UNIT – III

Work Flows of the process: Software process workflows, Iteration workflows.
Checkpoints of the process: Major mile stones, Minor Milestones, Periodic status assessments.
Iterative Process Planning: Work breakdown structures, planning guidelines, cost and schedule estimating, Iteration planning process, Pragmatic planning.

Workflow of the process

SOFTWARE PROCESS WORKFLOWS

The term WORKFLOWS is used to mean a thread of cohesive and mostly sequential activities. Workflows are mapped to product artifacts There are seven top-level workflows:

1. Management workflow: controlling the process and ensuring win conditions for all stakeholders
2. Environment workflow: automating the process and evolving the maintenance environment
3. Requirements workflow: analyzing the problem space and evolving the requirements artifacts
4. Design workflow: modeling the solution and evolving the architecture and design artifacts
5. Implementation workflow: programming the components and evolving the implementation and deployment artifacts
6. Assessment workflow: assessing the trends in process and product quality
7. Deployment workflow: transitioning the end products to the user

Figure 8-1 illustrates the relative levels of effort expected across the phases in each of the top-level workflows.

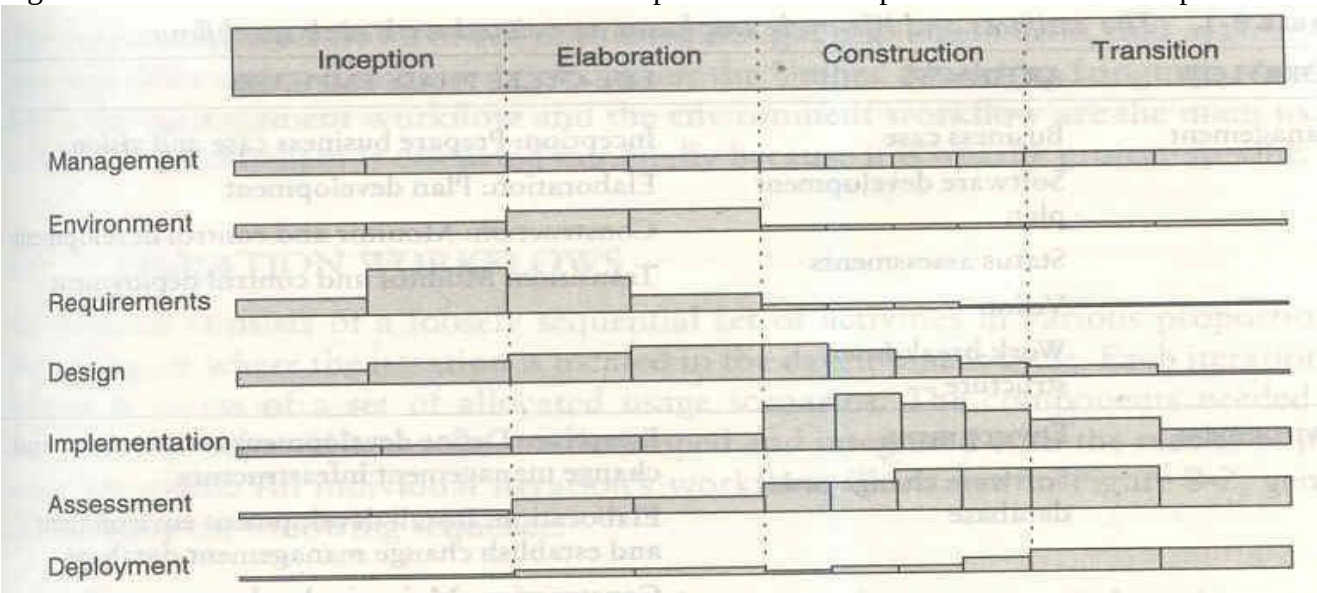


FIGURE 8-1. *Activity levels across the life-cycle phases*

Table 8-1 shows the allocation of artifacts and the emphasis of each workflow in each of the life-cycle phases of inception, elaboration, construction, and transition.

TABLE 8-1. *The artifacts and life-cycle emphases associated with each workflow*

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

ITERATION WORKFLOWS

Iteration consists of a loosely sequential set of activities in various proportions, depending on where the iteration is located in the development cycle. Each iteration is defined in terms of a set of allocated usage scenarios. An individual iteration's workflow, illustrated in Figure 8-2, generally includes the following sequence:

- Management: iteration planning to determine the content of the release and develop the detailed plan for the iteration; assignment of work packages, or tasks, to the development team
- Environment: evolving the software change order database to reflect all new baselines and changes to existing baselines for all product, test, and environment components

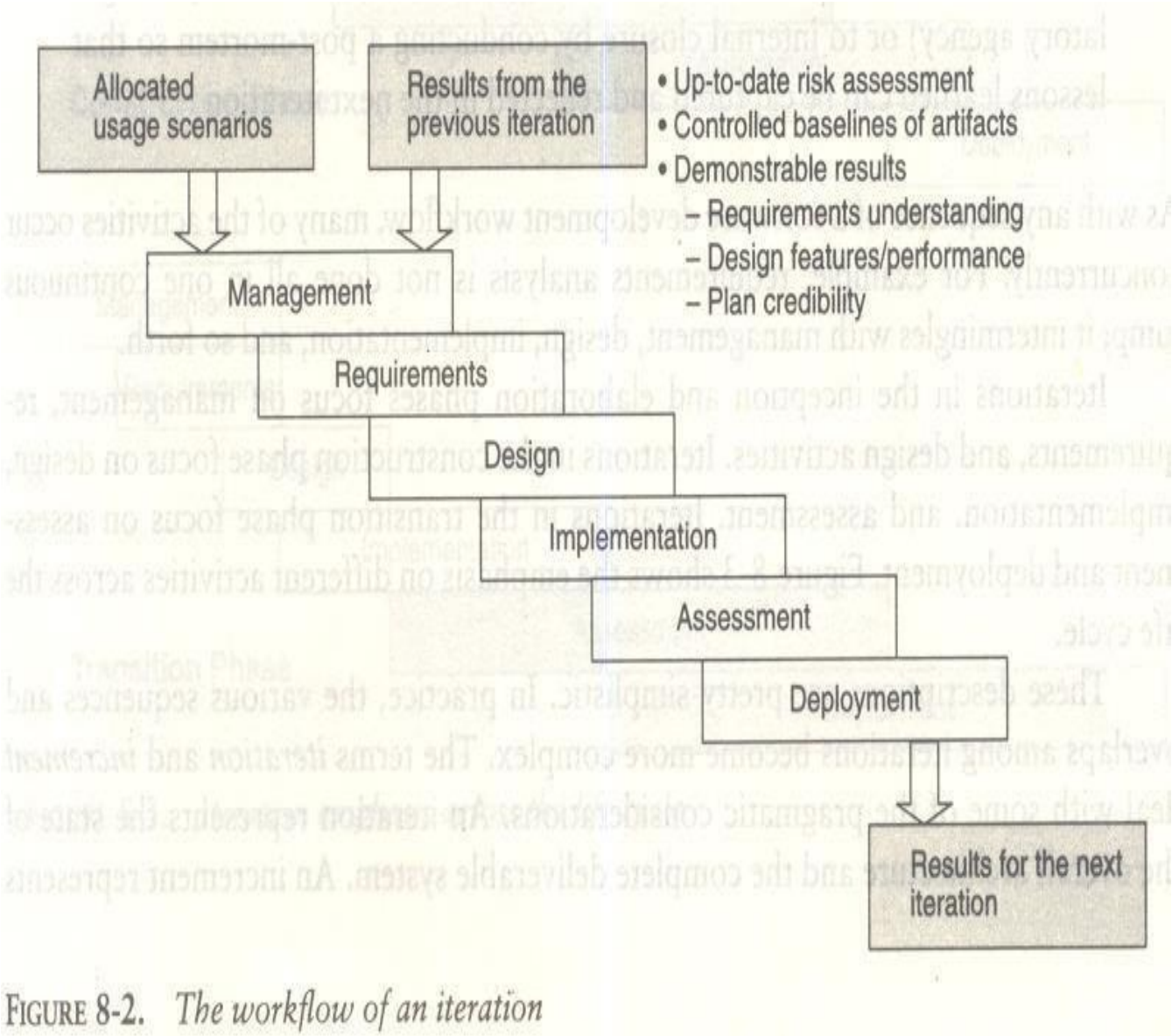


FIGURE 8-2. *The workflow of an iteration*

- Requirements: analyzing the baseline plan, the baseline architecture, and the baseline requirements set artifacts to fully elaborate the use cases to be demonstrated at the end of this iteration and their evaluation criteria; updating any requirements set artifacts to reflect changes necessitated by results of this iteration's engineering activities
- Design: evolving the baseline architecture and the baseline design set artifacts to elaborate fully the design model and test model components necessary to demonstrate against the evaluation criteria allocated to this iteration; updating design set artifacts to reflect changes necessitated by the results of this iteration's engineering activities

- Implementation: developing or acquiring any new components, and enhancing or modifying any existing components, to demonstrate the evaluation criteria allocated to this iteration; integrating and testing all new and modified components with existing baselines (previous versions)
- Assessment: evaluating the results of the iteration, including compliance with the allocated evaluation criteria and the quality of the current baselines; identifying any rework required and determining whether it should be performed before deployment of this release or allocated to the next release; assessing results to improve the basis of the subsequent iteration's plan
- Deployment: transitioning the release either to an external organization (such as a user, independent verification and validation contractor, or regulatory agency) or to internal closure by conducting a post-mortem so that lessons learned can be captured and reflected in the next iteration

Iterations in the inception and elaboration phases focus on management. Requirements, and design activities. Iterations in the construction phase focus on design, implementation, and assessment. Iterations in the transition phase focus on assessment and deployment. Figure 8-3 shows the emphasis on different activities across the life cycle. An iteration represents the state of the overall architecture and the complete deliverable system. An increment represents the current progress that will be combined with the preceding iteration to from the next iteration. Figure 8-4, an example of a simple development life cycle, illustrates the differences between iterations and increments.

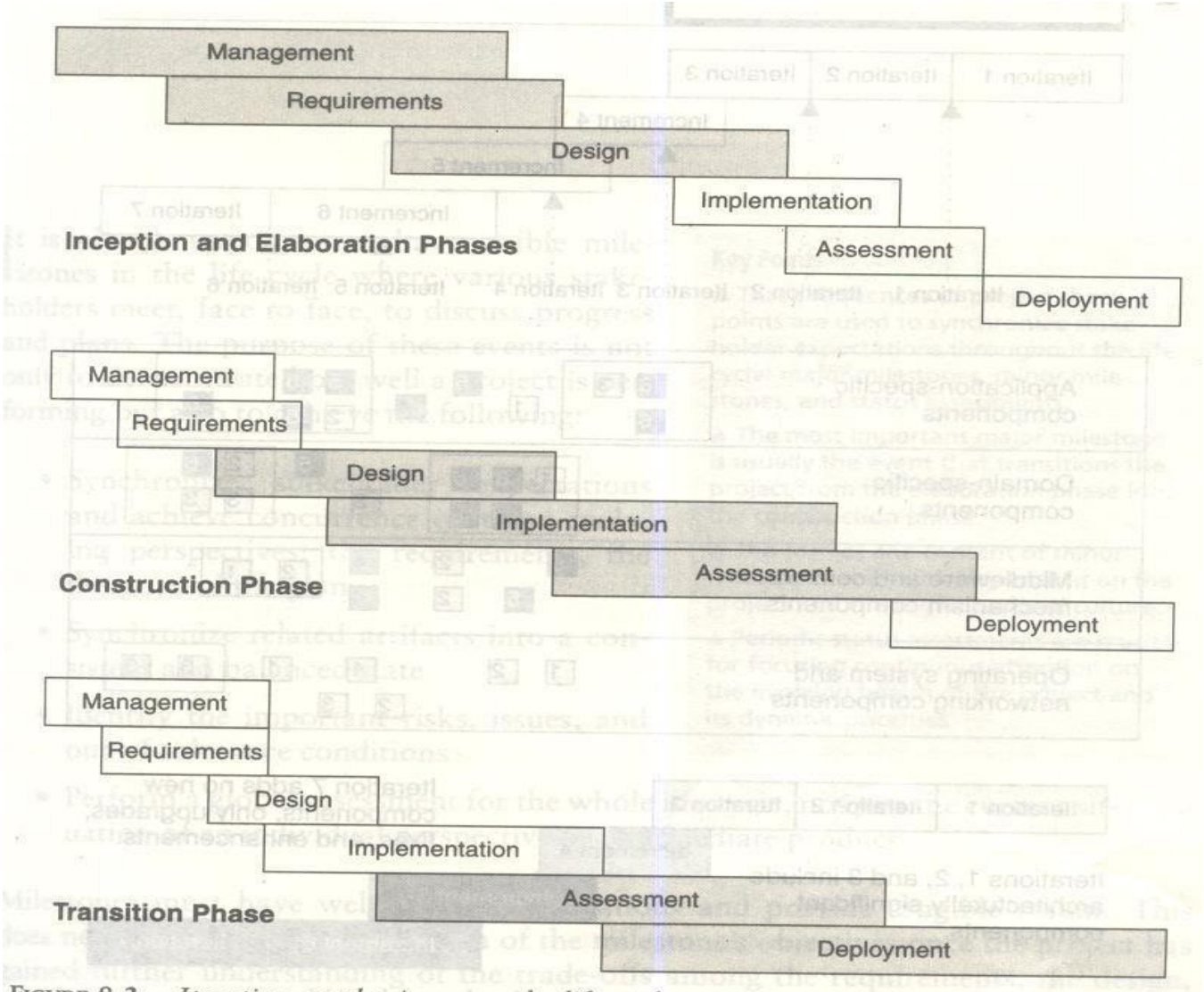


FIGURE 8-3. *Iteration emphasis across the life cycle*

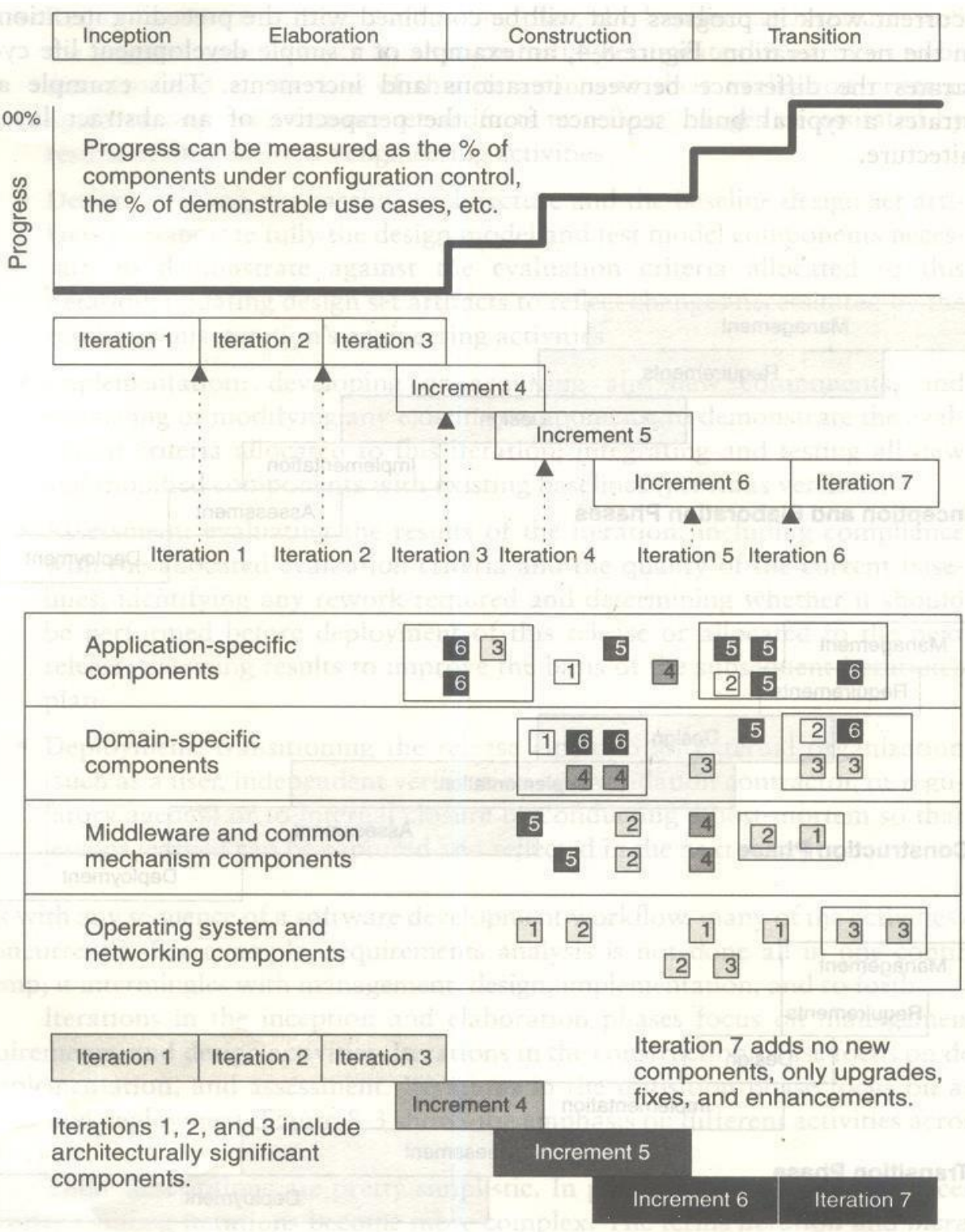


FIGURE 8-4. *A typical build sequence associated with a layered architecture*

9. Checkpoints of the process

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

1. *Major milestones.* These system wide events are held at the end of each development phase. They provide visibility to system wide issues, synchronize the management and engineering perspectives, and verify that the aims of the phase have been achieved.
2. *Minor milestones.* These iteration-focused events are conducted to review the content of an iteration in detail and to authorize continued work.
3. *Status assessments.* These periodic events provide management with frequent and regular insight into the progress being made.

Each of the four phases-inception, elaboration, construction, and transition consists of one or more iterations and concludes with a major milestone when a planned technical capability is produced in demonstrable form. An iteration represents a cycle of activities for which there is a well-defined intermediate result-a minor milestone-captured with two artifacts: a release specification (the evaluation criteria and plan) and a release description (the results). Major milestones at the end of each phase use formal, stakeholder-approved evaluation criteria and release descriptions; minor milestones use informal, development-team-controlled versions of these artifacts.
Figure 9-1 illustrates a typical sequence of project checkpoints for a relatively large project.

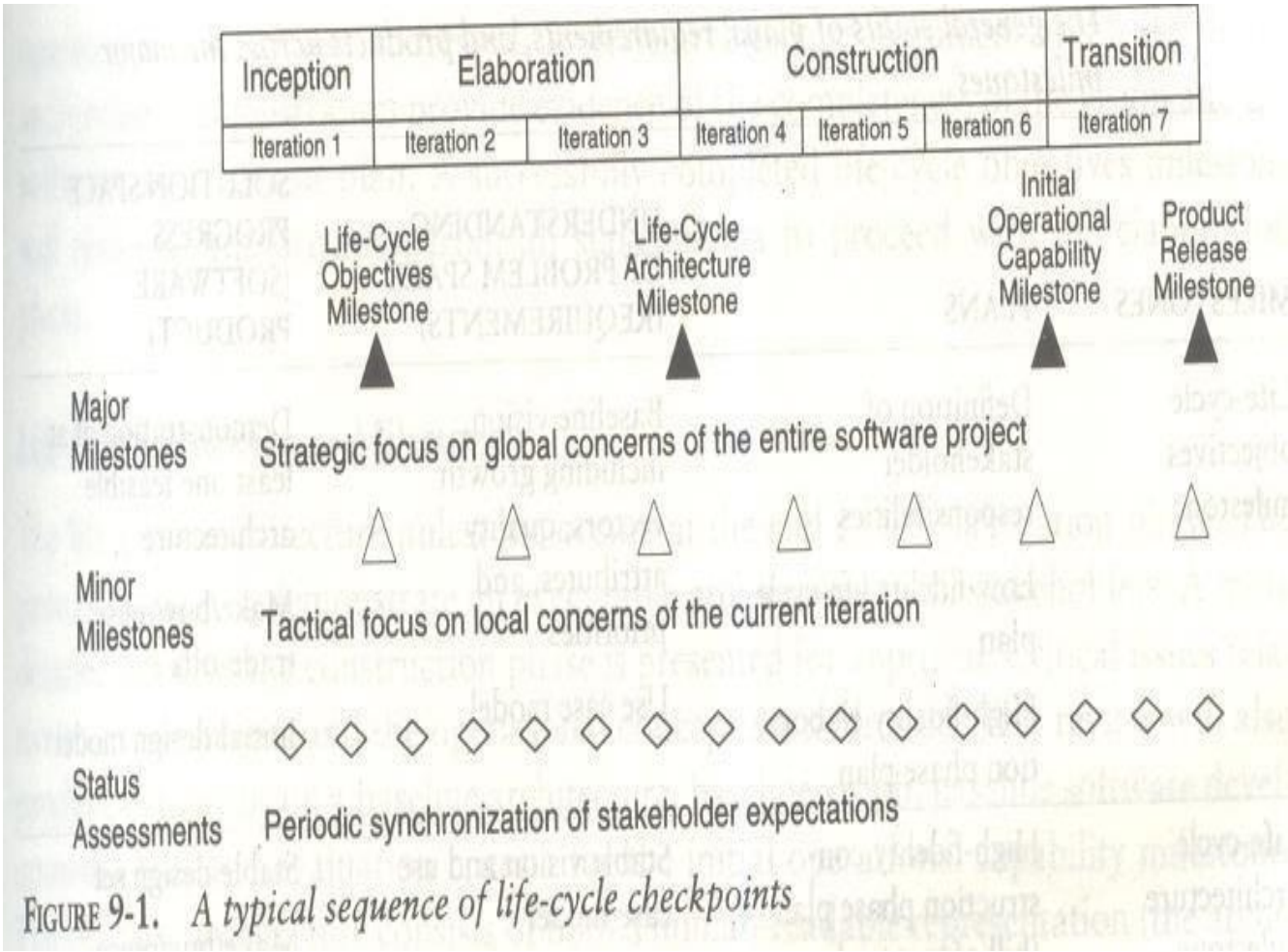


FIGURE 9-1. *A typical sequence of life-cycle checkpoints*

MAJOR MILESTONES

The four major milestones occur at the transition points between life-cycle phases. They can be used in many different process models, including the conventional waterfall model. In an iterative model, the major milestones are used to achieve concurrence among all stakeholders on the current state of the project. 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 summarizes the balance of information across the major milestones.

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

The life-cycle objectives milestone occurs at the end of the inception phase. The goal is to present to all stakeholders a recommendation on how to proceed with development, including a plan, estimated cost and schedule, and expected benefits and cost savings. A successfully completed life-cycle objectives milestone will result in authorization from all stakeholders to proceed with the elaboration phase.

Life-Cycle Architecture Milestone

The life-cycle architecture milestone occurs at the end of the elaboration phase. The primary goal is to demonstrate an executable architecture to all stakeholders. The baseline architecture consists of both a human- readable representation (the architecture document) and a configuration-controlled set of software components captured in the engineering artifacts. A successfully completed life-cycle architecture milestone will result in authorization from the stakeholders to proceed with the construction phase.

The technical data listed in Figure 9-2 should have been reviewed by the time of the lifecycle architecture milestone. Figure 9-3 provides default agendas for this milestone.

- **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

FIGURE 9-2. *Engineering artifacts available at the life-cycle architecture milestone*

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

The initial operational capability milestone occurs late in the construction phase. The goals are to assess the readiness of the software to begin the transition into customer/user sites and to authorize the start of acceptance testing. Acceptance testing can be done incrementally across multiple iterations or can be completed entirely during the transition phase is not necessarily the completion of the construction phase.

Product Release Milestone

The product release milestone occurs at the end of the transition phase. The goal is to assess the completion of the software and its transition to the support organization, if any. The results of acceptance testing are reviewed, and all open issues are addressed. Software quality metrics are reviewed to determine whether quality is sufficient for transition to the support organization.

MINOR MILESTONES

For most iterations, which have a one-month to six-month duration, only two minor milestones are needed: the iteration readiness review and the iteration assessment review.

- Iteration Readiness Review. This informal milestone is conducted at the start of each iteration to review the detailed iteration plan and the evaluation criteria that have been allocated to this iteration.
- Iteration Assessment Review. This informal milestone is conducted at the end of each iteration to assess the degree to which the iteration achieved its objectives and satisfied its evaluation criteria, to review iteration results, to review qualification test results (if part of the iteration), to determine the amount of rework to be done, and to review the impact of the iteration results on the plan for subsequent iterations.

The format and content of these minor milestones tend to be highly dependent on the project and the organizational culture. Figure 9-4 identifies the various minor milestones to be considered when a project is being planned.

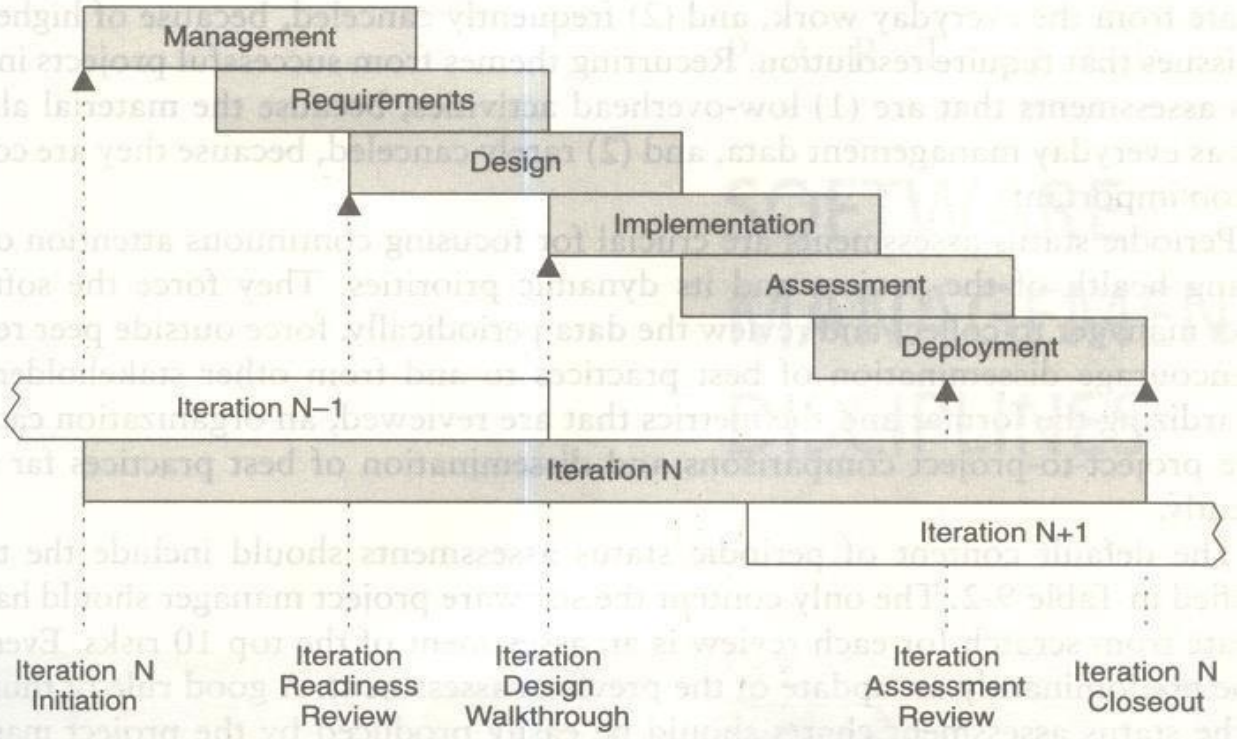


FIGURE 9-4. *Typical minor milestones in the life cycle of an iteration*

PERIODIC STATUS ASSESSMENTS

Periodic status assessments are management reviews conducted at regular intervals (monthly, quarterly) to address progress and quality indicators, ensure continuous attention to project dynamics, and maintain open communications among all stakeholders.

Periodic status assessments serve as project snapshots. While the period may vary, the recurring event forces the project history to be captured and documented. Status assessments provide the following:

- A mechanism for openly addressing, communicating, and resolving management issues, technical issues, and project risks
- Objective data derived directly from on-going activities and evolving product configurations
- A mechanism for disseminating process, progress, quality trends, practices, and experience information to and from all stakeholders in an open forum

Periodic status assessments are crucial for focusing continuous attention on the evolving health of the project and its dynamic priorities. They force the software project manager to collect and review the data periodically, force outside peer review, and encourage dissemination of best practices to and from other stakeholders.

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

TABLE 9-2. *Default content of status assessment reviews*

TOPIC	CONTENT
Personnel	Staffing plan vs. actuals Attritions, additions
Financial trends	Expenditure plan vs. actuals for the previous, current, and next major milestones Revenue forecasts
Top 10 risks	Issues and criticality 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 the next major milestone Pass/fail results for all acceptance criteria
Total product scope	Total size, growth, and acceptance criteria perturbations

Iterative process planning

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
- A 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 a 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.

Conventional work breakdown structures are prematurely structured around the product design. Figure 10-1 shows a typical conventional WBS that has been structured primarily around the subsystems of its product architecture, then further decomposed into the components of each subsystem. A WBS is the architecture for the financial plan.

Conventional work breakdown structures are prematurely decomposed, planned, and budgeted in either too little or too much detail. Large software projects tend to be over planned and small projects tend to be under planned. The basic problem with planning too much detail at the outset is that the detail does not evolve with the level of fidelity in the plan.

Conventional work breakdown structures are project-specific, and cross-project comparisons

are usually difficult or impossible. With no standard WBS structure, it is extremely difficult to compare plans, financial data, schedule data, organizational efficiencies, cost trends, productivity trends, or quality trends across multiple project.

Figure 10-1 Conventional work breakdown structure, following the product hierarchy

Management
System requirement and design
Subsystem 1
Component 11
Requirements
Design
Code
Test
Documentation
…(similar structures for other components)
Component 1N
Requirements
Design
Code
Test
Documentation
…(similar structures for other subsystems)
Subsystem M
Component M1
Requirements
Design
Code
Test
Documentation
…(similar structures for other components)
Component MN
Requirements
Design
Code
Test
Documentation
Integration and test
Test planning
Test procedure preparation
Testing
Test reports
Other support areas
Configuration control
Quality assurance
System administration

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 elements are defined for each phase of the life cycle (inception, elaboration, construction, and transition).
- Third-level 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) is shown in Figure 10-2. 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. Larger projects will have more levels and substructures.
- Organizational structure. Projects that include subcontractors or span multiple organizational entities may introduce constraints that necessitate different WBS allocations.
- Degree of custom development. Depending on the character of the project, there can be very different emphases in the requirements, design, and implementation workflows.
- Business context. Projects developing commercial products for delivery to a broad customer base may require much more elaborate substructures for the deployment element.
- Precedent experience. Very few projects start with a clean slate. Most of them are developed as new generations of a legacy system (with a mature WBS) or in the context of existing organizational standards (with preordained WBS expectations).

The WBS decomposes the character of the project and maps it to the life cycle, the budget, and the personnel. Reviewing a WBS provides insight into the important attributes, priorities, and structure of the project plan.

Another important attribute of a good WBS is that the planning fidelity inherent in each element is commensurate with the current life-cycle phase and project state. Figure 10-3 illustrates this idea. One of the primary reasons for organizing the default WBS the way I have is to allow for planning elements that range from planning packages (rough budgets that are maintained as an estimate for future elaboration rather than being decomposed into detail) through fully planned activity networks (with a well-defined budget and continuous assessment of actual versus planned expenditures).

Figure 10-2 Default work breakdown structure

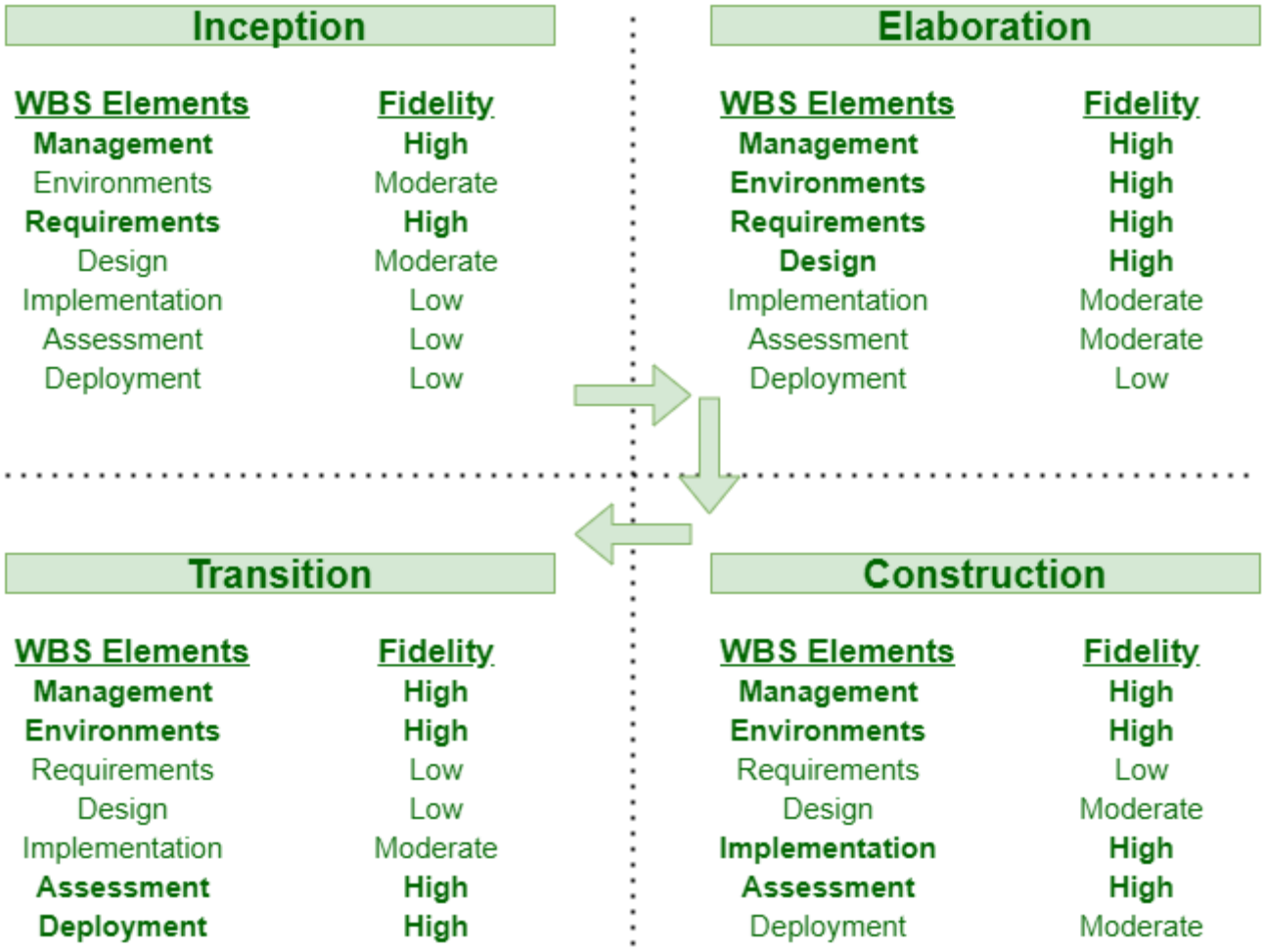
A Management
- **AA Inception phase management**
  - **AAA Business case development**
  - **AAB Elaboration phase release specifications**
  - **AAC Elaboration phase WBS specifications**
  - **AAD Software development plan**
  - **AAE Inception phase project control and status assessments**
- **AB 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**
  - **CCA 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**

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
F Assessment
FA Inception phase assessment
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 description
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



Evolution of Planning Fidelity in the WBS over the Life-Cycle

PLANNING GUIDELINES

Software projects span a broad range of application domains. It is valuable but risky to make specific planning recommendations independent of project context. Project-independent planning advice is also risky. There is the risk that the guidelines may pe adopted blindly without being adapted to specific project circumstances. Two simple planning guidelines should be considered when a project plan is being initiated or assessed. The first guideline, detailed in Table 10-1, prescribes a default allocation of costs among the first-level WBS elements. The second guideline, detailed in Table 10-2, prescribes the allocation of effort and schedule across the lifecycle phases.

10-1 Web budgeting defaults

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

Table 10-2 Default distributions of effort and schedule by phase

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

THE COST AND SCHEDULE ESTIMATING PROCESS

Project plans need to be derived from two perspectives. The first is a forward-looking, top-down approach. It starts with an understanding of the general requirements and constraints, derives a macro-level budget and schedule, then decomposes these elements into lower level budgets and intermediate milestones. From this perspective, the following planning sequence would occur:

1. The software project manager (and others) develops a characterization of the overall size, process, environment, people, and quality required for the project.
2. A macro-level estimate of the total effort and schedule is developed using a software cost estimation model.
3. The software project manager partitions the estimate for the effort into a top-level WBS using guidelines such as those in Table 10-1.
4. At this point, subproject managers are given the responsibility for decomposing each of the WBS elements into lower levels using their top-level allocation, staffing profile, and major milestone dates as constraints.

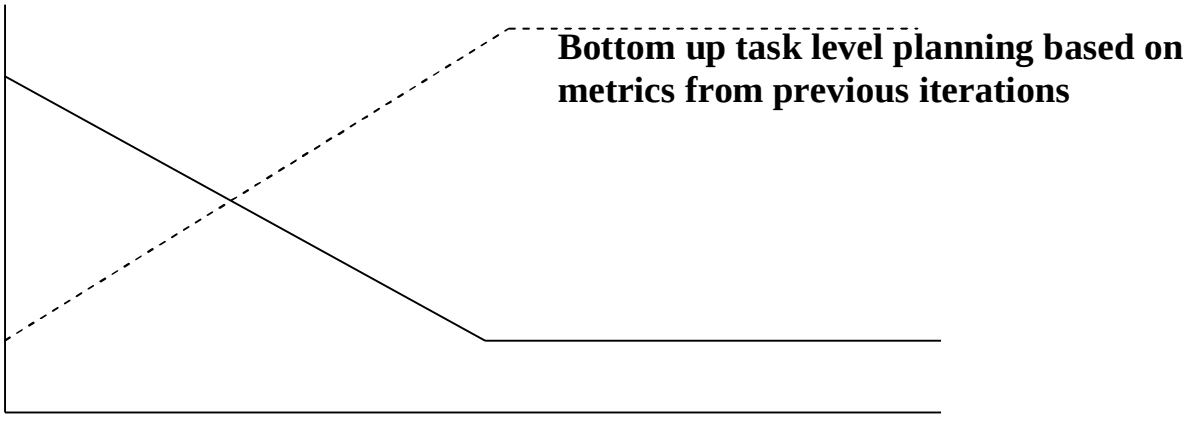
The second perspective is a backward-looking, bottom-up approach. We start with the end in mind, analyze the micro-level budgets and schedules, then sum all these elements into the higher level budgets and intermediate milestones. This approach tends to define and populate the WBS from the lowest levels upward. From this perspective, the following planning sequence would occur:

1. The lowest level WBS elements are elaborated into detailed tasks
2. Estimates are combined and integrated into higher level budgets and milestones.
3. Comparisons are made with the top-down budgets and schedule milestones.

Milestone scheduling or budget allocation through top-down estimating tends to exaggerate the project management biases and usually results in an overly optimistic plan. Bottom-up estimates usually exaggerate the performer biases and result in an overly pessimistic plan.

These two planning approaches should be used together, in balance, throughout the life cycle of the project. During the engineering stage, the top-down perspective will dominate because there is usually not enough depth of understanding nor stability in the detailed task sequences to perform credible bottom-up planning. During the production stage, there should be enough precedent experience and planning fidelity that the bottom-up planning perspective will dominate. Top-down approach should be well tuned to the project-specific parameters, so it should be used more as a global assessment technique. Figure 10-4 illustrates this life-cycle planning balance.

Figure 10-4 Planning balance throughout the life cycle



Top down project level planning based on microanalysis from previous projects

Engineering Stage		Production Stage	
Inception	**Elaboration**	**Construction**	**Transition**
Feasibility iteration	**Architecture iteration**	**Usable iteration**	**Product Releases**

Engineering stage planning emphasis	**Production stage planning emphasis**
Macro level task estimation for production stage artifacts	**Micro level task estimation for production stage artifacts**
Micro level task estimation for engineering artifacts	**Macro level task estimation for maintenance of engineering artifacts**
Stakeholder concurrence	**Stakeholder concurrence**
Coarse grained variance analysis of actual vs planned expenditures	**Fine grained variance analysis of actual vs planned expenditures**
Tuning the top down project independent planning guidelines into project specific planning guidelines	
WBS definition and elaboration	

THE ITERATION PLANNING PROCESS

Planning is concerned with defining the actual sequence of intermediate results. An evolutionary build plan is important because there are always adjustments in build content and schedule as early conjecture evolves into well-understood project circumstances. *Iteration* is used to mean a complete synchronization across the project, with a well-orchestrated global assessment of the entire project baseline.

- Inception iterations. The early prototyping activities integrate the foundation components of a candidate architecture and provide an executable framework for elaborating the critical use cases of the system. This framework includes existing components, commercial components, and custom prototypes sufficient to demonstrate a candidate architecture and sufficient requirements understanding to establish a credible business case, vision, and software development plan.
- Elaboration iterations. These iterations result in architecture, including a complete framework and infrastructure for execution. Upon completion of the architecture iteration, a few critical use cases should be demonstrable: (1) initializing the architecture, (2) injecting a scenario to drive the worst-case data processing flow through the system (for example, the peak transaction throughput or peak load scenario), and (3) injecting a scenario to drive the worst-case control flow through the system (for example, orchestrating the fault-tolerance use cases).
- Construction iterations. Most projects require at least two major construction iterations: an alpha release and a beta release.
- Transition iterations. Most projects use a single iteration to transition a beta release into the final product.

The general guideline is that most projects will use between four and nine iterations. The

typical project would have the following six-iteration profile:

- One iteration in inception: an architecture prototype
- Two iterations in elaboration: architecture prototype and architecture baseline
- Two iterations in construction: alpha and beta releases
- One iteration in transition: product release

A very large or unprecedented project with many stakeholders may require additional inception iteration and two additional iterations in construction, for a total of nine iterations.

PRAGMATIC PLANNING

Even though good planning is more dynamic in an iterative process, doing it accurately is far easier. While executing iteration N of any phase, the software project manager must be monitoring and controlling against a plan that was initiated in iteration N - 1 and must be planning iteration N + 1. The art of good project· management is to make trade-offs in the current iteration plan and the next iteration plan based on objective results in the current iteration and previous iterations. Aside from bad architectures and misunderstood requirements, inadequate planning (and subsequent bad management) is one of the most common reasons for project failures. Conversely, the success of every successful project can be attributed in part to good planning.

A project's plan is a definition of how the project requirements will be transformed into' a product within the business constraints. It must be realistic, it must be current, it must be a team product, it must be understood by the stakeholders, and it must be used. Plans are not just for managers. The more open and visible the planning process and results, the more ownership there is among the team members who need to execute it. Bad, closely held plans cause attrition. Good, open plans can shape cultures and encourage teamwork.

Unit – Important Questions

1.	**Define Model-Based software architecture?**
2.	**Explain various process workflows?**
3.	**Define typical sequence of life cycle checkpoints?**
4.	**Explain general status of plans, requirements and product across the major milestones.**
5.	**Explain conventional and Evolutionary work break down structures?**
6.	**Explain briefly planning balance throughout the life cycle?**