture are established.

Life-Cycle Architecture Milestone:

At the **Life-Cycle Architecture Milestone**, the development and transition plans become more detailed. The vision and use-case model become stable, construction evaluation criteria are identified, and the design and critical component prototypes become stable

I. Requirements	
A.	Use case model
B.	Vision document (text, use cases)
C.	Evaluation criteria for elaboration (text, scenarios)
II. Architecture	
A.	Design view (object models)
B.	Process view (if necessary, run-time layout, executable code structure)
C.	Component view (subsystem layout, make/buy/reuse component identification)
D.	Deployment view (target run-time layout, target executable code structure)
E.	Use case view (test case structure, test result expectation)
1.	Draft user manual
III. Source and executable libraries	
A.	Product components
B.	Test components
C.	Environment and tool components

Presentation Agenda	
I. Scope and objectives	
A.	Demonstration overview
II. Requirements assessment	
A.	Project vision and use cases
B.	Primary scenarios and evaluation criteria
III. Architecture assessment	
A.	Progress
1.	Baseline architecture metrics (progress to date and baseline for measuring future architectural stability, scrap, and rework)
2.	Development metrics baseline estimate (for assessing future progress)
3.	Test metrics baseline estimate (for assessing future progress of the test team)
B.	Quality
1.	Architectural features (demonstration capability summary vs. evaluation criteria)
2.	Performance (demonstration capability summary vs. evaluation criteria)
3.	Exposed architectural risks and resolution plans
4.	Affordability and make/buy/reuse trade-offs
IV. Construction phase plan assessment	
A.	Iteration content and use case allocation
B.	Next iteration(s) detailed plan and evaluation criteria
C.	Elaboration phase cost/schedule performance
D.	Construction phase resource plan and basis of estimate
E.	Risk assessment

Demonstration Agenda	
I. Evaluation criteria	
II. Architecture subset summary	
III. Demonstration environment summary	
IV. Scripted demonstration scenarios	
V. Evaluation criteria results and follow-up items	

FIGURE 9-3. Default agendas for the life-cycle architecture milestone.

Initial Operational Capability Milestone:

At the **Initial Operational Capability Milestone**, the transition plan, product-release acceptance criteria, and user manual are finalized to a sufficient level, while the implementation set and core capabilities become stable.

Product Release Milestone:

Finally, at the Product Release Milestone, the next-generation product plan and final user manual are available. The software reaches a stable deployment state with its complete features and required quality. Thus, each milestone ensures that requirements, plans, product development, and quality progress together in a balanced and consistent manner.

MINOR MILESTONES

The number of **iteration-specific informal milestones** depends on the **content and duration of the iteration**.

For most iterations lasting **one to six months**, two minor milestones are generally required:

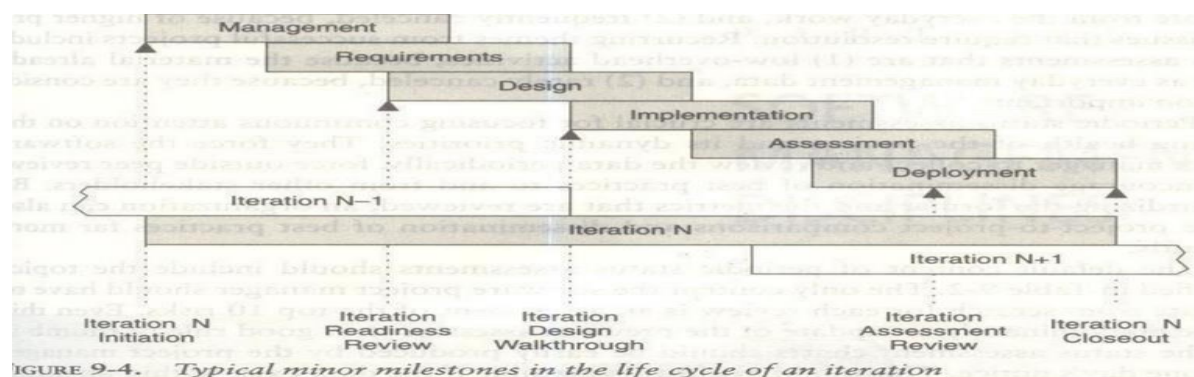
1. Iteration Readiness Review

Conducted at the **start of an iteration** to review the detailed iteration plan and the evaluation criteria assigned to it.

2. Iteration Assessment Review

Conducted at the **end of an iteration** to check whether the iteration achieved its objectives and satisfied the evaluation criteria. It also reviews the results, qualification tests, required rework, and impact on subsequent iterations.

The format and content of these minor milestones depend largely on the **project and organizational culture**. Different projects may therefore require different intermediate review points. In large-scale and unprecedented projects, additional design walkthroughs or other intermediate reviews may be required to assess progress and communicate results throughout the project.



Typical Minor Milestones in an Iteration Life Cycle

The figure shows how minor milestones are associated with different activities of an iteration. The major activities shown are Management, Requirements, Design, Implementation, Assessment, and Deployment.

An iteration begins with the Iteration N Initiation, followed by the Iteration Readiness Review. During the iteration, an Iteration Design Walkthrough may be conducted. At the later stage, the Iteration Assessment Review is conducted, followed by the Iteration Closeout. These milestones also connect the current iteration with the preceding and subsequent iterations.

PERIODIC STATUS ASSESSMENTS

Periodic status assessments are management reviews conducted at regular intervals, such as monthly or quarterly.

These assessments are used to review project progress and quality indicators, continuously monitor project activities, and maintain open communication among all stakeholders.

Periodic status assessments act as snapshots of the project. Although the assessment period may vary, conducting these assessments regularly causes the project history to be captured and documented.

Status assessments provide a mechanism for openly addressing, communicating, and resolving management issues, technical issues, and project risks.

They also provide objective data obtained directly from ongoing activities and changing product configurations.

Another purpose is to disseminate information about processes, progress, quality trends, practices, and experiences to and from all stakeholders through an open forum.

Status Assessments in Successful and Unsuccessful Project

In unsuccessful projects, status assessments often become high-overhead activities because the work required to prepare the status information is separate from the everyday project work.

They are also frequently cancelled because higher-priority issues require immediate attention.

In successful projects, status assessments are generally low-overhead activities because the required information already exists as everyday management data.

They are also rarely cancelled because the assessments are considered important.

Importance of Periodic Status Assessments

Periodic status assessments are important for maintaining continuous attention to the evolving health of the project and its changing priorities.

They require the software project manager to collect and review data periodically. They also provide opportunities for outside peer review and encourage the sharing of best practices among stakeholders.

Standardizing the format and metrics used in these assessments also allows project-to-project comparisons and makes the dissemination of best practices more efficient.

The default content of periodic status assessments should contain the topics identified in **Table 9-2**.

The main information that the software project manager needs to generate separately for each review is the **assessment of the top 10 risks**. Most of the other information is mainly an update of the previous assessment.

The status assessment charts should be easy for the project manager to produce, preferably **within one day's notice**. This is possible when the required data is available in an **automated environment**.

TOPIC	CONTENT
Personnel	Staffing Plans vs. Actual Plans Attritions, additions
Financial trends	Expenditure Plans vs. actual for the previous, current, and next major milestones Revenue forecasts
Top 10 risks	Issues and critically resolution plans Quantification (cost, time, quality) of exposure
Technical Progress	Configuration baseline schedules for major milestones Software management metrics and indicators Current change trends Test and quality assessments
Major milestone plans and results	Plan, schedule, and risks for next major milestone Pass/Fail results for all acceptance criteria
Total Product Scope	Total size, growth, and acceptance criteria perturbations

PROCESS PLANNING

Iterative Process Planning means that a software project plan is developed, reviewed, and updated repeatedly as the project progresses

Planning has two stages: Engineering stage – developing the plan, and Production stage – executing the plan.

Plans should be updated when requirements, understanding, or project conditions change.

Both under-planning and over-planning should be avoided; the right level of planning detail is important.

The Work Breakdown Structure (WBS) forms the basic structure or “architecture” of the project plan and should evolve throughout the project.

Cost and schedule should be estimated using:

- Macroanalysis: Top-down, project-level estimation.
- Microanalysis: Bottom-up, task-level estimation.

Planning errors should be identified and corrected early, because early corrections have less impact on project success

WORK BREAKDOWN STRUCTURE:

A good work breakdown structure and its synchronization with the process framework are critical factors in software project success. Development of a work breakdown structure dependent on the project management style, organizational culture, customer preference, financial constraints, and several other hard-to-define, project-specific parameters.

A WBS is simply a hierarchy of elements that decomposes the project plan into the discrete work tasks. A WBS provides the following information structure:

- A delineation of all significant work
- Clear task decomposition for assignment of responsibilities
- A framework for scheduling, budgeting, and expenditure tracking

Many parameters can drive the decomposition of work into discrete tasks: product subsystems, components, functions, organizational units, life-cycle phases, even geographies. Most systems have first-level decomposition by subsystem. Subsystems are then decomposed into their components, one of which is typically the software.

Conventional WBS Issues

Conventional work breakdown structures frequently suffer from three fundamental flaws.

1. They are prematurely structured around the product design.
2. They are prematurely decomposed, planned, and budgeted in either too much or Too little detail.
3. They are project-specific, and cross-project comparisons are usually difficult or impossible.

A conventional WBS breaks a software project according to its product architecture. The software is divided into subsystems, then components, and then activities such as requirements, design, coding, testing and documentation.

However, conventional WBSs have three major issues.

First, they are often structured around the product design too early. Since software architecture can change, this can make the WBS, budget and schedule difficult to change.

Second, conventional WBSs may be decomposed, planned and budgeted too early, with either too much or too little detail. Large projects tend to be overplanned, while small projects tend to be underplanned. Therefore, the WBS should evolve as more information becomes available.

Third, conventional WBSs are usually project-specific. Different projects use different structures and measurements, making it difficult to compare cost, schedule, productivity, quality and other performance measures across projects.

Conventional work breakdown structure, following the product hierarchy

Management in figure below :



Evolutionary Work Breakdown Structures

An evolutionary WBS should organize the planning elements around the process framework rather than the product framework. The basic recommendation for the WBS is to organize the hierarchy as follows:

- **First-level WBS:** Elements are the workflows (management, environment, Requirements, design, implementation, assessment, and deployment).
- **Second-level WBS:** Elements are defined for each phase of the life cycle (inception, elaboration, construction, and transition).
- **Third-level WBS:** Elements are defined for the focus of activities that produce the artifacts of each phase.

A default WBS consistent with the process framework (phases, workflows, and artifacts) this recommended structure provides one example of how the Elements of the process framework can be integrated into a plan. It provides a framework for estimating the costs and schedules of each element, allocating them Across a project organization, and tracking expenditures.

The structure shown is intended to be merely a starting point. It needs to be tailored to the specifics of a project in many ways.

Scale → Large projects need more levels and detailed substructures.

Organizational structure → Subcontractors or multiple organizations may require different WBS allocations.

Degree of custom development → Custom-built systems may need more detailed design and implementation activities.

Business context → Contract projects, commercial products, and large deployments may need more detailed management or deployment structures.

Precedent experience → Existing projects, legacy systems, and organizational standards can provide an existing WBS structure and lessons for new projects.

- A Management
 - AA Inception phase management
 - AAA Business case development
 - AAB Elaboration phase release specifications
 - AAC Elaboration phase WBS baselining
 - AAD Software development plan
 - AAE Inception phase project control and status assessments
 - AB Elaboration phase management
 - ABA Construction phase release specifications
 - ABB Construction phase WBS baselining
 - ABC Elaboration phase project control and status assessments
 - AC Construction phase management
 - ACA Deployment phase planning
 - ACB Deployment phase WBS baselining
 - ACC Construction phase project control and status assessments
 - AD Transition phase management
 - ADA Next generation planning
 - ADB Transition phase project control and status assessments
- B Environment
 - BA Inception phase environment specification
 - BB Elaboration phase environment baselining
 - BBA Development environment installation and administration
 - BBB Development environment integration and custom toolsmithing
 - BBC SCO database formulation
 - BC Construction phase environment maintenance
 - BCA Development environment installation and administration
 - BCB SCO database maintenance
 - BD Transition phase environment maintenance
 - BDA Development environment maintenance and administration
 - BDB SCO database maintenance
 - BDC Maintenance environment packaging and transition
- C Requirements
 - CA Inception phase requirements development
 - CAA Vision specification
 - CAB Use case modeling
 - CB Elaboration phase requirements baselining
 - CBA Vision baselining
 - CBB Use case model baselining
 - CC Construction phase requirements maintenance
 - CD Transition phase requirements maintenance

10.1 WORK BREAKDOWN STRUCTURES 145

- D Design
 - DA Inception phase architecture prototyping
 - DB Elaboration phase architecture baselining
 - DBA Architecture design modeling
 - DBB Design demonstration planning and conduct
 - DBC Software architecture description
 - DC Construction phase design modeling
 - DCA Architecture design model maintenance
 - DCB Component design modeling
 - DD Transition phase design maintenance
- E Implementation
 - EA Inception phase component prototyping
 - EB Elaboration phase component implementation
 - EBA Critical component coding demonstration integration
 - EC Construction phase component implementation
 - ECA Initial release(s) component coding and stand-alone testing
 - ECB Alpha release component coding and stand-alone testing
 - ECC Beta release component coding and stand-alone testing
 - ECD Component maintenance
 - ED Transition phase component maintenance
- F Assessment
 - FA Inception phase assessment planning
 - FB Elaboration phase assessment
 - FBA Test modeling
 - FBB Architecture test scenario implementation
 - FBC Demonstration assessment and release descriptions
 - FC Construction phase assessment
 - FCA Initial release assessment and release description
 - FCB Alpha release assessment and release description
 - FCC Beta release assessment and release description
 - FD Transition phase assessment
 - FDA Product release assessment and release descriptions
- G Deployment
 - GA Inception phase deployment planning
 - GB Elaboration phase deployment planning
 - GC Construction phase deployment
 - GCA User manual baselining
 - GD Transition phase deployment
 - GDA Product transition to user

FIGURE 10-2. Default work breakdown structure

PLANNING GUIDELINES:

Software projects cover a wide range of application domains. Therefore, making planning recommendations without considering the project context can be useful but risky.

Planning guidelines provide a **starting point for project planning**. They are based on the experience of many people and can provide a reliable basis for estimates. However, they should be modified according to the specific project.

Evolution of Planning Fidelity in WBS

Planning fidelity means the level of detail and accuracy of planning for different WBS elements. The planning becomes more detailed as the project progresses through its life cycle.

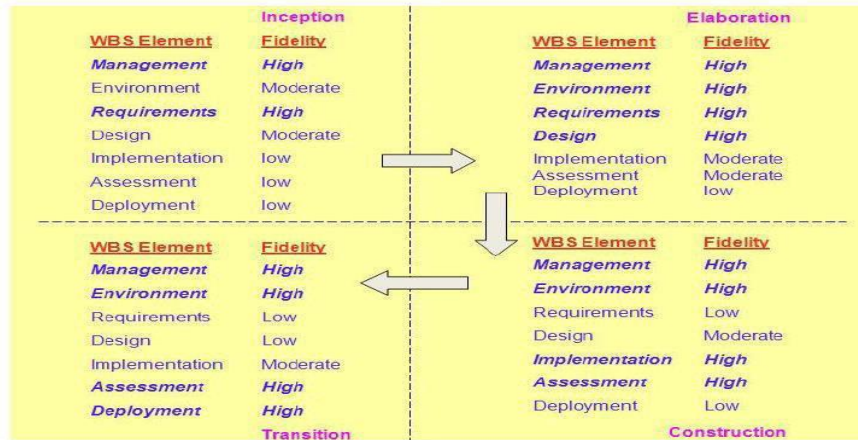
During **Inception**, Management and Environment have high fidelity. Requirements and Design have moderate fidelity, while Implementation and Deployment have low fidelity.

During **Elaboration**, Management, Environment, Requirements, and Design reach high fidelity. Implementation and Assessment have moderate fidelity, while Deployment remains low.

During **Construction**, most WBS elements have high fidelity. Deployment has moderate fidelity.

During **Transition**, all major WBS elements have **high fidelity**.

Thus, planning fidelity generally increases from **Inception** → **Elaboration** → **Construction** → **Transition**.



Project-Independent Planning Guidelines

Project-independent planning guidelines can be risky if they are followed blindly. Different projects have different parameters, business contexts, organizational cultures, and processes.

Therefore, these guidelines should be used as a **basis for planning**, but they must be adapted when the project requires different values.

Two important planning guidelines are provided. The first gives the **default cost allocation among first-level WBS elements**. The second gives the **default allocation of effort and schedule across life-cycle phases**.

These guidelines can be used with an initial total project cost to develop the initial staffing profile, resource allocation, project schedule, and WBS task budgets.

Table 10-1: WBS Budgeting Defaults

First Level WBS Element	Default Budget
Management	10%
Environment	10%
Requirement	10%
Design	15%
Implementation	25%
Assessment	25%
Deployment	5%
Total	100%

These percentages provide a **benchmark for planning**. Actual values may vary from project to project.

Domain	Inception	Elaboration	Construction	Transition
Effort	5%	20%	65%	10%
Schedule	10%	30%	50%	10%

Table 10-2: Default Distributions of Effort and Schedule by Phase

These values provide general guidelines for allocating **effort and schedule** across the life-cycle phases.

The values can vary depending on the application and its specific constraints. Therefore, the main purpose is not to follow these percentages exactly, but to understand **why a particular project may require a different distribution**.

THE COST AND SCHEDULE ESTIMATING PROCESS:

Project plans are derived from two perspectives: the forward-looking, top-down approach and the backward-looking, bottom-up approach.

Top-Down Approach

The top-down approach starts with the general requirements and constraints of the project. A macro-level budget and schedule are prepared first and then divided into lower-level budgets and intermediate milestones.

The software project manager develops an overall characterization of the required size, process, environment, people, and quality of the project.

A macro-level estimate of the total effort and schedule is then developed using a software cost estimation model.

The estimated effort is divided into a top-level WBS. The schedule is divided into major milestone dates, and the effort is distributed into a staffing profile. This produces a project-level plan.

Subproject managers then decompose each WBS element into lower levels using the top-level allocation, staffing profile, and major milestone dates as constraints.

Bottom-Up Approach

The bottom-up approach is backward-looking. It starts with the end of the project and develops micro-level budgets and schedules first. These are then combined into higher-level budgets and intermediate milestones.

The lowest-level WBS elements are divided into detailed tasks. The responsible WBS element manager estimates the budget and schedule for these tasks. These estimates include project-specific parameters.

The individual estimates are then combined and integrated into higher-level budgets and milestones. The biases of individual estimators are adjusted to provide a consistent basis for negotiation.

The bottom-up estimates are compared with the top-down budgets and schedule milestones. Differences are assessed, and adjustments are made to reach agreement between both estimates.

Balance Between the Two Approaches

Top-down estimating tends to exaggerate project management biases, often resulting in an overly optimistic plan. Bottom-up estimating tends to exaggerate performer biases, often resulting in an overly pessimistic plan.

Therefore, iteration is necessary. The results of one approach are used to validate and refine the other approach. Through multiple versions, the plan is gradually improved and ownership of the plan is developed at all levels of management.

Both approaches should be used together and in balance throughout the project life cycle.

During the engineering stage, the top-down approach dominates because there may not be enough detailed understanding or stability in task sequences for reliable bottom-up planning.

During the production stage, sufficient experience and planning detail are available, so the bottom-up approach dominates. The top-down approach can then be used mainly as a global assessment technique.

THE ITERATION PLANNING PROCESS

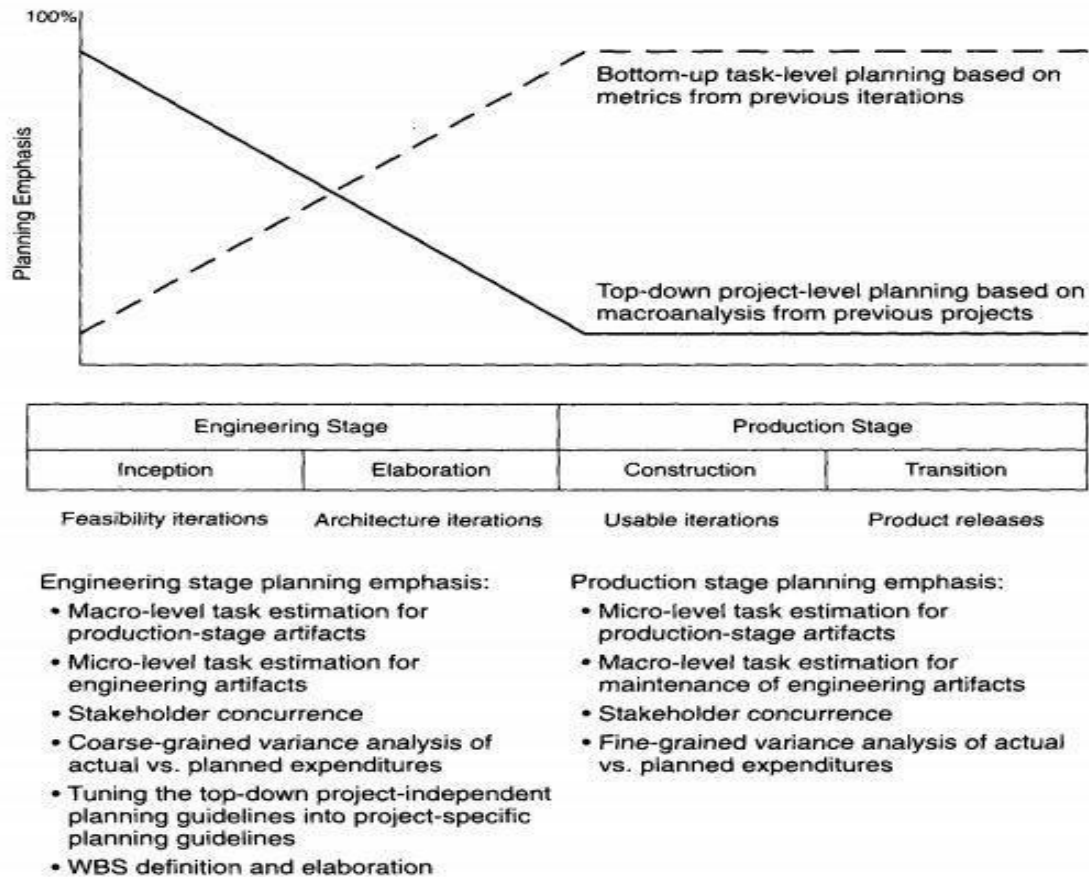


FIGURE 10-4. Planning balance throughout the life cycle

Iteration planning defines the actual sequence of intermediate results in a software project. It includes planning the content and schedule of major milestones and their intermediate iterations.

Iteration planning is an important part of the overall risk management plan. As the project progresses and the project circumstances become better understood, the build content and schedule may be adjusted.

Planning Balance Throughout the Life Cycle

In the Engineering Stage, which includes Inception and Elaboration, planning initially gives more importance to top-down project-level planning based on macro-level estimates from previous projects.

As the project progresses, more importance is given to bottom-up task-level planning based on metrics obtained from previous iterations.

In the Production Stage, which includes Construction and Transition, planning focuses more on detailed estimates based on actual project information and previous iterations.

The Engineering Stage includes estimation of tasks for production and engineering artifacts, stakeholder concurrence, analysis of actual versus planned expenditures, project-specific planning guidelines, and WBS definition and elaboration.

The Production Stage includes estimation of production and maintenance tasks, stakeholder concurrence, and detailed analysis of actual versus planned expenditures.

Iteration

An iteration is a complete synchronization across the project with a well-orchestrated assessment of the entire project baseline.

Smaller iterations, such as monthly, weekly, or daily builds, may also be performed. These mainly support the overall project-level synchronization.

Inception Iterations

During Inception, early prototyping activities integrate the fundamental components of a candidate architecture and provide an executable framework for elaborating the critical use cases.

The framework may include existing components, commercial components, and custom prototypes. It helps establish a credible business case, vision, and software development plan.

Most projects can complete this work in one iteration, although large-scale custom developments may require two iterations.

Elaboration Iterations

During Elaboration, iterations produce an architecture with a complete framework and infrastructure for execution.

The architecture should be tested using scenarios that represent the worst-case data-processing flow and control flow.

Most projects should plan for two elaboration iterations to achieve an acceptable architecture baseline. Unprecedented architectures may require additional iterations.

Construction Iterations

During Construction, most projects require at least two major iterations: an alpha release and a beta release.

The alpha release provides executable capability for all critical use cases. It represents about 70% of the total product breadth and provides the expected quality level.

The beta release typically provides about 95% of the total product breadth and achieves important quality attributes. Some features, robustness, and performance may still need improvement.

Additional construction iterations may be required to manage risks or optimize resource expenditures.

Transition Iterations

During Transition, most projects use a single iteration to move from the beta release to the final product release.

Additional small-scale iterations may be required to resolve defects, incorporate beta feedback, and improve performance.

However, because a full-scale transition to the user community has significant overhead, most projects use only one transition iteration.

Typical Number of Iterations

Most projects generally use four to nine iterations. A typical project uses six iterations.

The six-iteration profile consists of one Inception iteration, two Elaboration iterations, two Construction iterations, and one Transition iteration.

Highly precededented or very small projects may complete the work in four iterations. Very large or unprecedented projects with many stakeholders may require up to nine iterations.

The additional iterations increase management overhead but can help ensure proper risk management and stakeholder synchronization.

PRAGMATIC PLANNING

Good planning is a **dynamic and iterative process**. During iteration **N**, the project manager monitors and controls the project according to the plan prepared in iteration **N-1**. At the same time, iteration **N+1** must also be planned. Effective project management requires making **trade-offs between the current and next iteration plans** based on the objectives and results of the current and previous iterations.

In the early stages, this process may appear difficult. However, once the planning process becomes established, planning becomes easier as the project progresses and more detailed planning is required.

Importance of Planning

Apart from bad architectures and misunderstood requirements, **inadequate planning and poor management are common reasons for project failure**. The success of a project can largely be associated with three perspectives: **planning, requirements, and architecture**.

The important documents produced from these perspectives include the **software development plan, requirements specification, and architecture description document**. These documents are not usually used directly after they are produced. They mainly represent the visible part of the detailed work and decisions behind the project.

Planning Process

Although a planning document may not be very useful as an end product, the **act of planning is extremely important for project success**. Planning provides a framework for **making decisions** and ensures **buy-in from stakeholders and project performers**. It also transforms general development processes into **objective processes**.

A project plan defines how **project requirements are transformed into a product within the required constraints**. The plan must be **realistic and current**. It must also be a **team product**, understood by stakeholders, and actually used during the project.

Planning and Teamwork

Plans are **not only for managers**. When the planning process and its results are open and visible, team members develop greater **ownership** of the work they need to perform.

Poorly maintained plans can cause **frustration and reduce team commitment**. Good and open plans can **shape organizational culture and encourage teamwork**.