Chapter 4
4.1 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.**

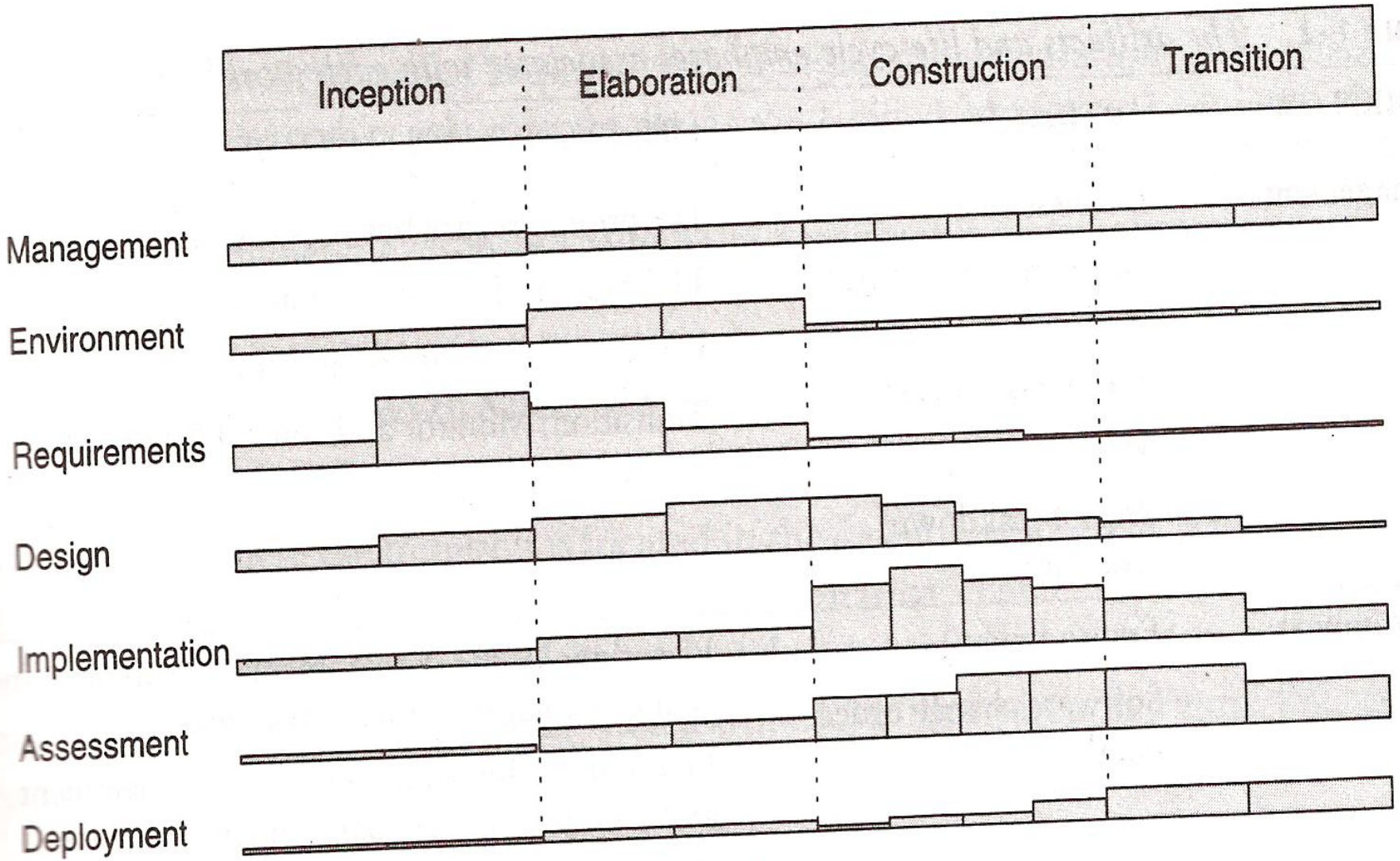


FIGURE 8-1. *Activity levels across the life-cycle phases*

4.2 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
  - 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 detem1ining 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

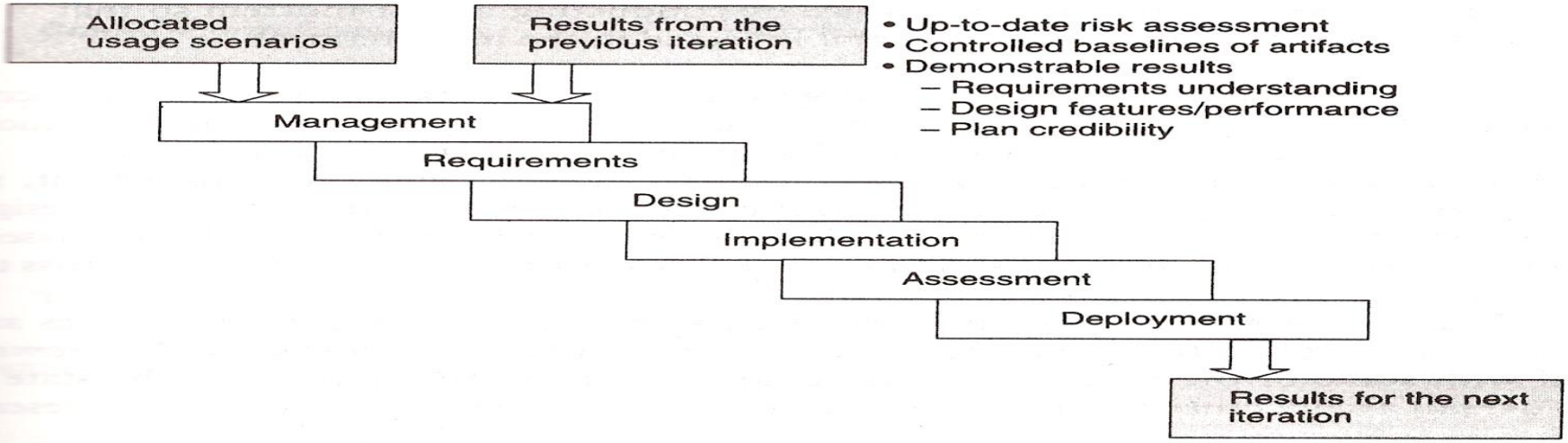


FIGURE 8-2. *The workflow of an iteration*

- 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.

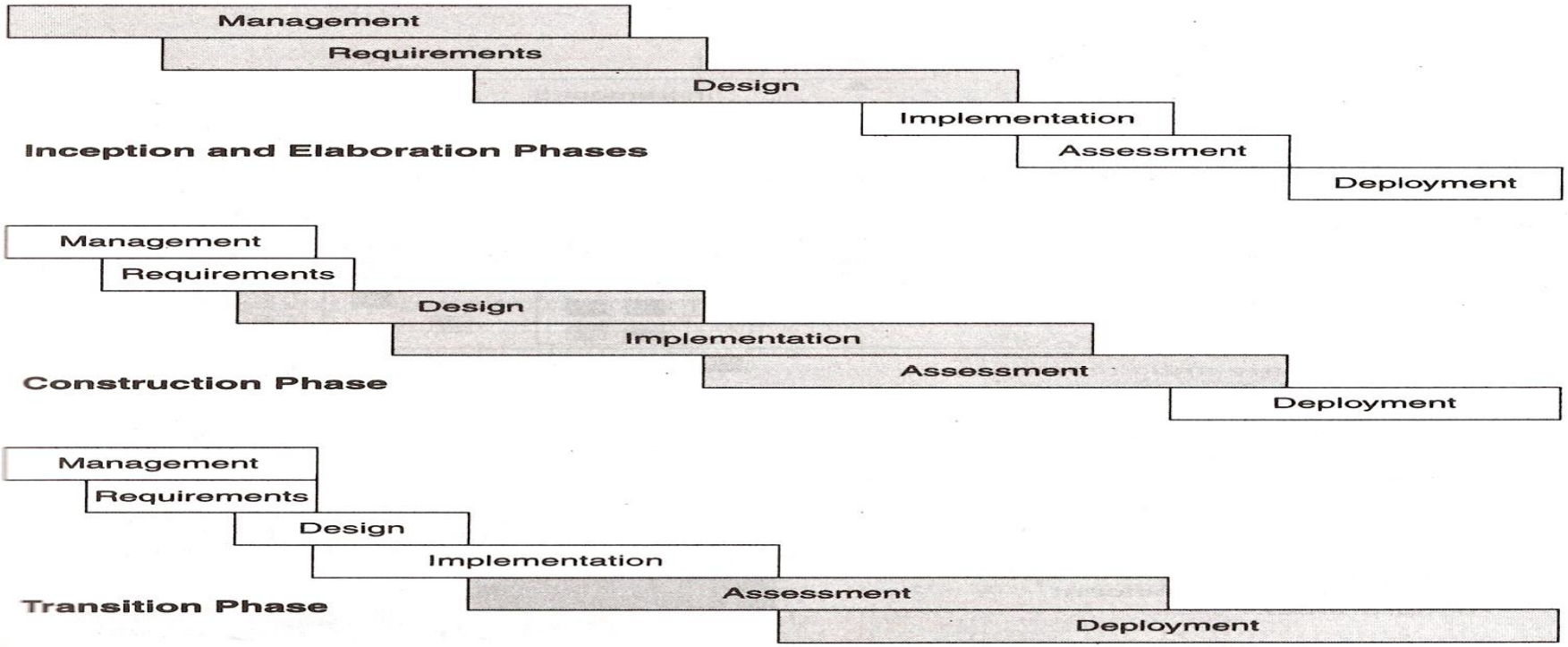


FIGURE 8-3. *Iteration emphasis across the life cycle*

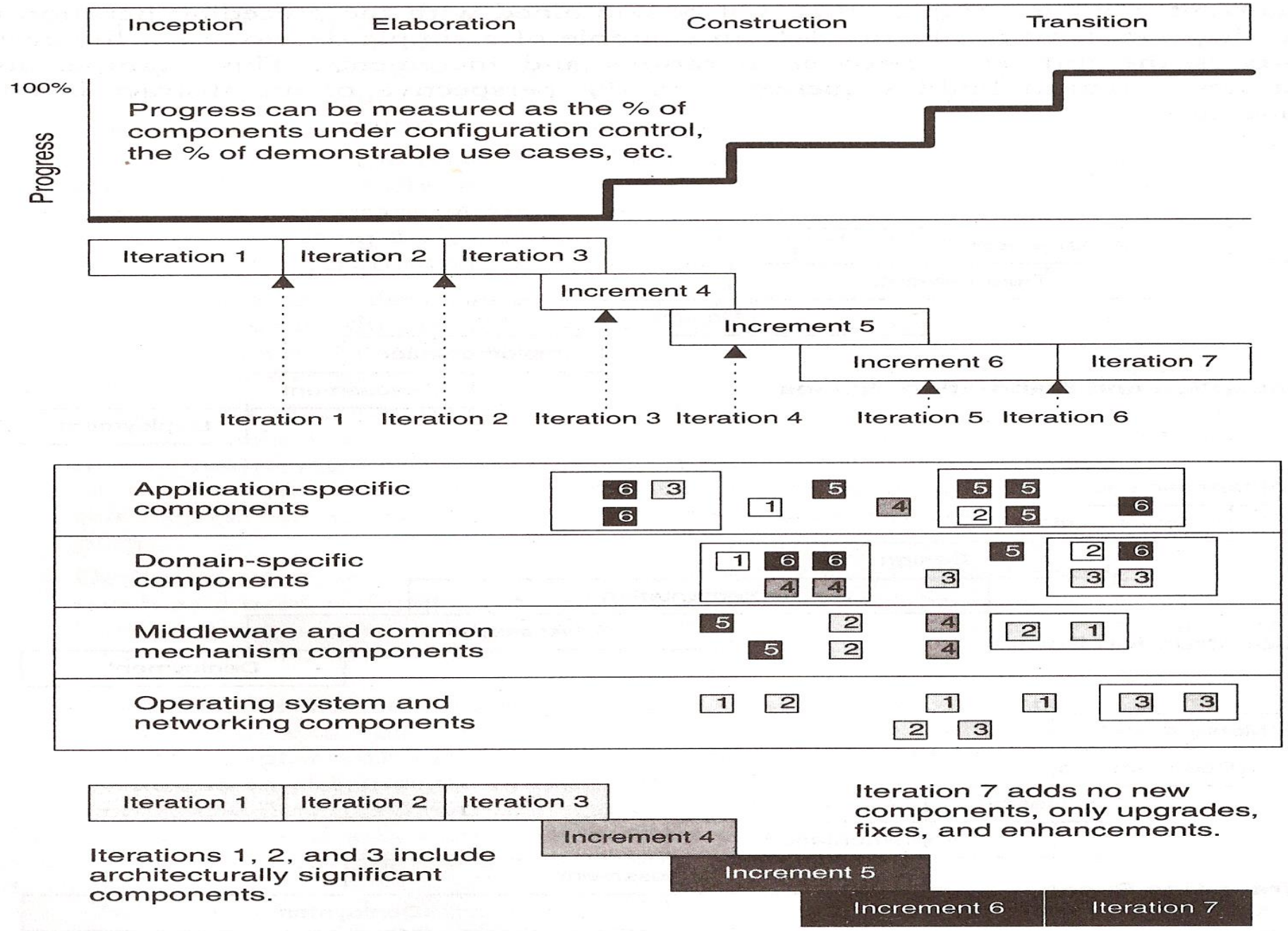


FIGURE 8-4. *A typical build sequence associated with a layered architecture*

MODEL QUESTIONS

1. Define the terms ‘model’ and ‘view’. What are the three different aspects of software architecture form management’s perspective?
2. Explain the significance of software architecture in modern software development process.
3. What does each of the views (design, process, component, deployment) address in the software architecture? Explain with an example.
4. What are the seven workflows in the life cycle?
5. What levels of activity takes place in these workflows during each of the four phases (inception, elaboration, construction and transition).
6. Define iteration. Discuss the sequence of activities in an iteration workflow.
7. Bring out the differences between iterations and increments along with suitable diagrams.

4.3. CHECKPOINTS OF THE PROCESS AND ITERATIVE PROCESS PLANNING

4.3.1 INTRODUCTION

- Three types of joint management reviews are conducted throughout the process:
  - Major Milestones: these system wide events are held ant 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.
  - Minor Milestones: theses iteration-focused events are conducted to review the content of an iteration in detail and to authorize continued work.
  - Status Assessments: These periodic events provide management with frequent and regular insight into the progress being made.
- Each of the four phases-inceptions, 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. 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.

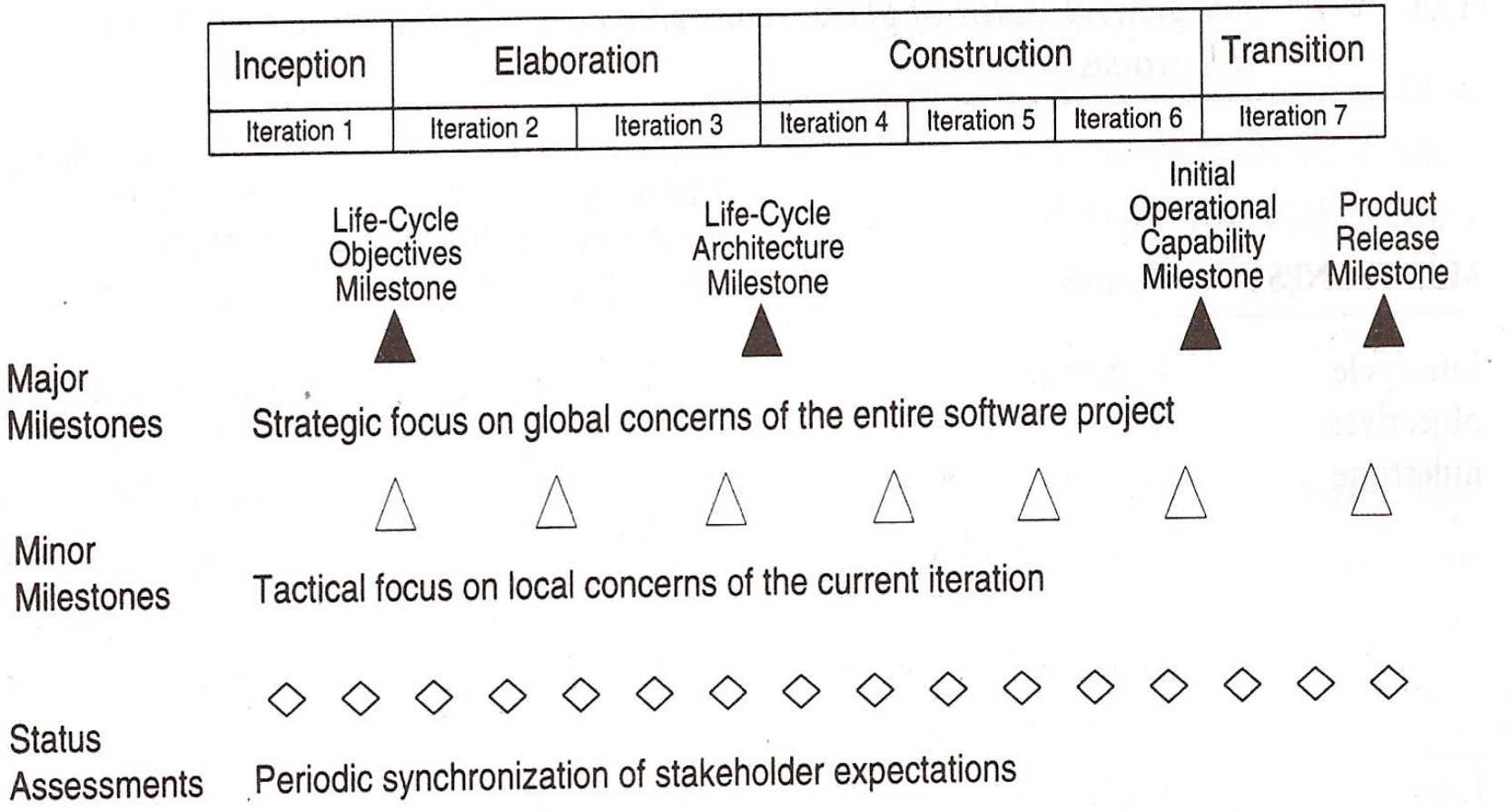


FIGURE 9-1. *A typical sequence of life-cycle checkpoints*

4.4 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 eth 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 sale and marketing teams.

- The following Table summarizes the balance of information across them major milestones.

MILESTONES	PLANS	UNDERSTANDING OF PROBLEM SPACE (REQUIREMENTS)	SOLUTION SPACE PROGRESS (SOFTWARE PRODUCT)
Life-cycle objectives milestone	1. Definition of stakeholder responsibilities 2. Low-fidelity life-cycle plan 3. High-fidelity elaboration phase plan	1. Baseline vision, including growth vectors, quality attributes and priorities 2. Use case model	1. Demonstration of at least one feasible architecture 2. Make./buy/reuse trade-offs 3. Initial design model
Life-cycle architecture milestone	1. High-fidelity construction phase plan (bill of materials, labor allocation) 2. Low-fidelity transition phase plan.	1. Stable visions and use case model 2. Evaluation criteria for construction release, initial operational capability 3. Draft user manual	1. Stable design set 2. Make/buy/reuse decisions 3. Critical component prototypes
Initial operational capability milestone	1. High-fidelity transition phase plan	1. Acceptance criteria for product release 2. Releasable user manual	1. Stable implementation set 2. Critical features and core capabilities 3. Objective insight into product qualities
Product release milestone	1. Next-generation product plan	Final user manual	1. Stable deployment set 2. Full features 3. Compliant quality

- **Life-Cycle Objectives Milestone**

- The life-cycle objectives milestone occurs at tile 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. At 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

I. Requirements assessment

A. Project vision and use cases

B. Primary scenarios and evaluation criteria

II. Architecture assessment

A. Progress

1. Baseline architecture metrics(process to date and baseline for measure future architectural stability, scarp and rework)
2. Development metrics baseline estimate (for assessing future progress)
3. Test metrics baseline estimate (for assessing future progress of the test team).

A. Quality

4. Architectural features (demonstration capability summary Vs. evaluation criteria)
5. Performance (demonstration capability summary Vs. evaluation criteria
6. Exposed architectural risks and resolution plans
7. Affordability and make/buy/reuse trade-offs

I. 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.***

4.5 MINOR MILESTONES

- Minor milestones are sometimes called as inch-pebbs.
- Minor milestones mainly focus on local concerns of current iteration.
- These iterative focused events are used to review iterative content in a detailed manner and authorize the continued work.
- Minor Milestone in the life cycle of Iteration: The number of iteration specific milestones is dependent on the iteration length and the content. A one month to six month iterative period requires only two minor milestones.
    - Iteration Readiness review: This informal milestone is conducted at the start of each iteration to review the detailed iteration plan and 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, 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.

4.6 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.

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

4.7 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.
- 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 first-level decomposition by subsystem. Subsystems are then decomposed into their components, one of which is typically the software.

4.7.1 CONVENTIONAL WBS ISSUES

- Conventional work breakdown structures frequently suffer from three fundamental flaws.
  - They are prematurely structured around the product design.
  - They are prematurely decomposed, planned, and budgeted in either too much or too little detail.
  - 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 subsystems. 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 projects.

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

FIGURE 10-1: ***Conventional work breakdown structure, following the product hierarchy***

4.7.2 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 decompose into detail) through fully planned activity networks (with a well-defined budget and continuous assessment of actual versus planned expenditures).

- 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 specifications
  - 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.

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

GBElaboration phase deployment planning

GCConstruction phase deployment

GCA User manual baselining

GD Transition phase deployment

GDA Product transition to user

FIGURE 10-2: ***Default work breakdown structure***

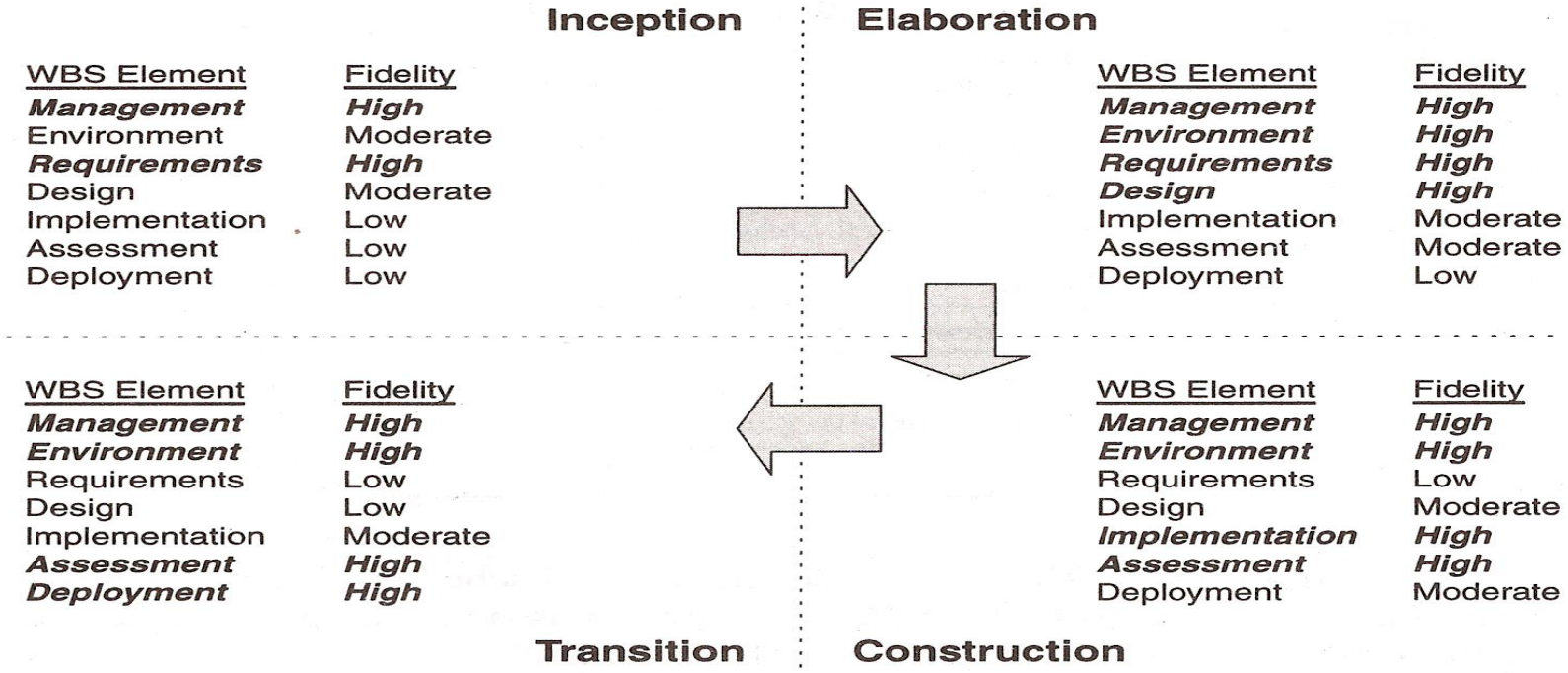


FIGURE 10-3. *Evolution of planning fidelity in the WBS over the life cycle*

4.8 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 be adopted blindly without being adapted to specific project circumstance. 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 10-25, prescribes allocation of effort and schedule across the lifecycle phases.

Table 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

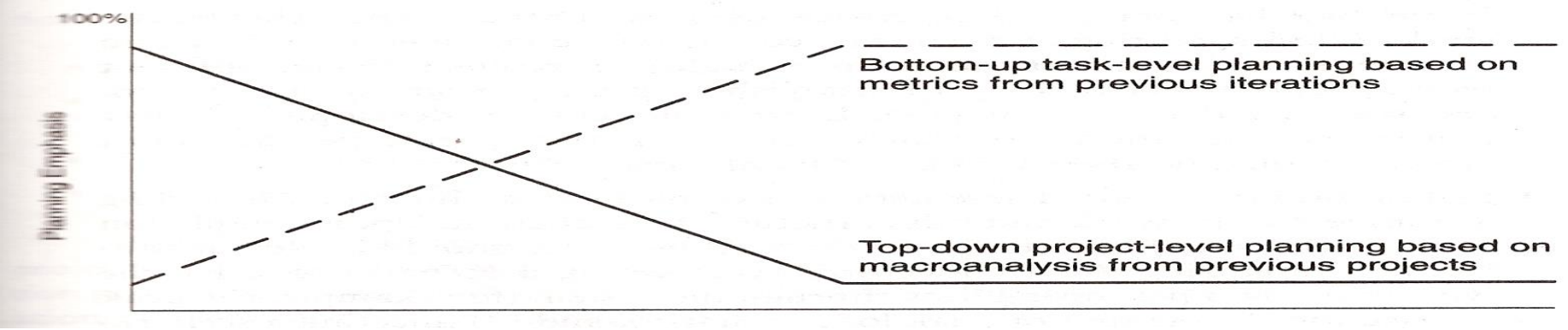
Domain	Inception	Elaboration	Construction	Transition
Effort	5%	20%	65%	10%
Schedule	10%	30%	50%	10%

4.9 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:
  - The software project manager (and others) develops a characterization of the overall size, process, environment, people, and quality required for the project.
  - A macro-level estimate of the total effort and schedule is developed using a software cost estimation model.
  - The software project manager partitions the estimate for the effort into top-level WBS using guidelines such as those in Table 10-1.
  - 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.



Engineering Stage		Production Stage	
Inception	Elaboration	Construction	Transition

Engineering stage planning emphasis:	Production 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	▪ 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

4.10 THE INTERATION PALNNING 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 circumstance. 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 candidate architecture and provide an executable framework for elaborating the critical use cases of eth system. This framework includes existing components, commercial components and custom prototypes sufficient to demonstrate candidate architecture and sufficient requirements understanding to establish a credible business case, vision and software development plan.

- **Elaboration Iteration:** 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 loan 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 iteration. 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 an two additional iterations in construction, for a total of nine iterations.

4.11 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 form bad architectures and misunderstood requirement, 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 stake holders, and it must be used. Plans are not just for mangers. 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.

_____

MODEL QUESTIONS

1. What is the content of minor checkpoints, major checkpoints and status assessments?
2. Who are the stakeholders and what are the major areas of their concern in software life cycle?
3. Discuss briefly about the major milestones in software life cycle.
4. Give the general status of plans, requirements and products across the major milestones in software life cycle.
5. What are immovable deadlines and movable deadlines in a project?
6. How are the checkpoints or synchronization points decided? Explain with an example.
7. Discuss in detail about the minor milestones in the life cycle of iteration.
8. Define periodic status assessment. What is the need of status assessment in software life cycle? Also discuss the default content of periodic status assessments.
9. What are the disadvantages of traditional work breakdown structures?
10. How should the evolutionary WBS be structured?
11. 'Two simple planning guidelines must be considered during project initiation and assessment'. Discuss them briefly.

12. Discuss the two perspectives of deriving project plans.
13. Explain in detail about the iteration planning process.
14. Compare and contrast engineering stage and production storage planning.
15. Write short notes on pragmatic type of planning.

_____

4.12. PROJECT ORGANIZATION AND RESPONSIBILITIES

INTRODUCTION: Software lines of business and project teams have different motivations. Software lines of business are motivated by return on investment, new business discriminators, market diversification and profitability. Software professionals in both types of organizations are motivated by career growth, job satisfaction and the opportunity to make a difference.

4.12.1 LINES-OF-BUSINESS ORGANIZATIONS: Figure 11-1 maps roles and responsibilities to a default line-of-business organization. This structure can be tailored to specific circumstances.

- The main features of the default organization are as follows:
  - Responsibility for process definition and maintenance is specific to a cohesive line of business.
  - Responsibility for process automation is an organizational role and is equal in importance to the process definition role.
- Organization roles may be fulfilled by a single individual or several different teams, depending on the scale of the organization.

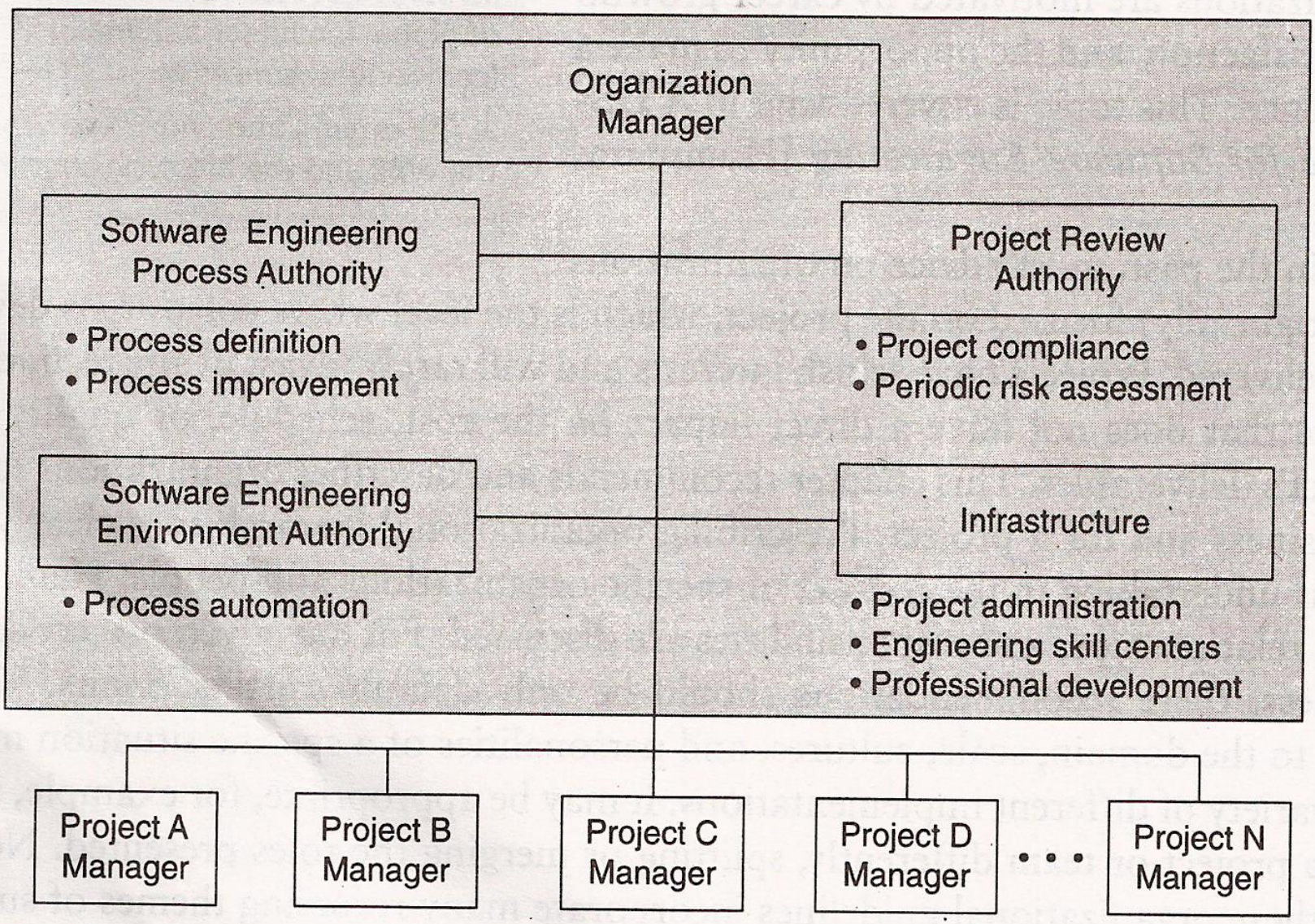


FIGURE 11-1. *Default roles in a software line-of-business organization*

- The line of business organization consists of four component teams.

- **SOFTWARE ENGINEERING PROCESS AUTHORITY**
- The software engineering process authority (SEPA) is responsible for exchanging the information and project guidance to or from the project practitioners.

- **PROJECT REVIEW AUTHORITY**
- The project review Authority (PRA) is responsible for reviewing the financial performance, customer commitments, risks and accomplishments, adherence to organizational policies by the customer etc.

- **SOFTWARE ENGINEERING ENVIRONMENT AUTHORITY**
- The software Engineering Environment Authority (SEEA) deals with the maintenance or organizations standard environment, training projects and process automation.

- **INFRASTRUCTURE**
- **An organization's infrastructure provides human resources support, project-independent research and development other capital software engineering assets. The typical components of the organizational infrastructure are as follows:**
  - **Project Administration: time accounting system; contracts, pricing, terms and conditions; corporate information systems integration.**
  - **Engineering Skill Centers: custom tools repository and maintenance, bid and proposal support, independent research and development.**
  - **Professional Development: Internal training boot camp, personnel recruiting, personnel skills database maintenance, literature and assets library, technical publications.**

4.13 PROJECT ORGANIZATIONS

- Figure 11-2 shows a default project organization and maps project-level roles and responsibilities. This structure can be tailored to the size and circumstance of the specific project organization are as follows:
  - *The project management team* is an active participant, responsible for producing as well as managing. Project management is not a spectator sport.
  - *The architecture team* is responsible for real artifacts and for the integration of components, not just for staff functions.
  - *The development team* owns the component construction and maintenance activities.
- *The assessment team* is separate form development.

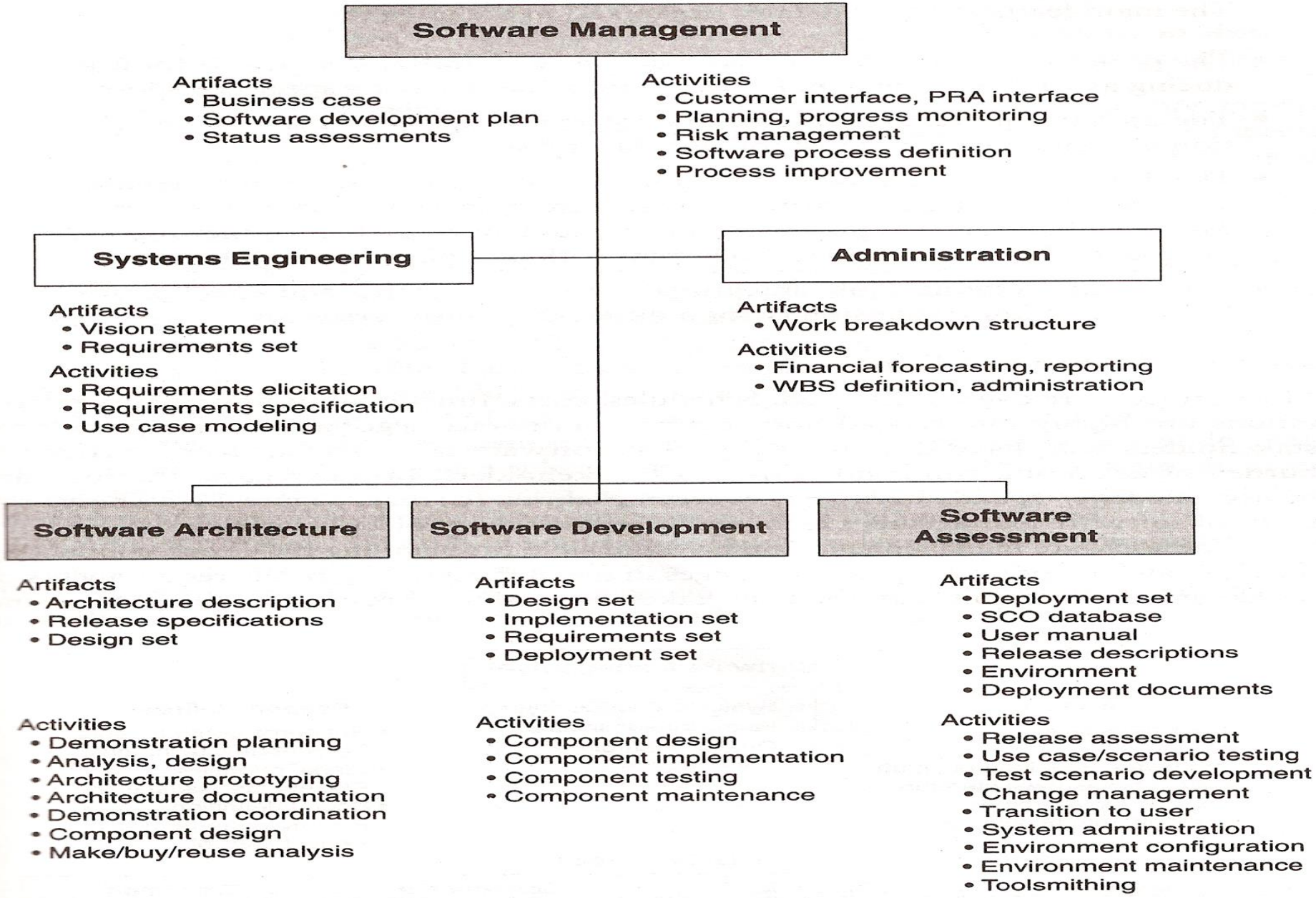


FIGURE 11-2. *Default project organization and responsibilities*

SOFTWARE MANAGEMENT TEAM

This is active participant in an organization and is incharge of producing as well as managing. As the software attributes, such as Schedules, costs, functionality and quality are interrelated to each other, negotiation among multiple stakeholders is required and these are carried out by the software management team.

Responsibilities: Software management team is responsible for:

- Effort planning
- Conducting the plan
- Adapting the plan according to the changes in requirements and design
- Resource management
- Stakeholders satisfaction
- Risk management
- Assignment or personnel
- Project controls and scope definition
- Quality assurance

SOFTWARE ARCHITECTURE TEAM

The software architecture team performs the tasks of integrating the components, creating real artifacts etc. The skill possessed by the architecture team is of utmost importance as it promotes team communications and implements the applications with a system-wide quality. The success of the development team is depends on the effectiveness of the architecture team along with the software management team controls the inception and elaboration phases of a life-cycle.

The architecture team must have:

Domain experience to generate an acceptable design and use-case view.

Software technology experience to generate an acceptable process view, component and development views.

- Responsibilities: Software architecture team is responsible for:
- System-level quality i.e., performance, reliability and maintainability.
- Requirements and design trade-offs.
- Component selection
- Technical risk solution
- Initial integration

- **SOFTWARE DEVELOPMENT TEAM**
  - The Development team is involved in the construction and maintenance activities. It is most application specific team. It consists of several sub teams assigned to the groups of components requiring a common skill set. The skill set include the following:
    - *Commercial component*: specialists with detailed knowledge of commercial components central to a system's architecture.
    - *Database*: specialists with experience in the organization, storage, and retrieval of data.
    - *Graphical user interfaces*: specialists with experience in the display organization; data presentation, and user interaction.
    - *Operating systems and networking*: specialists with experience in various control issues arises due to synchronization, resource sharing, reconfiguration, inter object communications, name space management etc.
    - *Domain applications*: Specialists with experience in the algorithms, application processing, or business rules specific to the system.
  - Responsibilities: Software development team is responsible for
    - Component development, testing and maintenance.
    - Component design and implementation
    - Component documentation.

- **SOFTWARE ASSESSMENT TEAM**
  - The team is involved in testing and product activities in parallel with the ongoing development. This is an independent team for utilizing the concurrency of activities. The use-case oriented and capability-based testing of a process is done by using two artifacts:
    - Release specification (the plan and evaluation criteria for a release);
    - Release description (the results of a release)
  - Responsibilities: The assessment team is responsible for
    - The exposure of the quality issues that affect the customer's expectations.
    - Metric analysis.
    - Verifying the requirements.
    - Independent testing.
    - Configuration control and user development.
    - Building project infrastructure.

4.14 EVOLUTION OF ORGANIZATIONS

- The project organization represents the architecture of the team and needs to evolve consistent with the project plan captured in the work breakdown structure. Figure 11-7 illustrates how the team's center of gravity shifts over the life cycle, with about 50% of the staff assigned to one set of activities in each phase.
- A different set of activities is emphasized in each phase, as follows:
  - **Inception team:** An organization focused on planning, with enough support from the other teams to ensure that the plans represent a consensus of all perspectives.
  - **Elaboration team:** An architecture-focused organization in which the driving forces of the project reside in the software architecture team and are supported, by the software development and software assessment teams as necessary to achieve a stable architecture baseline.
  - **Construction team:** A fairly balanced organization in which most of the activity resides in the software development and software assessment teams.
  - **Transition team:** A customer-focused organization[12] in which usage feedback drives the deployment activities

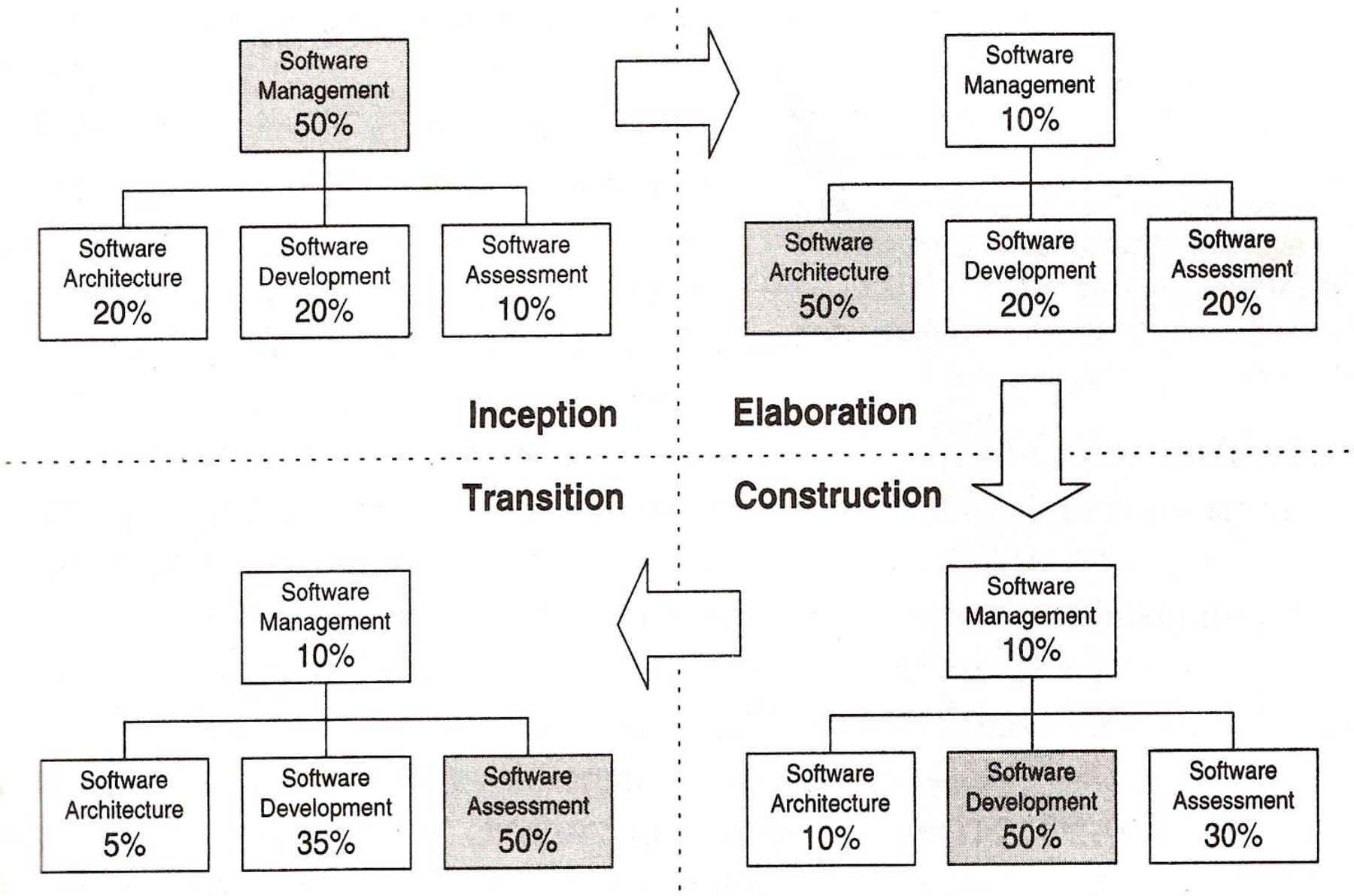


FIGURE 11-7. *Software project team evolution over the life cycle*

4.15 PROCESS AUTOMATION

- Three levels of process are
- **Metaprocess:** An organization's policies, procedures, and practices for managing a software intensive line of business. The automation support for this level is called an infrastructure. An infrastructure is an inventory of preferred tools, artifact templates, microprocess guidelines, macroprocess guidelines, project performance repository, database of organizational skill sets, and library of precedent examples of past project plans and results.
- **Macroprocess:** A project's policies, procedures, and practices for producing a complete software product within certain cost, schedule, and quality constraints. The automation support for a project's process is called an. environment. An environment is a specific collection of tools to produce a specific set of artifacts as governed by a specific project plan.
- **Microprocess:** A project team's policies, procedures, and practices for achieving an artifact of the software process. The automation support for generating an artifact is generally called a tool. Typical tools include requirements management, visual modeling, compilers, editors, debuggers, change management, metrics automation, document automation, test automation, cost estimation, and workflow automation.

4.16 TOOLS: AUTOMATION BUILDING BLOCKS

- It introduces some of the important tools that tend to be needed universally across software projects and that correlate well to the process framework. (Many other tools and process automation aids are not included.) Most of the core software development tools map closely to one of the process workflows, as illustrated ill Figure 12-1.

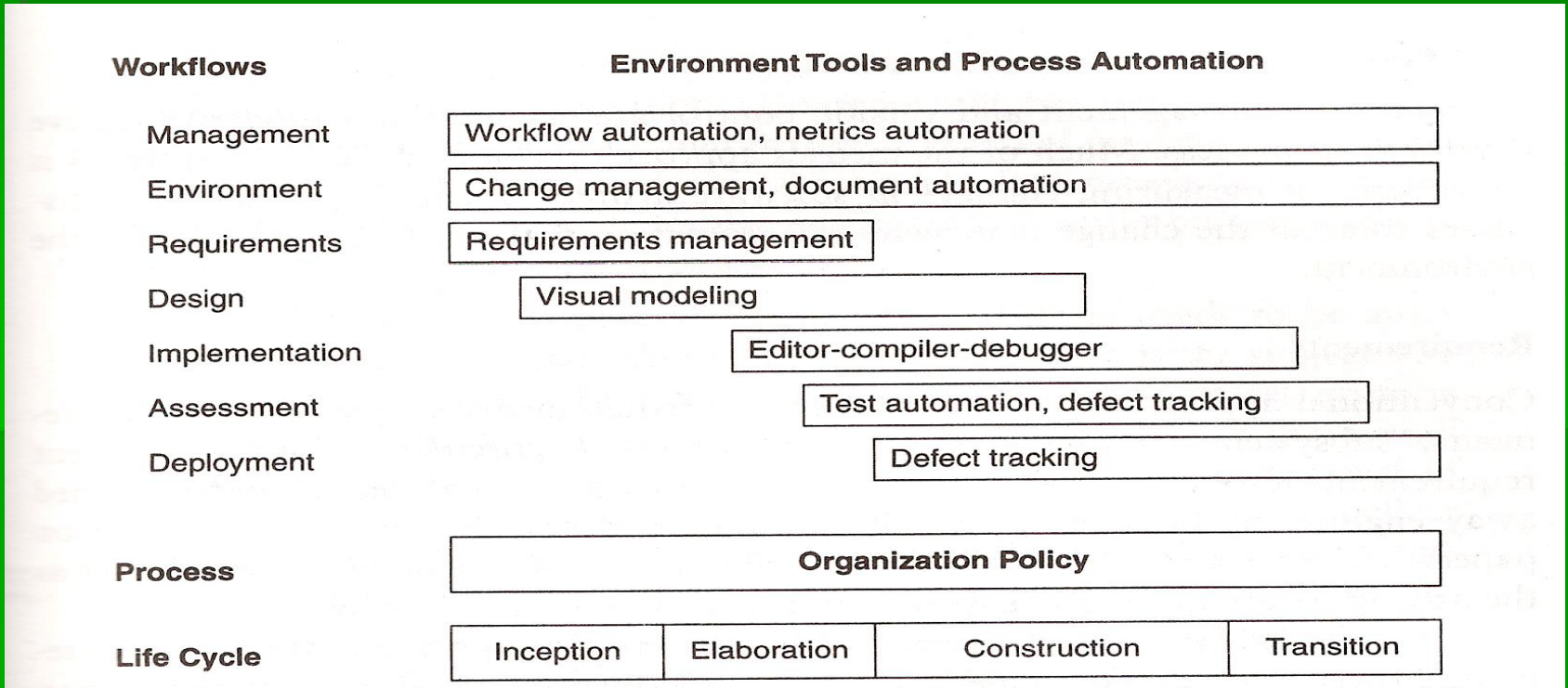


FIGURE 12-1. *Typical automation and tool components that support the process workflows*

- **MANAGEMENT**
  - There are many opportunities for automating the project planning and control activities of the management workflow. Software cost estimation tools and WBS tools are useful for generating the planning artifacts. For managing against a plan, workflow management tools and a software project control panel that can maintain an on-line version of the status assessment are advantageous. This automation support can considerably improve the insight of the metrics collection and reporting concepts.
- **ENVIRONMENT**
  - Configuration management and version control are essential in a modern iterative development process. (change management automation that must be supported by the environment.

- **REQUIREMENTS**
  - Conventional approaches decomposed system requirements into subsystem requirements, subsystem requirements into component requirements, and component requirements into unit requirements. The equal treatment of all requirements drained away engineering hours from the driving requirements then wasted that time on paperwork associated with detailed traceability that was inevitably discarded later as the driving requirements and subsequent design understanding evolved.
  - The ramifications of this approach on the environment's support for requirements management are twofold:

    1. The recommended requirements approach is dependent on both textual and model-based representations
    2. Traceability between requirements and other artifacts needs to be automated.
- **DESIGN**
  - The too1s that Support the requirements, design, implementation, and assessment workflows are usually used together. The primary support required for the design workflow is visual modeling, which is used for capturing design models, presenting them in human-readable format, and translating them into source code. Architecture-first and demonstration-based process is enabled by existing architecture components and middleware.

- **IMPLEMENTATION**
  - The implementation workflow relies primarily on a programming environment (editor, compiler, debugger, linker, run time) but must also include substantial integration with the change management tools, visual modeling tools, and test automation tools to support productive iteration.
- **ASSESSMENT AND DEPLOYMENT**
  - The assessment workflow requires all the tools just discussed as well as additional capabilities to support test automation and test management. To increase change freedom, testing and document production must be mostly automated. Defect tracking is another important tool that supports assessment: It provides the change management instrumentation necessary to automate metrics and control release baselines. It is also needed to support the deployment workflow throughout the life cycle.

4.17 THE PROJECT ENVIRONMENT

- The project environment artifacts evolve through three discrete states: the prototyping environment, the development environment, and the maintenance environment.
- The proto typing environment includes an architecture tested for prototyping project architectures to evaluate trade-offs during the inception and elaboration phases of the life cycle. This informal configuration of tools should be capable of supporting the following activities:
  - **Performance trade-offs and technical risk analyses**
  - **Make /buy trade-offs and feasibility studies for commercial products**
  - **Fault tolerance/dynamic reconfiguration trade-offs**
  - **Analysis of the risks associated with transitioning to full-scale implementation**
  - **Development of test scenarios, tools, and instrumentation suitable for analyzing the requirements.**
- The development environment should include a full suite of development tools needed to support the various process workflows and to support round-trip engineering to the maximum extent possible.
- The maintenance environment should typically coincide with a mature version of the development environment. In some cases, the maintenance environment may be a subset of the development environment delivered as one of the project's end products.

- Four important environment disciplines that is critical to the management context and the success of a modern iterative development process:
  - Tools must be integrated to maintain consistency and traceability. Roundtrip Engineering is the term used to describe this key requirement for environments that support iterative development.
  - Change management must be automated and enforced to manage multiple, iterations and to enable change freedom. Change is the fundamental primitive of iterative development.
  - Organizational infrastructures A common infrastructure promotes interproject consistency, reuse of training, reuse of lessons learned, and other strategic improvements to the organization's metaprocess.
  - Extending automation support for stakeholder environments enables further support for paperless exchange of information and more effective review of engineering artifacts.

4.17.1 ROUND-TRIP ENGINEERING

- Round-trip engineering is the environment support necessary to maintain consistency among the engineering artifacts.
- Figure 12-2 depicts some important transitions between information repositories. The automated translation of design models to source code (both forward and reverse engineering) is fairly well established. The automated translation of design models to process (distribution) models is also becoming straightforward through technologies such as ActiveX and the Common Object Request Broker Architecture (CORBA).
- The primary reason for round-trip engineering is to allow freedom in changing software engineering data sources.

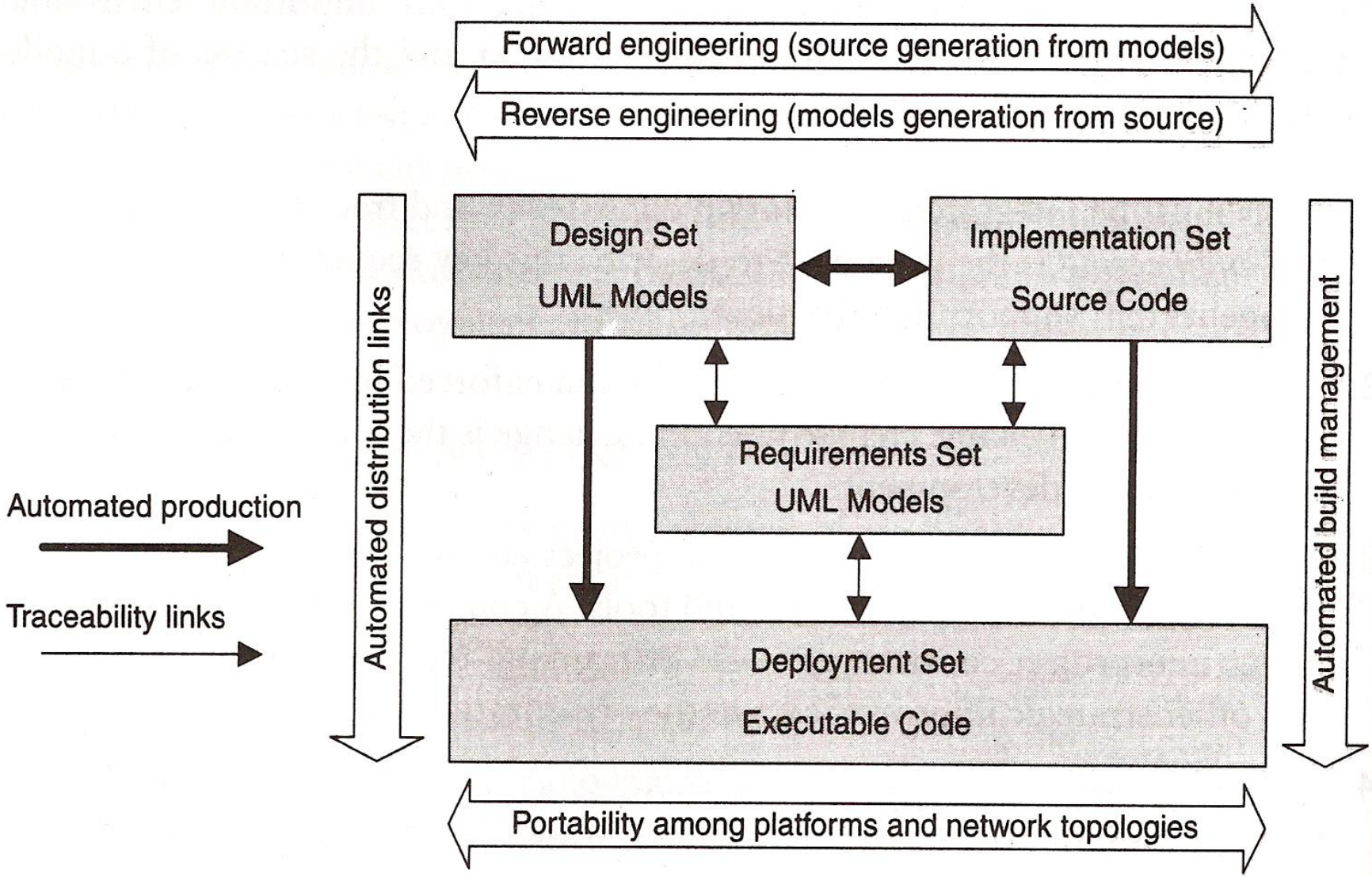


FIGURE 12-2. *Round-trip engineering*

4.17.2 CHANGE MANAGEMENT

- Change management is as critical to iterative processes as planning. Tracking changes in the technical artifacts is crucial to understanding the true technical progress trends and quality trends toward delivering an acceptable end product or interim release. In a modern process-in which requirements, design, and implementation set artifacts are captured in rigorous notations early in the life cycle and are evolved through multiple generations-change management has become fundamental to all phases and almost all activities.

SOFTWARE CHANGE ORDERS

- The atomic unit of software work that is authorized to create, modify, or obsolesce components within a configuration baseline is called a software change order (SCO). Software change orders are a key mechanism for partitioning, allocating, and scheduling software work against an established software baseline and for assessing progress and quality. The example SCO shown in Figure 12-3 is a good starting point for describing a set of change primitives. It shows the level of detail required to achieve the metrics and change management rigor necessary for a modern software process.
- The basic fields of the SCO are title, description, metrics, resolution, assessment and disposition.
- **Title.** The title is suggested by the originator and is finalized upon acceptance by the configuration control board (CCB).

- **Description:** The problem description includes the name of the originator, date of origination, CCB-assigned SCO identifier, and relevant version identifiers of related support software.
- **Metrics:** The metrics collected for each sea are important for planning, for scheduling, and for assessing quality improvement. Change categories are type 0 (critical bug), type 1 (bug), type 2 (enhancement), type 3 (new feature), and type 4 (other)
- **Resolution:** This field includes the name of the person responsible for implementing the change, the components changed, the actual metrics, and a description of the change.

- **Assessment:** This field describes the assessment technique as either inspection, analysis, demonstration, or test. Where applicable, it should also reference all existing test cases and new test cases executed, and it should identify all different test configurations, such as platforms, topologies, and compilers.
- **Disposition:** The SCO is assigned one of the following states by the CCB:
- **Proposed:** written, pending CCB review
- **Accepted:** CCB-approved for resolution
- **Rejected:** closed, with rationale, such as not a problem, duplicate, obsolete change, resolved by another SCO
- **Archived:** accepted but postponed until a later release
- **In progress:** assigned and actively being resolved by the development organization
- **In assessment:** resolved by the development organization; being assessed by a test organization
- **Closed:** completely resolved, with the concurrence of all CCB members.

Title: _____

Description

Name: _____ Date: _____

Project: _____

Metrics

Category: _____ (0/1 error, 2 enhancement, 3 new feature, 4 other)

Initial Estimate

Breakage: _____

Rework: _____

Actual Rework Expended

Analysis: _____ Test: _____

Implement: _____ Document: _____

Resolution

Analyst: _____

Software Component: _____

Assessment

Method: _____ (inspection, analysis, demonstration, test)

Tester: _____ Platforms: _____ Date: _____

Disposition

State: _____ Release: _____ Priority _____

Acceptance: _____ Date: _____

Closure: _____ Date: _____

FIGURE 12-3. *The primitive components of a software change order*

CONFIGURATION BASELINE

A configuration baseline is a named collection of software components and supporting documentation that is subject to change management and is upgraded, maintained, tested, statused and obsolesced as a unit.

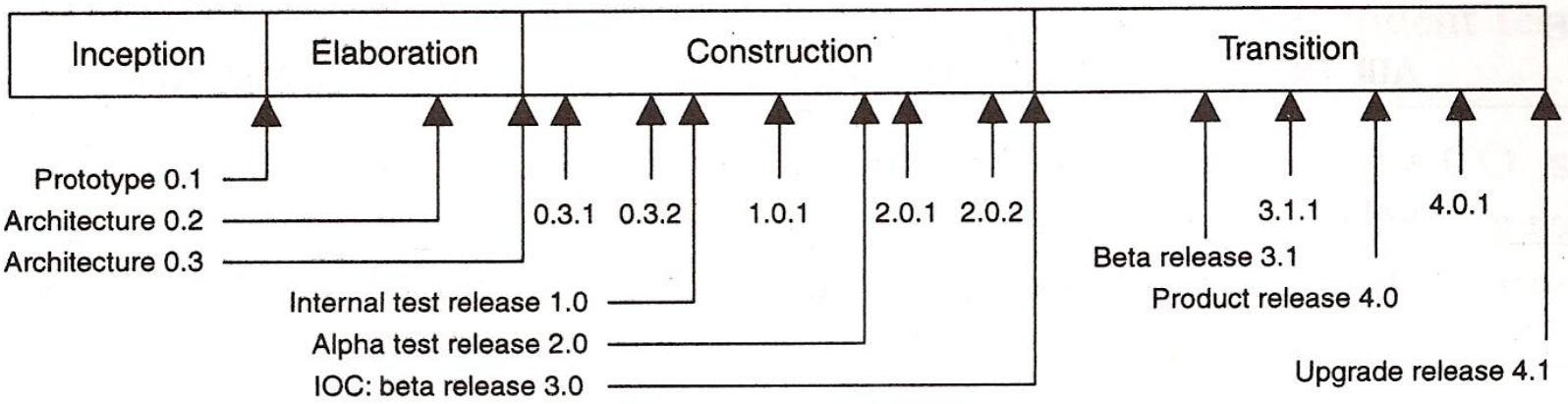
There are general1y two classes of baselines: external product releases and internal testing releases.

A configuration baseline is a named collection of components that is treated as a unit. It is controlled formally because it is a packaged exchange between groups. A project may release a configuration baseline to the user community for beta testing.

Generally, three levels of baseline releases arc required for most systems: major, minor, and interim. Each level corresponds to a numbered identifier such as N.M.X, where N is the major release number, M is the minor release number, and X is the interim release identifier. A major release represents a new generation of the product or project, while a minor release represents the same basic product but with enhanced features, performance, or quality. Major and minor releases are intended to be external product releases that are persistent and supported for a period of time. An interim release corresponds to a developmental configuration that is intended to be transient. The shorter its life cycle, the better. Figure 12-4 shows examples of some release name histories for two different situations.

- Once software is placed in a controlled baseline, all changes are tracked. A distinction must be made for the cause of a change. Change categories are as follows:
  - Type 0: Critical failures, which are defects that are nearly always fixed before any external release.
  - Type 1: A bug or defect that either does not impair the usefulness of the system or can be worked around.
  - Type 2: A change that is an enhancement rather than a response to a defect.
  - Type 3: A change that is necessitated by an update to the requirements.
  - Type 4: changes that are not accommodated by the other categories.
- Table 12-1 provides examples of these changes in the context of two different project domains: a large-scale, reliable air traffic control system and a packaged software development tool

Typical project release sequence for a large-scale, one-of-a-kind project



Typical project release sequence for a small commercial product

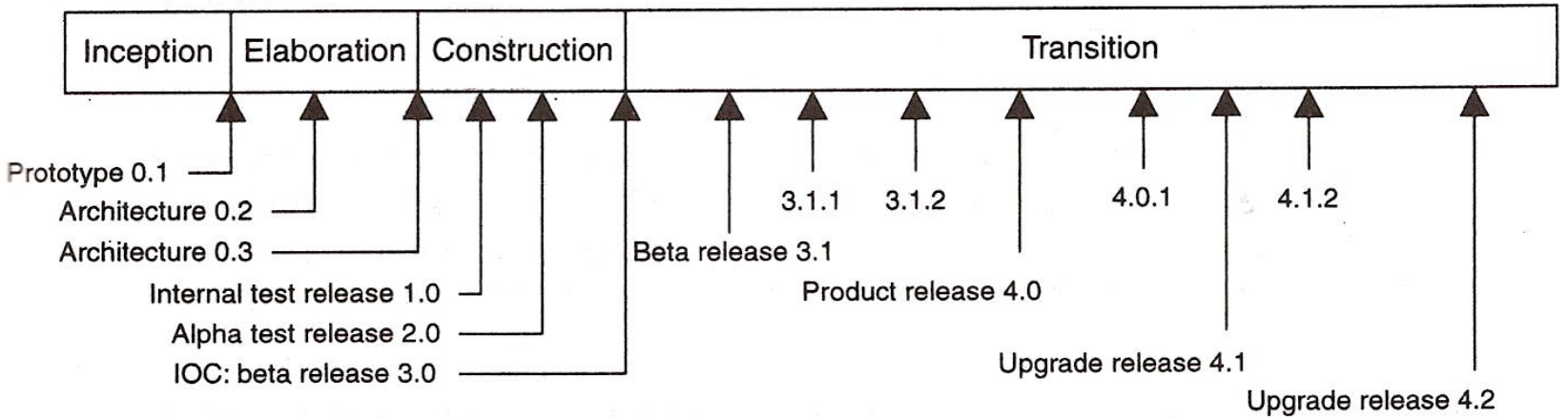


FIGURE 12-4. *Example release histories for a typical project and a typical product*

Change Type	Air Traffic control Project	Packaged visual Modeling Tool
Type 0	Control deadlock and loss of flight data	Loss of user data
Type 1	Display response time that exceeds the requirement by 0.5 second	Browser expands but does not collapse displayed entries
Type 2	Add internal message field for response time instrumentation	Use of color to differentiate updates from previous version of visual model
Type 3	Increase air traffic management capacity from 1,200 to 2,400 simultaneous flights	Port to new platform such as WinNT
Type 4	Upgrade from Oracle 7 to Oracle 8 to improve query performance	Exception raised when interfacing to MS Excel 5.0 due to windows resource management bug.

CONFIGURATION CONTROL BOARD

- A CCB is a team of people that functions as the decision authority on the content of configuration baselines. A CCB usually includes the software manager, software architecture manager, software development manager, software assessment manager and other stakeholders (customer, software project manager, systems engineer, user) who are integral to the maintenance of a controlled software delivery system. The [bracketed] words constitute the state of an SCO transitioning through the process.
- [Proposed]: A proposed change is drafted and submitted to the CCB. The proposed change must include a technical description of the problem and an estimate of the resolution effort.

- [Accepted, archived or rejected]: The CCB assigns a unique identifier and accepts, archives, or rejects each proposed change. Acceptance includes the change for resolution in the next release; archiving accepts the change but postpones it for resolution in a future release; and rejection judges the change to be without merit, redundant with other proposed changes, or out of scope.
- [In progress]: the responsible person analyzes, implements and tests a solution to satisfy the SCQ. This task includes updating documentation, release notes and SCO metrics actuals and submitting new SCOs.
- [In assessment]: The independent test assesses whether the SCO is completely resolved. When the independent test team deems the change to be satisfactorily resolved, the SCO is submitted to the CCB for final disposition and closure.
- [Closed]: when the development organization, independent test organization and CCB concur that the SCO is resolved, it is transitioned to a closed status.

4.17.3 INFRASTRUCTURES

- From a process automation perspective, the organization's infrastructure provides the organization capital assets, including two key artifacts: a policy that captures the standards for project software development processes, and an environment that captures an inventory of tools.
- **ORGANIZATION POLICY**
- The organization policy is usually packaged as a handbook that defines the life cycle and the process primitives (major milestones, intermediate artifacts, engineering repositories, metrics, roles and responsibilities). The handbook provides a general framework for answering the following questions:
  - What gets done? (activities and artifacts)
  - When does it get done? (mapping to the life-cycle phases and milestones)
  - Who does it? (team roles and responsibilities)
- How do we know that it is adequate? (Checkpoints, metrics and standards of performance).

- The need for balance is an important consideration in defining organizational policy. Effective organizational policies have several recurring themes:
  - They are concise and avoid policy statements that fill 6-inch-thick documents.
  - They confine the policies to the real shalls, then enforce them.
  - They avoid using the word should in policy statements. Rather than a menu of options (shoulds), policies need a concise set of mandatory standards (shalls).
  - Waivers are the exception, not the rule.
- Appropriate policy is written at the appropriate level.
- The organization policy is the defining document for the organization's software policies. In any process assessment, this is the tangible artifact that says what you do. From this document, reviewers should be able to question and review projects and personnel and determine whether the organization does what it says. Figure 12-5 shows a general outline for an organizational policy.

I. Process-Primitive definitions

A. Life-cycle phases (inception, elaboration, construction, transition)

B. Checkpoints (major milestones, minor milestones, status assessments)

C. Artifacts (requirements, design, implementation, deployment, management sets)

D. Roles and responsibilities (PRA, SEPA, SEEA, project teams).

II. Organization software policies

A. Work breakdown structure

B. Software development plan

C. Baseline change management

D. Software metrics

E. Development environment

F. Evaluation criteria and acceptance criteria

G. Risk management

H. Testing and assessment.

III. Walver policy

IV. Appendixes

A. Current process assessment

B. Software process improvement plan.

FIGURE 12-5: Organization policy outline

- **ORGANIZATION ENVIRONMENT**
  - Some of the typical components of an organization's automation building blocks are as follows:
    - Standardized tool selections (through investment by the organization in a site license or negotiation of a favorable discount with a tool vendor so that project teams are motivated economically to use that tool), which promote common workflows and a higher ROI on training.
    - Standard notations for artifacts, such as UML for all design models, or Ada 95 for all custom-developed, reliability-critical implementation artifacts.
    - Tool adjuncts such as existing artifact templates (architecture description, evaluation criteria, release descriptions, status assessment) or customizations.
    - Activity templates (iteration planning, major milestone activities, configuration control boards).

- Other indirectly useful components of an organization's infrastructure
  - A reference library of precedent experience for planning, assessing and improving process performance parameters; answers for how well? How much? Why?
  - Existing case studies, including objective benchmarks of performance for successful projects that followed the organization process.
  - A library of project artifact examples such as software development plans, architecture descriptions and status assessment histories.
  - Mock audits and compliance traceability for external process assessment frameworks.
- Such as the software Engineering Institute's Capability Maturity Model (SEI CMM)

4.17.4 STAKEHOLDER ENVIRONMENTS

- The transition to a modern iterative development process with supporting automation should not be restricted to the development team. many large scale contractual projects include people in external organization that represent other stakeholders participating in the development process.
- An on-line environment accessible by the external stakeholders allows them to participate in the process as follows:
  - Accept and use executable increments for hands-on evaluation.
  - Use the same on-line tools, data and reports that the software development organization uses to manage and monitor the project.
  - Avoid excessive travel, paper interchange delays, format translations, paper and shipping costs and other overhead costs.

- **FIGURE 12-6:** Illustrates some of the new opportunities for value-added activities by external stakeholders in large contractual efforts. There are several important reasons for extending development environment resources into certain stakeholder domains.
  - Technical artifacts are not just paper. Electronic artifacts in rigorous notations such as visual models and source code are viewed far more efficiently by using tools with smart browsers.
  - Independent assessments of the evolving artifacts are encouraged by electronic read-only access to on-line data such as configuration baseline libraries and the change management database. Reviews and inspections, breakage/rework assessments, metrics analyses and even beta testing can be performed independently of the development team.
  - Even paper documents should be delivered electronically to reduce production costs and turn around time.

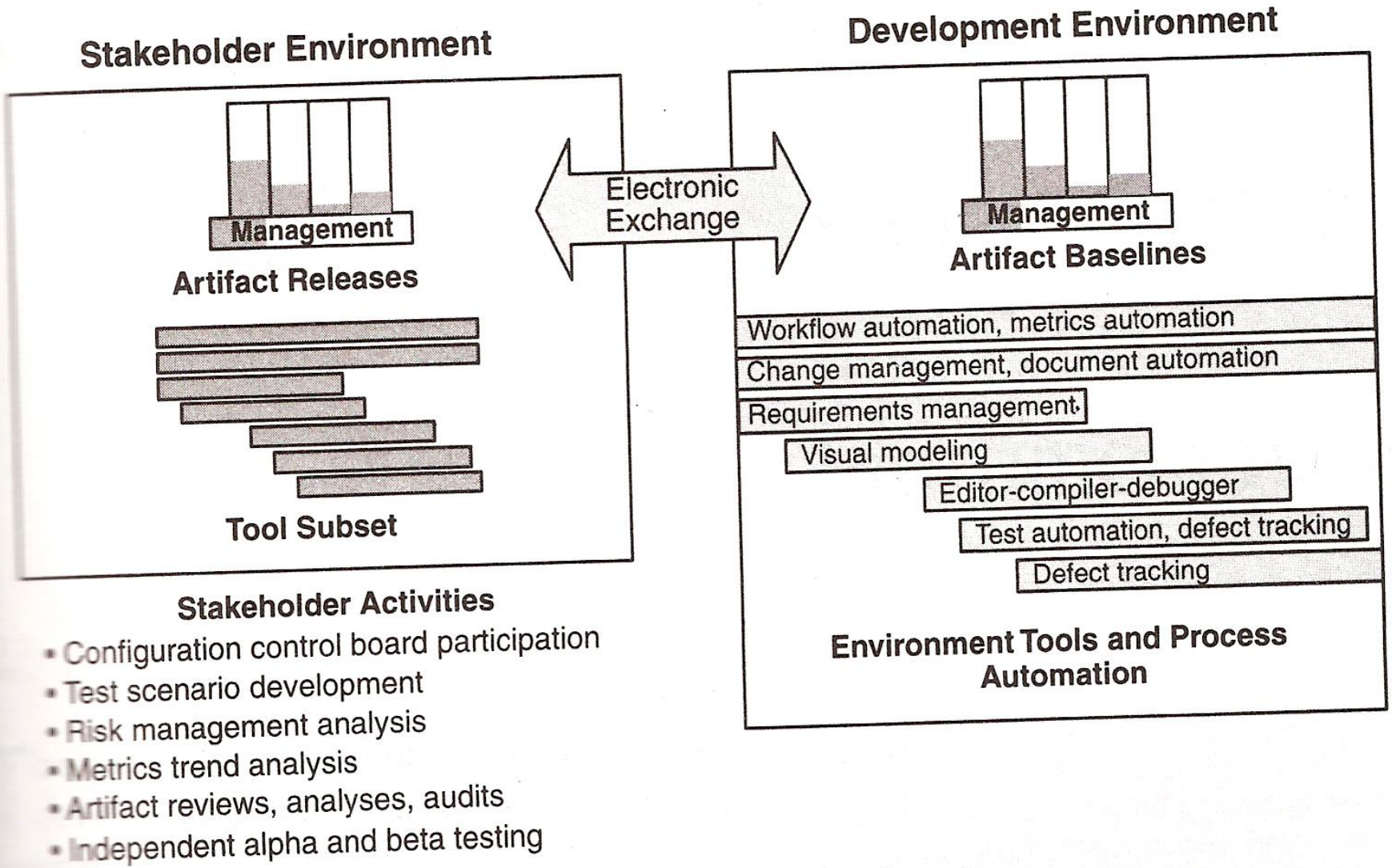


FIGURE 12-6. *Extending environments into stakeholder domains*

MODEL QUESTIONS

1. What are the main features of the default line-of-business organization?
2. What are the four component teams in a default line-of-business organization and their responsibility?
3. What are the four component teams in a default project organization and their responsibility?
4. How does the emphasis in the four teams evolve over the course of the entire project?
5. What is the reason for looking at organizations from project as well as line-of-business perspective?
6. What are the steps in identifying project roles? Name any five project roles and the skills needed for them.
7. What are the benefits of matching people to roles?
8. Explain the process of critical path method on the following PERT chart with requirements, coding and system taking 4 weeks and each others taking 2 weeks each. Shown in figure.
9. Discuss the evolution of software project team over the software life cycle.
10. Discuss the three levels of process along with their automation support.

11. ‘Many automation tools are available for software development process’. Support your answer.
12. Discuss the major critical concerns associated with the software workflows.
13. What are the three discrete states of the project environment? Also discuss the four environment disciplines that are critical to the management.
14. Explain briefly about Round-trip engineering.
15. What are software change orders? Explain the various fields of SCO.
16. What are the sources of change? Why should change be made in a controlled way?
17. Define a configuration baseline.
18. Write short notes on
    - Configuration Control Board (CCB)
    - Organization’s Infrastructure.
19. Discuss briefly about the stakeholder environments.