UNIT-V
PART - A

Iterative Process Planning : Work Breakdown Structures, Planning Guidelines, Cost and Schedule Estimating, Iteration Planning Process, Pragmatic Planning.

Overview

- Work Breakdown Structures
  - Conventional WBS Issues
  - Evolutionary Work Breakdown Structures

Work Breakdown Structures

- 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.

Work Breakdown Structures

- 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

Work Breakdown Structures

- 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 WBS Issues

Conventional work breakdown structures are prematurely structured around the product design.

- Below figure 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.

- Management
- System requirements 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

Conventional WBS Issues

- Once this structure is ingrained in the WBS and then allocated to responsible managers with budgets, schedules, and expected deliverables, a concrete planning foundation has been set that is difficult and expensive to change.
- A WBS is the architecture for the financial plan.
- Just as software architectures need to encapsulate components that are likely to change, so must planning architectures.

Conventional WBS Issues

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 WBS Issues

Conventional work breakdown structures are project-specific, and cross-project comparisons are usually difficult or impossible

- Most organizations allow individual projects to define their own project-specific structure tailored to the projects manager's style, the customer's demands, or other project-specific preferences.
- 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 projects.
- Each project organizes the work differently and uses different units of measure.

Conventional WBS Issues

- Some of the following simple questions, which are critical to any organizational process improvement program, cannot be answered by most project teams that use conventional work breakdown structure.
  - What is the ratio of productive activities (requirements, design, implementation, assessment, deployment) to overhead activities (management, environment)?
  - What is the percentage of effort expended in rework activities?
  - What is the percentage of cost expended in software capital equipment (the environment expenditures)?
  - What is the ratio of productive testing versus (unproductive) integration?
  - What is the cost of release N (as a basis for planning release N+1)?

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).
  - These elements are usually allocated to a single team and constitute the anatomy of a project for the purposes of planning and comparison with other projects.

EVOLUTIONARY WORK BREAKDOWN STRUCTURES

- Second-level elements are defined for each phase of the life cycle (inception, elaboration, construction, and transition).
  - These elements allow the fidelity of the plan to evolve more naturally with the level of understanding of the requirements and architecture, and the risks therein.
- Third-level elements are defined for the focus of activities that produce the artifacts of each phase.
  - These elements may be the lowest level in the hierarchy that collects the cost of a discrete artifact for a given phase, or they may be decomposed further into several lower level activities that, taken together, produce a single artifact.

EVOLUTIONARY WORK BREAKDOWN STRUCTURES

- A default WBS consistent with the process framework (phases, workflows, and artifacts) is shown in below figure.
- 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.

- 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

- 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

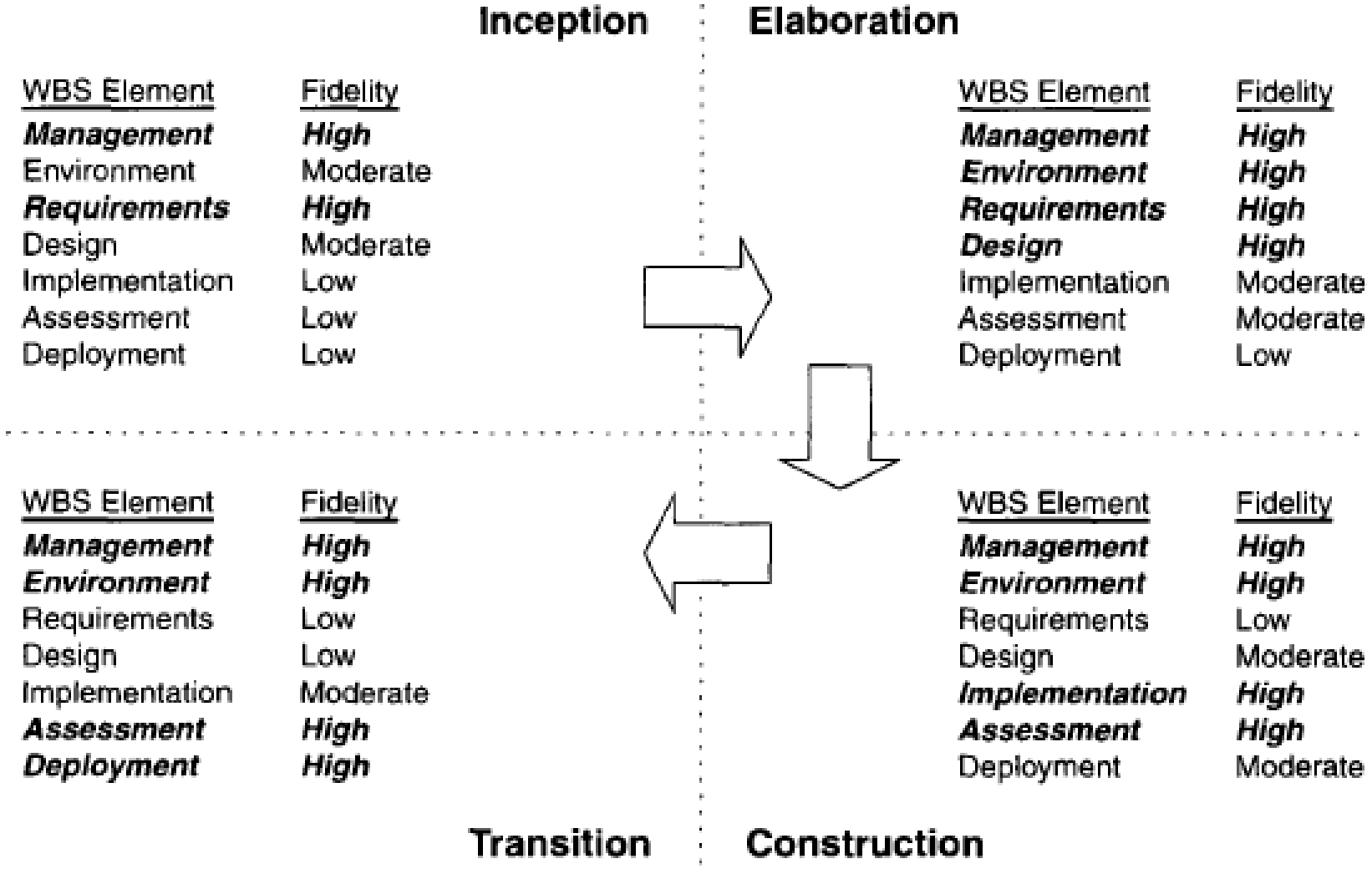
EVOLUTIONARY WORK BREAKDOWN STRUCTURES

- 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).

EVOLUTIONARY WORK BREAKDOWN STRUCTURES

- 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. Below figure illustrates this idea.
- One of the primary reasons for organizing the default WBS the way 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).

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.
- It is valuable because most people in management positions are looking for a starting point, a skeleton they can flesh out with project-specific details.

Planning Guidelines

- Project-independent planning advice is also risky.
- There is the risk that the guidelines may be adopted blindly without being adapted to specific project circumstances.
- Blind adherence to someone else's project-independent planning advice is a sure sign of an incompetent management team.

Planning Guidelines

- There is also the risk of misinterpretation.
- The variability of project parameters, project business contexts, organizational cultures, and project processes makes it extremely easy to make mistakes that have significant potential impact.
- **Two simple planning guidelines should be considered when a project plan is being initiated or assessed.**
- **The first guideline, detailed in first table, prescribes a default allocation of costs among the first-level WBS elements.**
- **The second guideline, detailed in second table, prescribes the allocation of effort and schedule across the life-cycle phases.**

WBS Budgeting Defaults

FIRST-LEVEL WBS ELEMENT	DEFAULT BUDGET
Management	10%
Environment	10%
Requirements	10%
Design	15%
Implementation	25%
Assessment	25%
Deployment	5%
Total	100%

Default distribution of effort and schedule by phase

DOMAIN	INCEPTION	ELABORATION	CONSTRUCTION	TRANSITION
Effort	5%	20%	65%	10%
Schedule	10%	30%	50%	10%

Planning Guidelines

- The first table provides default allocations for budgeted costs of each first-level WBS element.
- While these values are certain to vary across projects, this allocation provides a good benchmark for assessing the plan by understanding the rationale for deviations from these guidelines.
- An important point here is that this is cost allocation, not effort allocation.
- To avoid misinterpretation, two explanations are necessary.

1. The cost of different labor categories is inherent in these numbers.
2. The cost of hardware and software assets that support the process automation and development teams is also included in the environment element.

Planning Guidelines

- The second table provides guidelines for allocating effort and schedule across the life-cycle phases.
- Although these values can also vary widely, depending on the specific constraints of an application, they provide an average expectation across a spectrum of application domains.
- Achieving consistency using these specific values is not as important as understanding why your project may be different.

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.

The Cost and Schedule Estimating Process

- 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 first table. The project manager also partitions the schedule into major milestone dates and partitions the effort into a staffing profile using guidelines such as those in second table. Now there is a project-level plan. These sorts of estimates tend to ignore many detailed project-specific parameters.
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 Cost and Schedule Estimating Process

- **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:

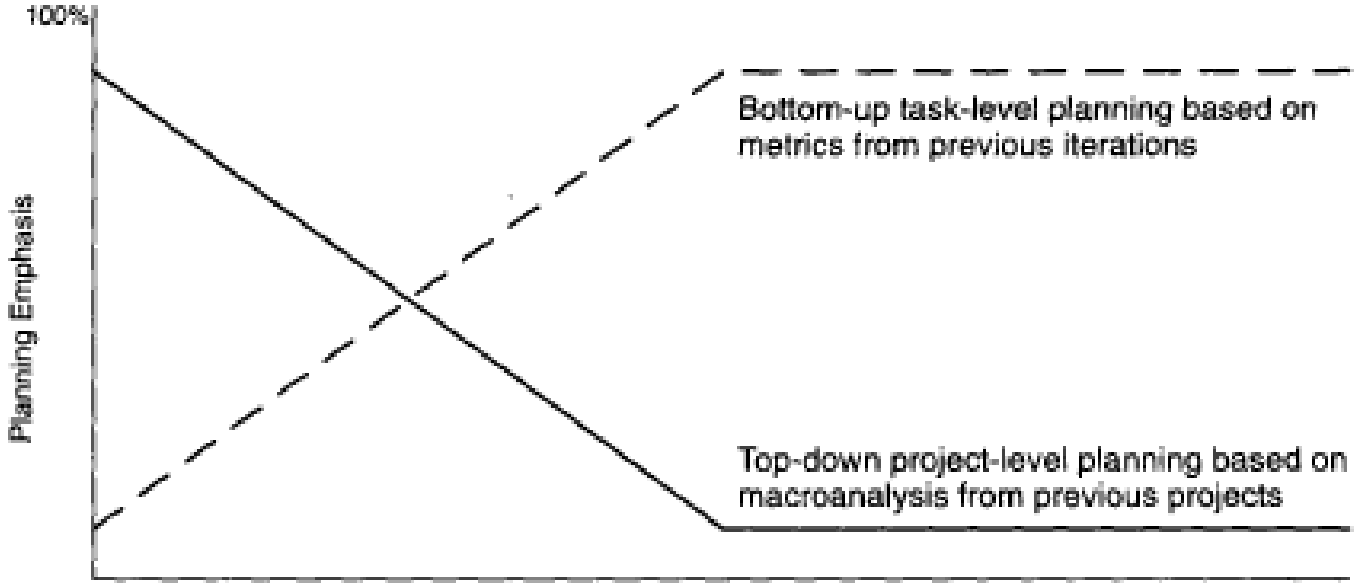
1. The lowest level WBS elements are elaborated into detailed tasks, for which budgets and schedules are estimated by the responsible WBS element manager.
2. Estimates are combined and integrated into higher level budgets and milestones.
3. Comparisons are made with the top-down budgets and schedule milestones.

The Cost and Schedule Estimating Process

- 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.
- Iteration is necessary, using the results of one approach to validate and refine the results of the other approach, thereby evolving the plan through multiple versions.
- This process instills ownership of the plan in all levels of management.

The Cost and Schedule Estimating Process

- 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.
- By then, the top-down approach should be well tuned to the project-specific parameters, so it should be used more as a global assessment technique.
- Below figure illustrates this life-cycle planning balance.



Engineering Stage		Production Stage	
Inception	Elaboration	Construction	Transition
Feasibility iterations	Architecture iterations	Usable iterations	Product releases

Engineering stage planning emphasis:

- Macro-level task estimation for production-stage artifacts
- Micro-level task estimation for engineering artifacts
- Stakeholder concurrence
- Coarse-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

Production stage planning emphasis:

- Micro-level task estimation for production-stage artifacts
- Macro-level task estimation for maintenance of engineering artifacts
- Stakeholder concurrence
- Fine-grained variance analysis of actual vs. planned expenditures

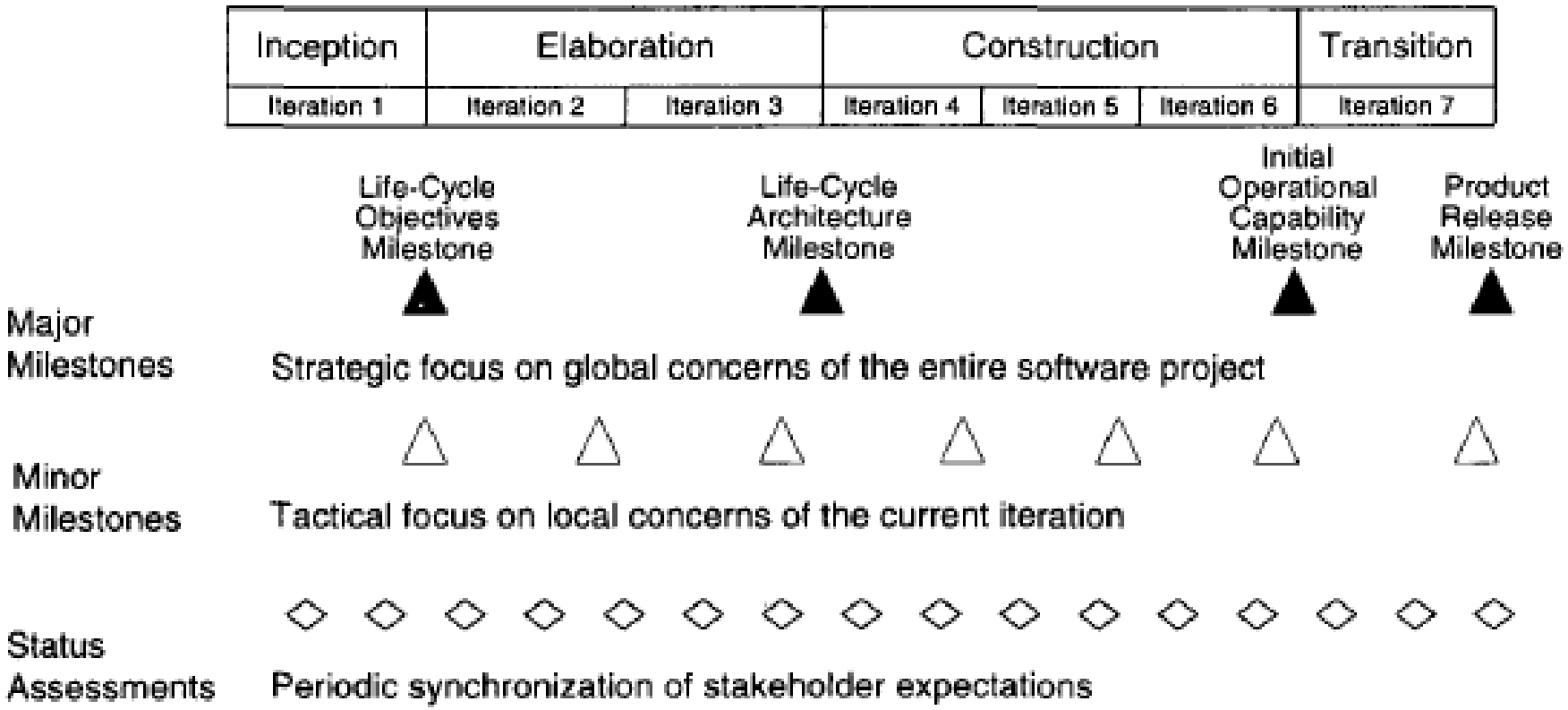
THE ITERATION PLANNING PROCESS

- Planning is concerned with defining the actual sequence of intermediate results.
- Planning the content and schedule of the major milestones and their intermediate iterations is probably the most tangible form of the overall risk management plan.
- 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.

THE ITERATION PLANNING PROCESS

- Iteration is used to mean a complete synchronization across the project, with a well-orchestrated global assessment of the entire project baseline.
- Other micro-iterations, such as monthly, weekly, or daily builds, are performed en route to these project-level synchronization points.

Checkpoints of the process



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)
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.
- **An alpha release** would include executable capability for all the critical use cases. It usually represents only about 70% of the total product breadth and performs at quality levels (performance and reliability) below those expected in the final product.
- **A beta release** typically provides 95% of the total product capability breadth and achieves some of the important quality attributes.

Transition Iterations

- Most projects use a single iteration to transition a beta release into the final product.
- Again, numerous informal, small-scale iterations may be necessary to resolve all the defects, incorporate beta feed-back, and incorporate performance improvements.
- However, because of the overhead associated with a full-scale transition to the user community, most projects learn to live with a single iteration between a beta release and the final product release.

Transition Iterations

- 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 an 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.

PRAGMATIC PLANNING

- Aside from bad architectures and misunderstood requirements, inadequate planning (and subsequent bad management) is one of the most common reasons for project failures.
- The success of every successful project can be attributed in part to good planning.

PRAGMATIC PLANNING

- While a planning document is not very useful as an end item, the act of planning is extremely important to project success.
- It provides a framework and forcing functions for making decisions, ensures buy-in on the part of stakeholders and performers and transforms subjective, generic process frameworks into objective processes.
- 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.

PRAGMATIC PLANNING

- 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.