

Important questions

1.	<i>Explain briefly Waterfall model. Also explain Conventional s/w management performance?</i>
2.	<i>Define Software Economics. Also explain Pragmatic s/w cost estimation?</i>
3.	<i>Explain Important trends in improving Software economics?</i>
4.	<i>Explain five staffing principal offered by Boehm. Also explain Peer Inspections?</i>
5..	<i>Explain principles of conventional software engineering?</i>
6.	<i>Explain briefly principles of modern software management</i>

5. Life cycle phases

Characteristic of a successful software development process is the well-defined separation between "research and development" activities and "production" activities. Most unsuccessful projects exhibit one of the following characteristics:

- An overemphasis on research and development
- An overemphasis on production.

Successful modern projects-and even successful projects developed under the conventional process-tend to have a very well-defined project milestone when there is a noticeable transition from a research attitude to a production attitude. Earlier phases focus on achieving functionality. Later phases revolve around achieving a product that can be shipped to a customer, with explicit attention to robustness, performance, and finish.

A modern software development process must be defined to support the following:

- Evolution of the plans, requirements, and architecture, together with well defined synchronization points
- Risk management and objective measures of progress and quality
- Evolution of system capabilities through demonstrations of increasing functionality

5.1 ENGINEERING AND PRODUCTION STAGES

To achieve economies of scale and higher returns on investment, we must move toward a software manufacturing process driven by technological improvements in process automation and component-based development. Two stages of the life cycle are:

1. The **engineering stage**, driven by less predictable but smaller teams doing design and synthesis activities
2. The **production stage**, driven by more predictable but larger teams doing construction, test, and deployment activities

TABLE 5-1. *The two stages of the life cycle: engineering and production*

LIFE-CYCLE ASPECT	ENGINEERING STAGE EMPHASIS	PRODUCTION STAGE EMPHASIS
Risk reduction	Schedule, technical feasibility	Cost
Products	Architecture baseline	Product release baselines
Activities	Analysis, design, planning	Implementation, testing
Assessment	Demonstration, inspection, analysis	Testing
Economics	Resolving diseconomies of scale	Exploiting economies of scale
Management	Planning	Operations

The transition between engineering and production is a crucial event for the various stakeholders. The production plan has been agreed upon, and there is a good enough understanding of the problem and the solution that all stakeholders can make a firm commitment to go ahead with production.

Engineering stage is decomposed into two distinct phases, inception and elaboration, and the production stage into construction and transition. These four phases of the life-cycle process are loosely mapped to the conceptual framework of the spiral model as shown in Figure 5-1

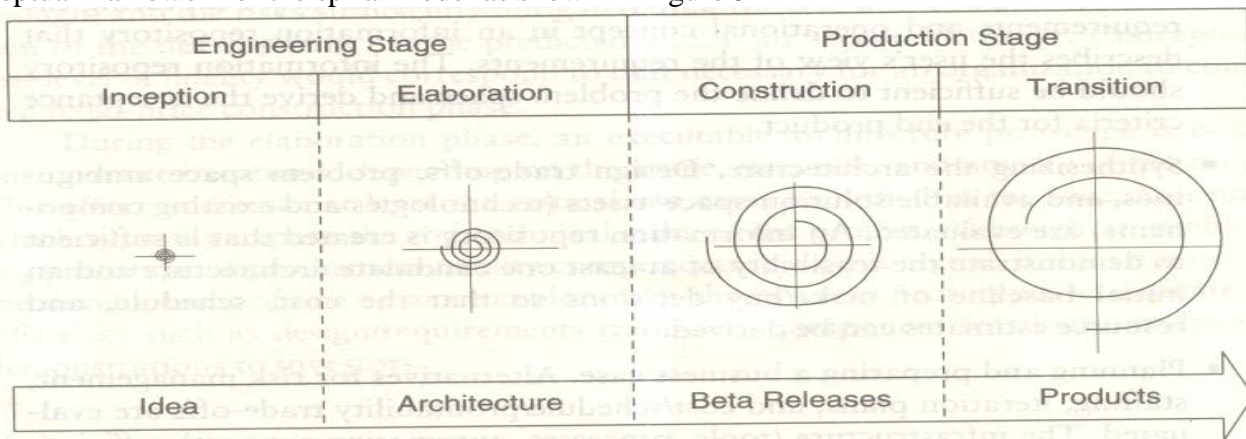


FIGURE 5-1. *The phases of the life-cycle process*

5.2 INCEPTION PHASE

The overriding goal of the inception phase is to achieve concurrence among stakeholders on the life-cycle objectives for the project.

PRIMARY OBJECTIVES

- Establishing the project's software scope and boundary conditions, including an operational concept, acceptance criteria, and a clear understanding of what is and is not intended to be in the product
- Discriminating the critical use cases of the system and the primary scenarios of operation that will drive the major design trade-offs
- Demonstrating at least one candidate architecture against some of the primary scenanos
- Estimating the cost and schedule for the entire project (including detailed estimates for the elaboration phase)
- Estimating potential risks (sources of unpredictability)

ESSENTIAL ACTIVITIES

- Formulating the scope of the project. The information repository should be sufficient to define the problem space and derive the acceptance criteria for the end product.
- Synthesizing the architecture. An information repository is created that is sufficient to demonstrate the feasibility of at least one candidate architecture and an, initial baseline of make/buy decisions so that the cost, schedule, and resource estimates can be derived.
- Planning and preparing a business case. Alternatives for risk management, staffing, iteration plans, and cost/schedule/profitability trade-offs are evaluated.

PRIMARY EVALUATION CRITERIA

- Do all stakeholders concur on the scope definition and cost and schedule estimates?
- Are requirements understood, as evidenced by the fidelity of the critical use cases?
- Are the cost and schedule estimates, priorities, risks, and development processes credible?
- Do the depth and breadth of an architecture prototype demonstrate the preceding criteria? (The primary value of prototyping candidate architecture is to provide a vehicle for understanding the scope and assessing the credibility of the development group in solving the particular technical problem.)
- Are actual resource expenditures versus planned expenditures acceptable

5.2 ELABORATION PHASE

At the end of this phase, the "engineering" is considered complete. The elaboration phase activities must ensure that the architecture, requirements, and plans are stable enough, and the risks sufficiently mitigated, that the cost and schedule for the completion of the development can be predicted within an acceptable range. During the elaboration phase, an executable architecture prototype is built in one or more iterations, depending on the scope, size, & risk.

PRIMARY OBJECTIVES

- Baseline the architecture as rapidly as practical (establishing a configuration-managed snapshot in which all changes are rationalized, tracked, and maintained)
- Baseline the vision
- Baseline a high-fidelity plan for the construction phase
- Demonstrating that the baseline architecture will support the vision at a reasonable cost in a reasonable time

ESSENTIAL ACTIVITIES

- Elaborating the vision.
- Elaborating the process and infrastructure.
- Elaborating the architecture and selecting components.

PRIMARY EVALUATION CRITERIA

- Is the vision stable?
- Is the architecture stable?
- Does the executable demonstration show that the major risk elements have been addressed and credibly resolved?
- Is the construction phase plan of sufficient fidelity, and is it backed up with a credible basis of estimate?
- Do all stakeholders agree that the current vision can be met if the current plan is executed to develop the

complete system in the context of the current architecture?

- Are actual resource expenditures versus planned expenditures acceptable?

5.4 CONSTRUCTION PHASE

During the construction phase, all remaining components and application features are integrated into the application, and all features are thoroughly tested. Newly developed software is integrated where required. The construction phase represents a production process, in which emphasis is placed on managing resources and controlling operations to optimize costs, schedules, and quality.

PRIMARY OBJECTIVES

- Minimizing development costs by optimizing resources and avoiding unnecessary scrap and rework
- Achieving adequate quality as rapidly as practical
- Achieving useful versions (alpha, beta, and other test releases) as rapidly as practical

ESSENTIAL ACTIVITIES

- Resource management, control, and process optimization
- Complete component development and testing against evaluation criteria
- Assessment of product releases against acceptance criteria of the vision

PRIMARY EVALUATION CRITERIA

- Is this product baseline mature enough to be deployed in the user community? (Existing defects are not obstacles to achieving the purpose of the next release.)
- Is this product baseline stable enough to be deployed in the user community? (Pending changes are not obstacles to achieving the purpose of the next release.)
- Are the stakeholders ready for transition to the user community?
- Are actual resource expenditures versus planned expenditures acceptable?

5.5 TRANSITION PHASE

The transition phase is entered when a baseline is mature enough to be deployed in the end-user domain. This typically requires that a usable subset of the system has been achieved with acceptable quality levels and user documentation so that transition to the user will provide positive results. This phase could include any of the following activities:

1. Beta testing to validate the new system against user expectations
2. Beta testing and parallel operation relative to a legacy system it is replacing
3. Conversion of operational databases
4. Training of users and maintainers

The transition phase concludes when the deployment baseline has achieved the complete vision.

PRIMARY OBJECTIVES

- Achieving user self-supportability
- Achieving stakeholder concurrence that deployment baselines are complete and consistent with the evaluation criteria of the vision
- Achieving final product baselines as rapidly and cost-effectively as practical

ESSENTIAL ACTIVITIES

- Synchronization and integration of concurrent construction increments into consistent deployment baselines
- Deployment-specific engineering (cutover, commercial packaging and production, sales rollout kit development, field personnel training)
- Assessment of deployment baselines against the complete vision and acceptance criteria in the requirements set

EVALUATION CRITERIA

- Is the user satisfied?
- Are actual resource expenditures versus planned expenditures acceptable?

UNIT - III

Artifacts of the process: The artifact sets, Management artifacts, Engineering artifacts, programmatic artifacts.

Model based software architectures: A Management perspective and technical perspective.

6. Artifacts of the process

6.1 THE ARTIFACT SETS

To make the development of a complete software system manageable, distinct collections of information are organized into artifact sets. *Artifact* represents cohesive information that typically is developed and reviewed as a single entity.

Life-cycle software artifacts are organized into five distinct sets that are roughly partitioned by the underlying language of the set: management (ad hoc textual formats), requirements (organized text and models of the problem space), design (models of the solution space), implementation (human-readable programming language and associated source files), and deployment (machine-process able languages and associated files). The artifact sets are shown in Figure 6-1.

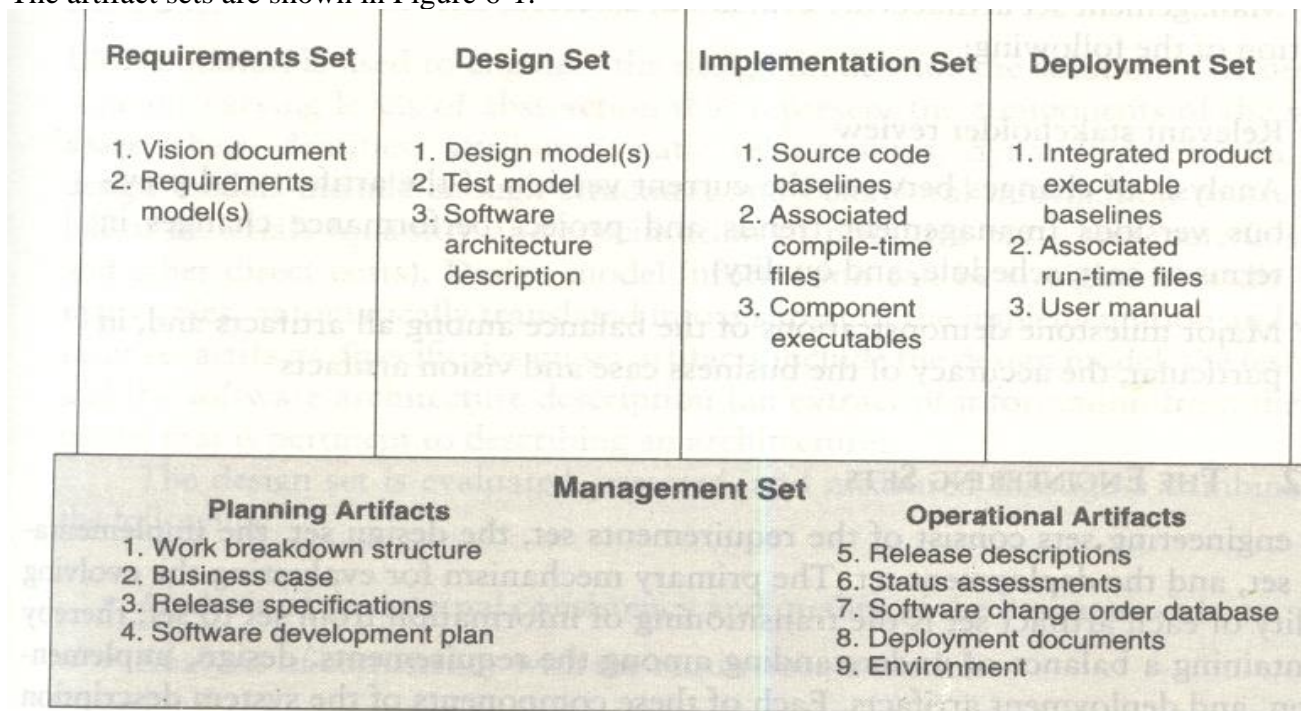


FIGURE 6-1. Overview of the artifact sets

6.1.1 THE MANAGEMENT SET

The management set captures the artifacts associated with process planning and execution. These artifacts use ad hoc notations, including text, graphics, or whatever representation is required to capture the "contracts" among project personnel (project management, architects, developers, testers, marketers, administrators), among stakeholders (funding authority, user, software project manager, organization manager, regulatory agency), and between project personnel and stakeholders. Specific artifacts included in this set are the work breakdown structure (activity breakdown and financial tracking mechanism), the business case (cost, schedule, profit expectations), the release specifications (scope, plan, objectives for release baselines), the software development plan (project process instance), the release descriptions (results of release baselines), the status assessments (periodic snapshots of project progress), the software change orders (descriptions of discrete baseline changes), the deployment documents (cutover plan, training course, sales rollout kit), and the environment (hardware and software tools, process automation, & documentation).

Management set artifacts are evaluated, assessed, and measured through a combination of the following:

- Relevant stakeholder review
- Analysis of changes between the current version of the artifact and previous versions
- Major milestone demonstrations of the balance among all artifacts and, in particular, the accuracy of the business case and vision artifacts

6.1.2 THE ENGINEERING SETS

The engineering sets consist of the requirements set, the design set, the implementation set, and the deployment set.

Requirements Set

Requirements artifacts are evaluated, assessed, and measured through a combination of the following:

- Analysis of consistency with the release specifications of the management set
- Analysis of consistency between the vision and the requirements models
- Mapping against the design, implementation, and deployment sets to evaluate the consistency and completeness and the semantic balance between information in the different sets
- Analysis of changes between the current version of requirements artifacts and previous versions (scrap, rework, and defect elimination trends)
- Subjective review of other dimensions of quality

Design Set

UML notation is used to engineer the design models for the solution. The design set contains varying levels of abstraction that represent the components of the solution space (their identities, attributes, static relationships, dynamic interactions). The design set is evaluated, assessed, and measured through a combination of the following:

- Analysis of the internal consistency and quality of the design model
- Analysis of consistency with the requirements models
- Translation into implementation and deployment sets and notations (for example, traceability, source code generation, compilation, linking) to evaluate the consistency and completeness and the semantic balance between information in the sets
- Analysis of changes between the current version of the design model and previous versions (scrap, rework, and defect elimination trends)
- Subjective review of other dimensions of quality

Implementation set

The implementation set includes source code (programming language notations) that represents the tangible implementations of components (their form, interface, and dependency relationships)

Implementation sets are human-readable formats that are evaluated, assessed, and measured through a combination of the following:

- Analysis of consistency with the design models
- Translation into deployment set notations (for example, compilation and linking) to evaluate the consistency and completeness among artifact sets
- Assessment of component source or executable files against relevant evaluation criteria through inspection, analysis, demonstration, or testing
- Execution of stand-alone component test cases that automatically compare expected results with actual results
- Analysis of changes between the current version of the implementation set and previous versions (scrap, rework, and defect elimination trends)
- Subjective review of other dimensions of quality

Deployment Set

The deployment set includes user deliverables and machine language notations, executable software, and the build scripts, installation scripts, and executable target specific data necessary to use the product in its target environment.

Deployment sets are evaluated, assessed, and measured through a combination of the following:

- Testing against the usage scenarios and quality attributes defined in the requirements set to evaluate the consistency and completeness and the semantic balance between information in the two sets
- Testing the partitioning, replication, and allocation strategies in mapping components of the implementation set to physical resources of the deployment system (platform type, number, network topology)
- Testing against the defined usage scenarios in the user manual such as installation, user-oriented dynamic reconfiguration, mainstream usage, and anomaly management
- Analysis of changes between the current version of the deployment set and previous versions (defect elimination trends, performance changes)
- Subjective review of other dimensions of quality

Each artifact set is the predominant development focus of one phase of the life cycle; the other sets take on check and balance roles. As illustrated in Figure 6-2, each phase has a predominant focus: Requirements are the focus of the inception phase; design, the elaboration phase; implementation, the construction phase; and deployment, the transition phase. The management artifacts also evolve, but at a fairly constant level across the life cycle.

Most of today's software development tools map closely to one of the five artifact sets.

1. Management: scheduling, workflow, defect tracking, change management, documentation, spreadsheet, resource management, and presentation tools
2. Requirements: requirements management tools
3. Design: visual modeling tools
4. Implementation: compiler/debugger tools, code analysis tools, test coverage analysis tools, and test management tools
5. Deployment: test coverage and test automation tools, network management tools, commercial components (operating systems, GUIs, RDBMS, networks, middleware), and installation tools.

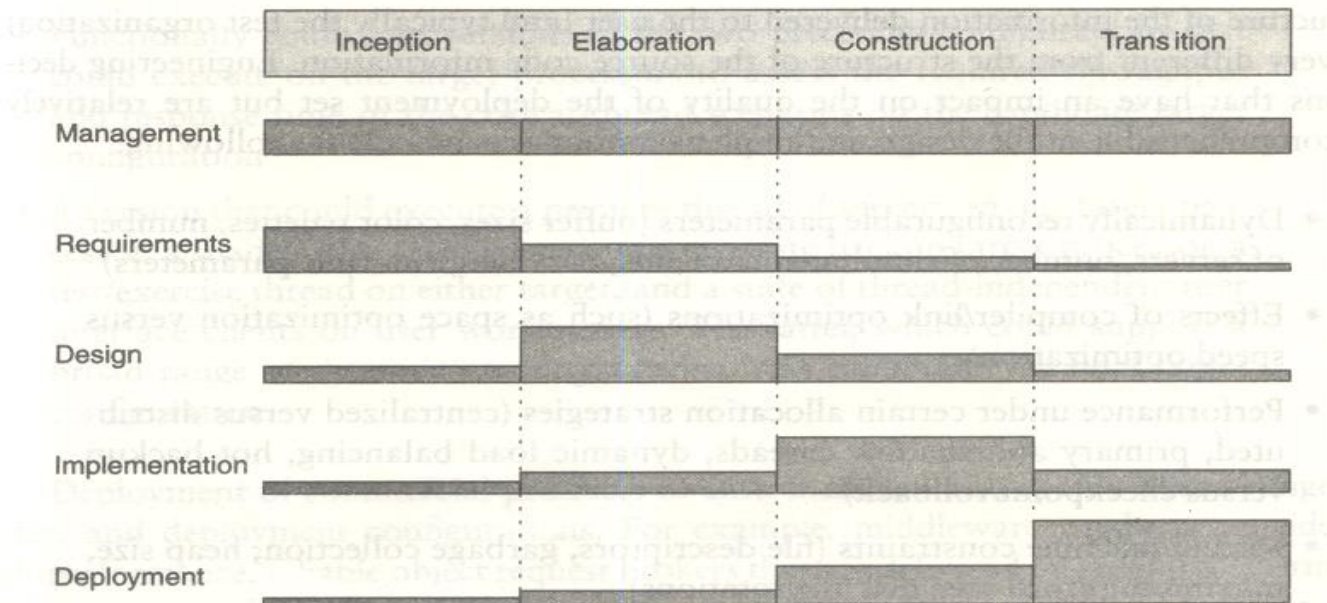


FIGURE 6-2. Life-cycle focus on artifact sets

Implementation Set versus Deployment Set

The separation of the implementation set (source code) from the deployment set (executable code) is important because there are very different concerns with each set. The structure of the information delivered to the user (and typically the test organization) is very different from the structure of the source code information. Engineering decisions that have an impact on the quality of the deployment set but are relatively incomprehensible in the design and implementation sets include the following:

- Dynamically reconfigurable parameters (buffer sizes, color palettes, number of servers, number of simultaneous clients, data files, run-time parameters)
- Effects of compiler/link optimizations (such as space optimization versus speed optimization)
- Performance under certain allocation strategies (centralized versus distributed, primary and shadow threads, dynamic load balancing, hot backup versus checkpoint/rollback)
- Virtual machine constraints (file descriptors, garbage collection, heap size, maximum record size, disk file rotations)
- Process-level concurrency issues (deadlock and race conditions)
- Platform-specific differences in performance or behavior

6.1.3 ARTIFACT EVOLUTION OVER THE LIFE CYCLE

Each state of development represents a certain amount of precision in the final system description. Early in the life cycle, precision is low and the representation is generally high. Eventually, the precision of representation is high and everything is specified in full detail. Each phase of development focuses on a particular artifact set. At the end of each phase, the overall system state will have progressed on all sets, as illustrated in Figure 6-3.

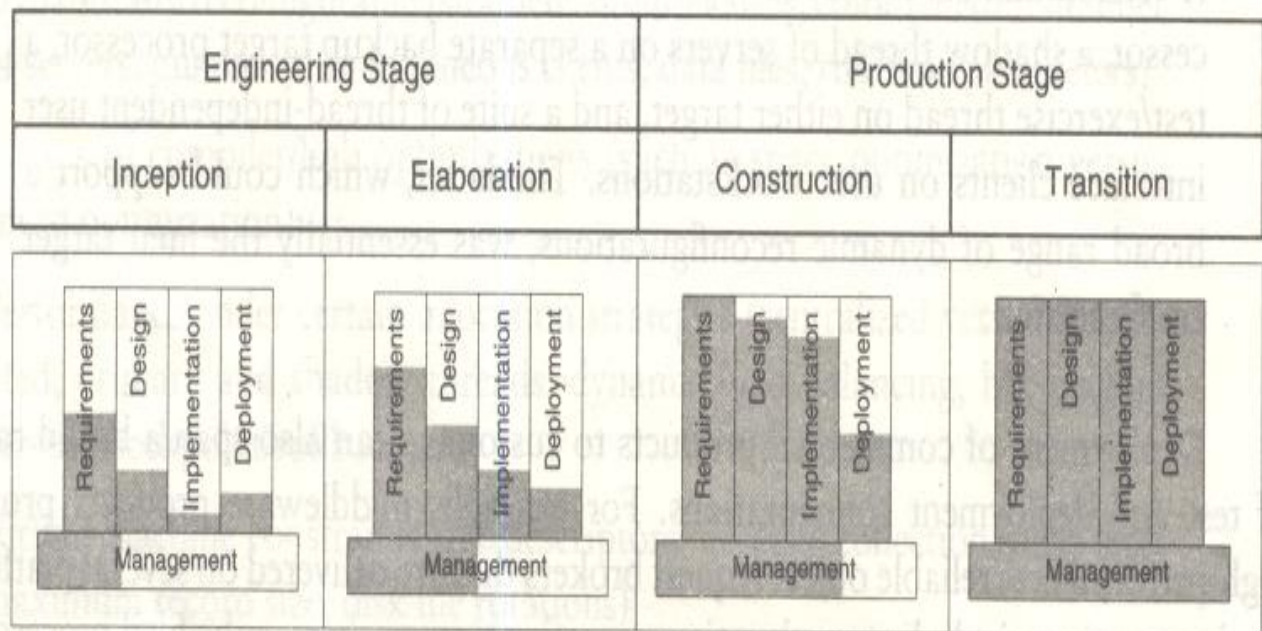


FIGURE 6-3. *Life-cycle evolution of the artifact sets*

The **inception** phase focuses mainly on critical requirements usually with a secondary focus on an initial deployment view. During the **elaboration phase**, there is much greater depth in requirements, much more breadth in the design set, and further work on implementation and deployment issues. The main focus of the **construction** phase is design and implementation. The main focus of the **transition** phase is on achieving consistency and completeness of the deployment set in the context of the other sets.

6.1.4 TEST ARTIFACTS

- The test artifacts must be developed concurrently with the product from inception through deployment. Thus, testing is a full-life-cycle activity, not a late life-cycle activity.
- The test artifacts are communicated, engineered, and developed within the same artifact sets as the developed product.
- The test artifacts are implemented in programmable and repeatable formats (as software programs).
- The test artifacts are documented in the same way that the product is documented.
- Developers of the test artifacts use the same tools, techniques, and training as the software engineers developing the product.

Test artifact subsets are highly project-specific, the following example clarifies the relationship between test artifacts and the other artifact sets. Consider a project to perform seismic data processing for the purpose of oil exploration. This system has three fundamental subsystems: (1) a sensor subsystem that captures raw seismic data in real time and delivers these data to (2) a technical operations subsystem that converts raw data into an organized database and manages queries to this database from (3) a display subsystem that allows workstation operators to examine seismic data in human-readable form. Such a system would result in the following test artifacts:

- Management set. The release specifications and release descriptions capture the objectives, evaluation criteria, and results of an intermediate milestone. These artifacts are the test plans and test results negotiated among internal project teams. The software change orders capture test results (defects, testability changes, requirements ambiguities, enhancements) and the closure criteria associated with making a discrete change to a baseline.
- Requirements set. The system-level use cases capture the operational concept for the system and the acceptance test case descriptions, including the expected behavior of the system and its quality attributes. The entire requirement set is a test artifact because it is the basis of all assessment activities across the life cycle.
- Design set. A test model for nondeliverable components needed to test the product baselines is captured in the design set. These components include such design set artifacts as a seismic event simulation for creating realistic sensor data; a "virtual operator" that can support unattended, after-hours test cases; specific instrumentation suites for early demonstration of resource usage; transaction rates or response times; and use case test drivers and component stand-alone test drivers.
- Implementation set. Self-documenting source code representations for test components and test drivers provide the equivalent of test procedures and test scripts. These source files may also include human-readable data files representing certain statically defined data sets that are explicit test source files. Output files from test drivers provide the equivalent of test reports.
- Deployment set. Executable versions of test components, test drivers, and data files are provided.

6.2 MANAGEMENT ARTIFACTS

The management set includes several artifacts that capture intermediate results and ancillary information necessary to document the product/process legacy, maintain the product, improve the product, and improve the process.

Business Case

The business case artifact provides all the information necessary to determine whether the project is worth investing in. It details the expected revenue, expected cost, technical and management plans, and backup data necessary to demonstrate the risks and realism of the plans. The main purpose is to transform the vision into economic terms so that an organization can make an accurate ROI assessment. The financial forecasts are evolutionary, updated with more accurate forecasts as the life cycle progresses. Figure 6-4

provides a default outline for a business case.

Software Development Plan

The software development plan (SDP) elaborates the process framework into a fully detailed plan. Two indications of a useful SDP are periodic updating (it is not stagnant shelfware) and understanding and acceptance by managers and practitioners alike. Figure 6-5 provides a default outline for a software development plan.

-
- I. Context (domain, market, scope)**
 - II. Technical approach**
 - A. Feature set achievement plan
 - B. Quality achievement plan
 - C. Engineering trade-offs and technical risks
 - III. Management approach**
 - A. Schedule and schedule risk assessment
 - B. Objective measures of success
 - IV. Evolutionary appendixes**
 - A. Financial forecast
 - 1. Cost estimate
 - 2. Revenue estimate
 - 3. Bases of estimates

FIGURE 6-4. *Typical business case outline*

-
- I. Context (scope, objectives)**
 - II. Software development process**
 - A. Project primitives
 - 1. Life-cycle phases
 - 2. Artifacts
 - 3. Workflows
 - 4. Checkpoints
 - B. Major milestone scope and content
 - C. Process improvement procedures
 - III. Software engineering environment**
 - A. Process automation (hardware and software resource configuration)
 - B. Resource allocation procedures (sharing across organizations, security access)
 - IV. Software change management**
 - A. Configuration control board plan and procedures
 - B. Software change order definitions and procedures
 - C. Configuration baseline definitions and procedures
 - V. Software assessment**
 - A. Metrics collection and reporting procedures
 - B. Risk management procedures (risk identification, tracking, and resolution)
 - C. Status assessment plan
 - D. Acceptance test plan
 - VI. Standards and procedures**
 - A. Standards and procedures for technical artifacts
 - VII. Evolutionary appendixes**
 - A. Minor milestone scope and content
 - B. Human resources (organization, staffing plan, training plan)

FIGURE 6-5. *Typical software development plan outline*

Work Breakdown Structure

Work breakdown structure (WBS) is the vehicle for budgeting and collecting costs. To monitor and control a project's financial performance, the software project manager must have insight into project costs and how they are expended. The structure of cost accountability is a serious project planning constraint.

Software Change Order Database

Managing change is one of the fundamental primitives of an iterative development process. With greater change freedom, a project can iterate more productively. This flexibility increases the content, quality, and number of iterations that a project can achieve within a given schedule. Change freedom has been achieved in practice through automation, and today's iterative development environments carry the burden of change management. Organizational processes that depend on manual change management techniques have encountered major inefficiencies.

Release Specifications

The scope, plan, and objective evaluation criteria for each baseline release are derived from the vision statement as well as many other sources (make/buy analyses, risk management concerns, architectural considerations, shots in the dark, implementation constraints, quality thresholds). These artifacts are intended to evolve along with the process, achieving greater fidelity as the life cycle progresses and requirements understanding matures. Figure 6-6 provides a default outline for a release specification

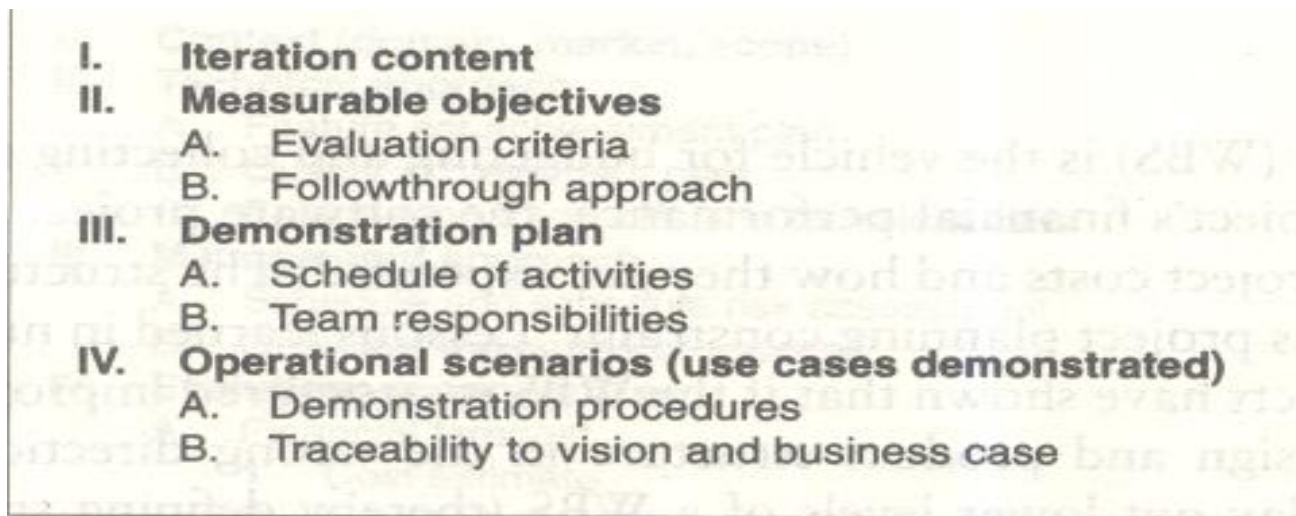
- 
- I. Iteration content**
 - II. Measurable objectives**
 - A. Evaluation criteria
 - B. Followthrough approach
 - III. Demonstration plan**
 - A. Schedule of activities
 - B. Team responsibilities
 - IV. Operational scenarios (use cases demonstrated)**
 - A. Demonstration procedures
 - B. Traceability to vision and business case

FIGURE 6-6. *Typical release specification outline*

Release Descriptions

Release description documents describe the results of each release, including performance against each of the evaluation criteria in the corresponding release specification. Release baselines should be accompanied by a release description document that describes the evaluation criteria for that configuration baseline and provides substantiation (through demonstration, testing, inspection, or analysis) that each criterion has been addressed in an acceptable manner. Figure 6-7 provides a default outline for a release description.

Status Assessments

Status assessments provide periodic snapshots of project health and status, including the software project manager's risk assessment, quality indicators, and management indicators. Typical status assessments should include a review of resources, personnel staffing, financial data (cost and revenue), top 10 risks, technical progress (metrics snapshots), major milestone plans and results, total project or product scope & action items

-
- I. Context**
 - A. Release baseline content
 - B. Release metrics
 - II. Release notes**
 - A. Release-specific constraints or limitations
 - III. Assessment results**
 - A. Substantiation of passed evaluation criteria
 - B. Follow-up plans for failed evaluation criteria
 - C. Recommendations for next release
 - IV. Outstanding issues**
 - A. Action items
 - B. Post-mortem summary of lessons learned

FIGURE 6-7. *Typical release description outline*

Environment

An important emphasis of a modern approach is to define the development and maintenance environment as a first-class artifact of the process. A robust, integrated development environment must support automation of the development process. This environment should include requirements management, visual modeling, document automation, host and target programming tools, automated regression testing, and continuous and integrated change management, and feature and defect tracking.

Deployment

A deployment document can take many forms. Depending on the project, it could include several document subsets for transitioning the product into operational status. In big contractual efforts in which the system is delivered to a separate maintenance organization, deployment artifacts may include computer system operations manuals, software installation manuals, plans and procedures for cutover (from a legacy system), site surveys, and so forth. For commercial software products, deployment artifacts may include marketing plans, sales rollout kits, and training courses.

Management Artifact Sequences

In each phase of the life cycle, new artifacts are produced and previously developed artifacts are updated to incorporate lessons learned and to capture further depth and breadth of the solution. Figure 6-8 identifies a typical sequence of artifacts across the life-cycle phases.

△ Informal version

▲ Controlled baseline

Inception	Elaboration		Construction			Transition
Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5	Iteration 6	Iteration 7

Management Set

1. Work breakdown structure		▲		▲		▲
2. Business case	▲	▲	▲			▲
3. Release specifications	△	▲	▲	▲	▲	▲
4. Software development plan	▲		▲			
5. Release descriptions	△	△	▲	▲	▲	▲
6. Status assessments	△	△	△	△	△	△
7. Software change order data				▲	▲	▲
8. Deployment documents			△		△	▲
9. Environment	△		▲		▲	

Requirements Set

1. Vision document	▲		▲			▲
2. Requirements model(s)	▲		▲			▲

Design Set

1. Design model(s)	△		▲			▲
2. Test model	△		▲			▲
3. Architecture description	△		▲			▲

Implementation Set

1. Source code baselines			▲	▲	▲	▲
2. Associated compile-time files			▲	▲	▲	▲
3. Component executables			▲	▲	▲	▲

Deployment Set

1. Integrated product-executable baselines			▲	▲	▲	▲
2. Associated run-time files			▲	▲	▲	▲
3. User manual			△		▲	

FIGURE 6-8. Artifact sequences across a typical life cycle

6.3 ENGINEERING ARTIFACTS

Most of the engineering artifacts are captured in rigorous engineering notations such as UML, programming languages, or executable machine codes. Three engineering artifacts are explicitly intended for more general review, and they deserve further elaboration.

Vision Document

The vision document provides a complete vision for the software system under development and supports the contract between the funding authority and the development organization. A project vision is meant to be changeable as understanding evolves of the requirements, architecture, plans, and technology. A good vision document should change slowly. Figure 6-9 provides a default outline for a vision document.

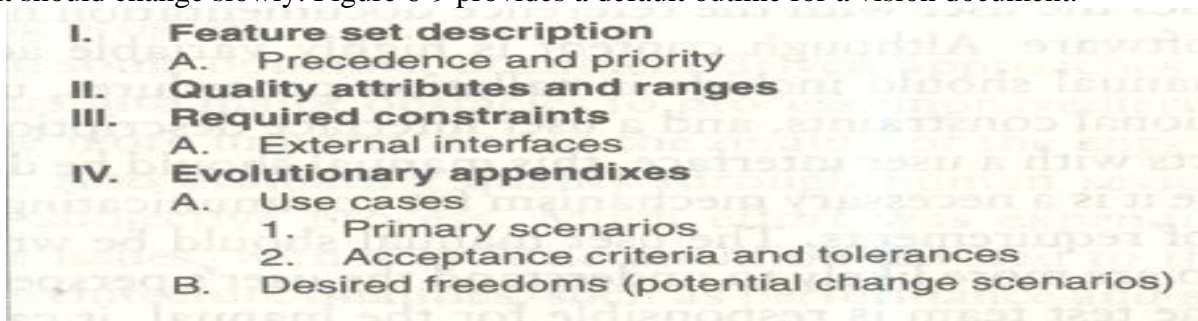
- 
- I. Feature set description**
 - A. Precedence and priority
 - II. Quality attributes and ranges**
 - III. Required constraints**
 - A. External interfaces
 - IV. Evolutionary appendixes**
 - A. Use cases
 - 1. Primary scenarios
 - 2. Acceptance criteria and tolerances
 - B. Desired freedoms (potential change scenarios)

FIGURE 6-9. Typical vision document outline

Architecture Description

The architecture description provides an organized view of the software architecture under development. It is extracted largely from the design model and includes views of the design, implementation, and deployment sets sufficient to understand how the operational concept of the requirements set will be achieved. The breadth of the architecture description will vary from project to project depending on many factors. Figure 6-10 provides a default outline for an architecture description.

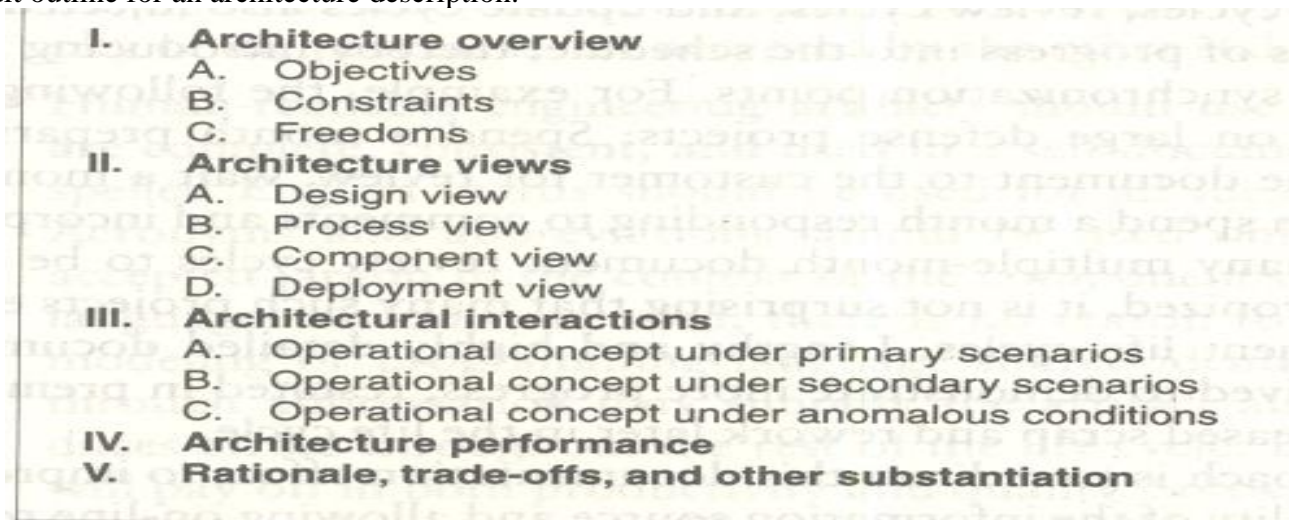
- 
- I. Architecture overview**
 - A. Objectives
 - B. Constraints
 - C. Freedoms
 - II. Architecture views**
 - A. Design view
 - B. Process view
 - C. Component view
 - D. Deployment view
 - III. Architectural interactions**
 - A. Operational concept under primary scenarios
 - B. Operational concept under secondary scenarios
 - C. Operational concept under anomalous conditions
 - IV. Architecture performance**
 - V. Rationale, trade-offs, and other substantiation**

FIGURE 6-10. Typical architecture description outline

Software User Manual

The software user manual provides the user with the reference documentation necessary to support the delivered software. Although content is highly variable across application domains, the user manual should include installation procedures, usage procedures and guidance, operational constraints, and a user interface description, at a minimum. For software products with a user interface, this manual should be developed early in the life cycle because it is a necessary mechanism for communicating and stabilizing an important subset of requirements. The user manual should be written by members of the test team, who are more likely to understand the user's perspective than the development team.

6.4 PRAGMATIC ARTIFACTS

- **People want to review information but don't understand the language of the artifact.** Many interested reviewers of a particular artifact will resist having to learn the engineering language in which the artifact is written. It is not uncommon to find people (such as veteran software managers, veteran quality assurance specialists, or an auditing authority from a regulatory agency) who react as follows: "I'm not going to learn UML, but I want to review the design of this software, so give me a separate description such as some flowcharts and text that I can understand."

- **People want to review the information but don't have access to the tools.** It is not very common for the development organization to be fully tooled; it is extremely rare that the/other stakeholders have any capability to review the engineering artifacts on-line. Consequently, organizations are forced to exchange paper documents. Standardized formats (such as UML, spreadsheets, Visual Basic, C++, and Ada 95), visualization tools, and the Web are rapidly making it economically feasible for all stakeholders to exchange information electronically.

- **Human-readable engineering artifacts should use rigorous notations that are complete, consistent, and used in a self-documenting manner.** Properly spelled English words should be used for all identifiers and descriptions. Acronyms and abbreviations should be used only where they are well accepted jargon in the context of the component's usage. Readability should be emphasized and the use of proper English words should be required in all engineering artifacts. This practice enables understandable representations, browse able formats (paperless review), more-rigorous notations, and reduced error rates.

- **Useful documentation is self-defining: It is documentation that gets used.**

- **Paper is tangible; electronic artifacts are too easy to change.** On-line and Web-based artifacts can be changed easily and are viewed with more skepticism because of their inherent volatility.

Unit – III Important questions

1.	Explain briefly two stages of the life cycle engineering and production.
2.	Explain different phases of the life cycle process?
3.	Explain the goal of Inception phase, Elaboration phase, Construction phase and Transition phase.
4.	Explain the overview of the artifact set
5.	Write a short note on (a) Management Artifacts (b) Engineering Artifacts (c) Pragmatic Artifacts

7. Model based software architecture

7.1 ARCHITECTURE: A MANAGEMENT PERSPECTIVE

The most critical technical product of a software project is its architecture: the infrastructure, control, and data interfaces that permit software components to cooperate as a system and software designers to cooperate efficiently as a team. When the communications media include multiple languages and intergroup literacy varies, the communications problem can become extremely complex and even unsolvable. If a software development team is to be successful, the inter project communications, as captured in the software architecture, must be both accurate and precise

From a management perspective, there are three different aspects of architecture.

1. An *architecture* (the intangible design concept) is the design of a software system this includes all engineering necessary to specify a complete bill of materials.
2. An *architecture baseline* (the tangible artifacts) is a slice of information across the engineering artifact sets sufficient to satisfy all stakeholders that the vision (function and quality) can be achieved within the parameters of the business case (cost, profit, time, technology, and people).
3. An *architecture description* (a human-readable representation of an architecture, which is one of the components of an architecture baseline) is an organized subset of information extracted from the design set model(s). The architecture description communicates how the intangible concept is realized in the tangible artifacts.

The number of views and the level of detail in each view can vary widely.

The importance of software architecture and its close linkage with modern software development processes can be summarized as follows:

- Achieving a stable software architecture represents a significant project milestone at which the critical make/buy decisions should have been resolved.
- Architecture representations provide a basis for balancing the trade-offs between the problem space (requirements and constraints) and the solution space (the operational product).
- The architecture and process encapsulate many of the important (high-payoff or high-risk) communications among individuals, teams, organizations, and stakeholders.
- Poor architectures and immature processes are often given as reasons for project failures.
- A mature process, an understanding of the primary requirements, and a demonstrable architecture are important prerequisites for predictable planning.
- Architecture development and process definition are the intellectual steps that map the problem to a solution without violating the constraints; they require human innovation and cannot be automated.

7.2 ARCHITECTURE: A TECHNICAL PERSPECTIVE

An architecture framework is defined in terms of views that are abstractions of the UML models in the design set. The design model includes the full breadth and depth of information. An architecture view is an abstraction of the design model; it contains only the architecturally significant information. Most real-world systems require four views: design, process, component, and deployment. The purposes of these views are as follows:

- Design: describes architecturally significant structures and functions of the design model
- Process: describes concurrency and control thread relationships among the design, component, and deployment views
- Component: describes the structure of the implementation set
- Deployment: describes the structure of the deployment set

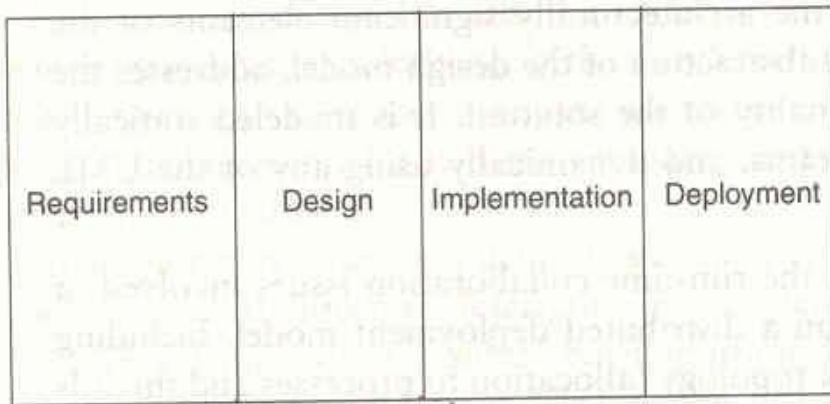
Figure 7-1 summarizes the artifacts of the design set, including the architecture views and architecture description.

The requirements model addresses the behavior of the system as seen by its end users, analysts, and testers. This view is modeled statically using use case and class diagrams, and dynamically using sequence, collaboration, state chart, and activity diagrams.

- The *use case view* describes how the system's critical (architecturally significant) use cases are realized by elements of the design model. It is modeled statically using use case diagrams, and dynamically using any of the UML behavioral diagrams.
- The *design view* describes the architecturally significant elements of the design model. This view, an abstraction of the design model, addresses the basic structure and functionality of the solution. It is modeled statically using class and object diagrams, and dynamically using any of the UML behavioral diagrams.
- The *process view* addresses the run-time collaboration issues involved in executing the architecture on a distributed deployment model, including the logical software network topology (allocation to processes and threads of control), interprocess communication, and state management. This view is modeled statically using deployment diagrams, and dynamically using any of the UML behavioral diagrams.
- The *component view* describes the architecturally significant elements of the implementation set. This view, an abstraction of the design model, addresses the software source code realization of the system from the perspective of the project's integrators and developers, especially with regard to releases and configuration management. It is modeled statically using component diagrams, and dynamically using any of the UML behavioral diagrams.
- The *deployment view* addresses the executable realization of the system, including the allocation of logical processes in the distribution view (the logical software topology) to physical resources of the deployment network (the physical system topology). It is modeled statically using deployment diagrams, and dynamically using any of the UML behavioral diagrams.

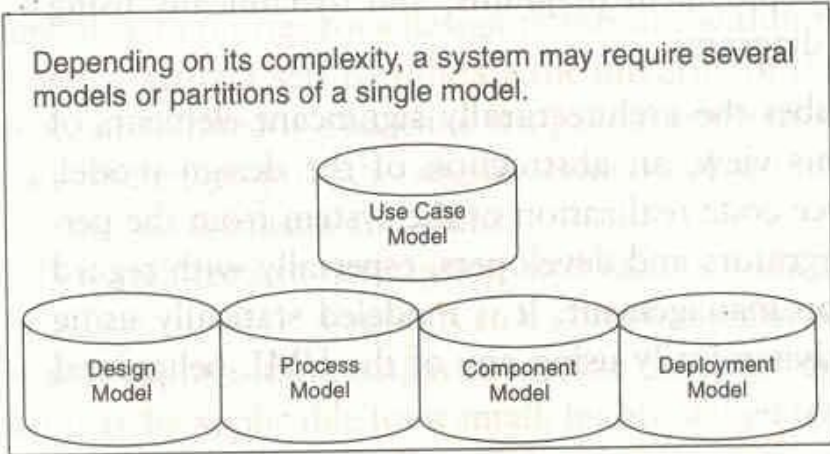
Generally, an architecture baseline should include the following:

- Requirements: critical use cases, system-level quality objectives, and priority relationships among features and qualities
- Design: names, attributes, structures, behaviors, groupings, and relationships of significant classes and components
- Implementation: source component inventory and bill of materials (number, name, purpose, cost) of all primitive components
- Deployment: executable components sufficient to demonstrate the critical use cases and the risk associated with achieving the system qualities



The requirements set may include UML models describing the problem space.

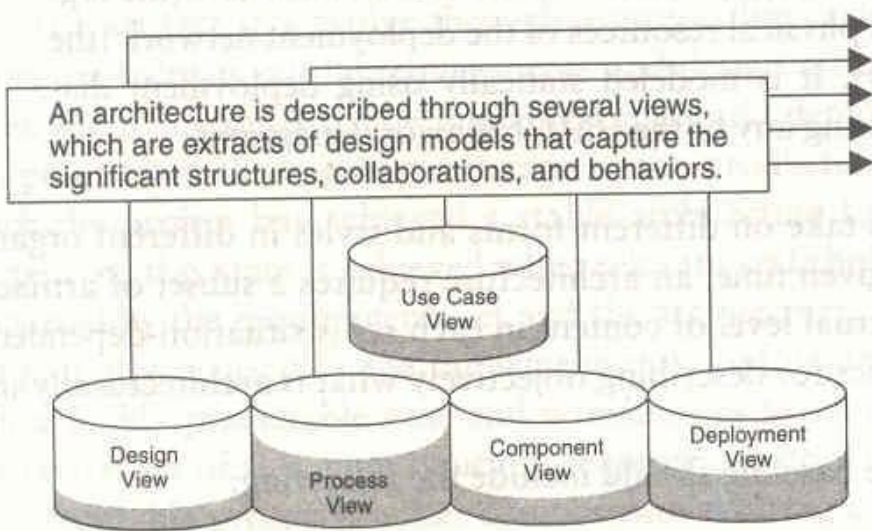
The design set includes all UML design models describing the solution space.



The *design, process, and use case models* provide for visualization of the logical and behavioral aspects of the design.

The *component model* provides for visualization of the implementation set.

The *deployment model* provides for visualization of the deployment set.



Architecture Description Document

- Design view
- Process view
- Use case view
- Component view
- Deployment view
- Other views (optional)
- Other material:
 - Rationale
 - Constraints

FIGURE 7-1. *Architecture, an organized and abstracted view into the design models*

UNIT - IV

Work Flows of the process: Software process workflows, Iteration workflows.

Checkpoints of the process: Major mile stones, Minor Milestones, Periodic status assessments.

Iterative Process Planning: Work breakdown structures, planning guidelines, cost and schedule estimating, Iteration planning process, Pragmatic planning

Workflow of the process

SOFTWARE PROCESS WORKFLOWS

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

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

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

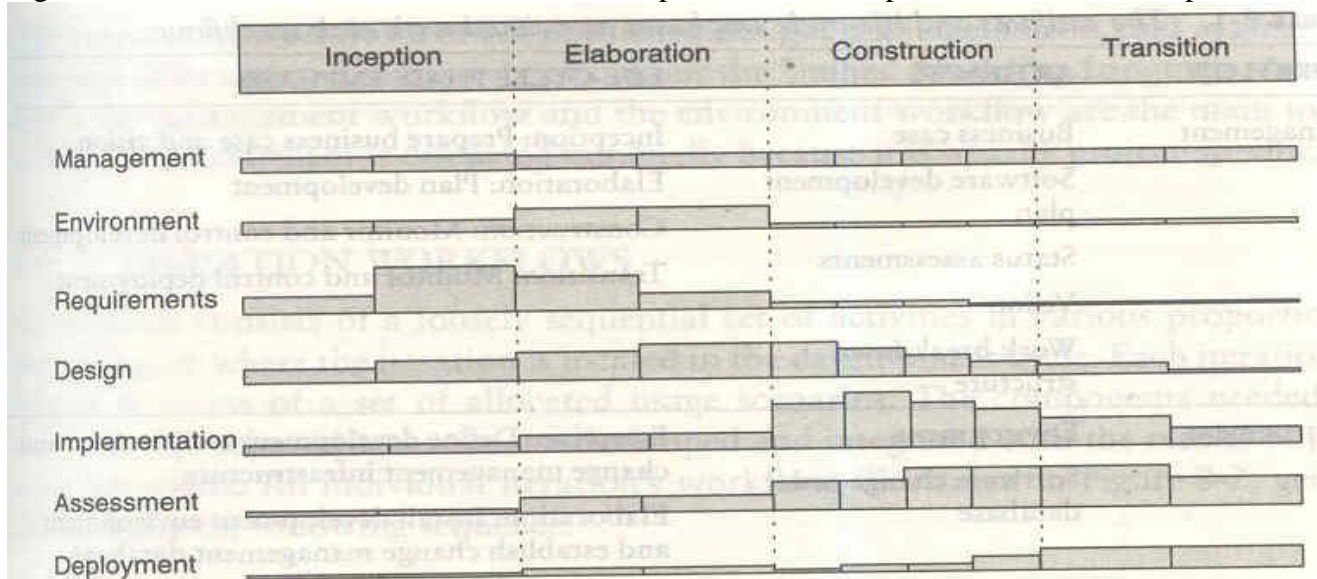


FIGURE 8-1. Activity levels across the life-cycle phases

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

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

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

ITERATION WORKFLOWS

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

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

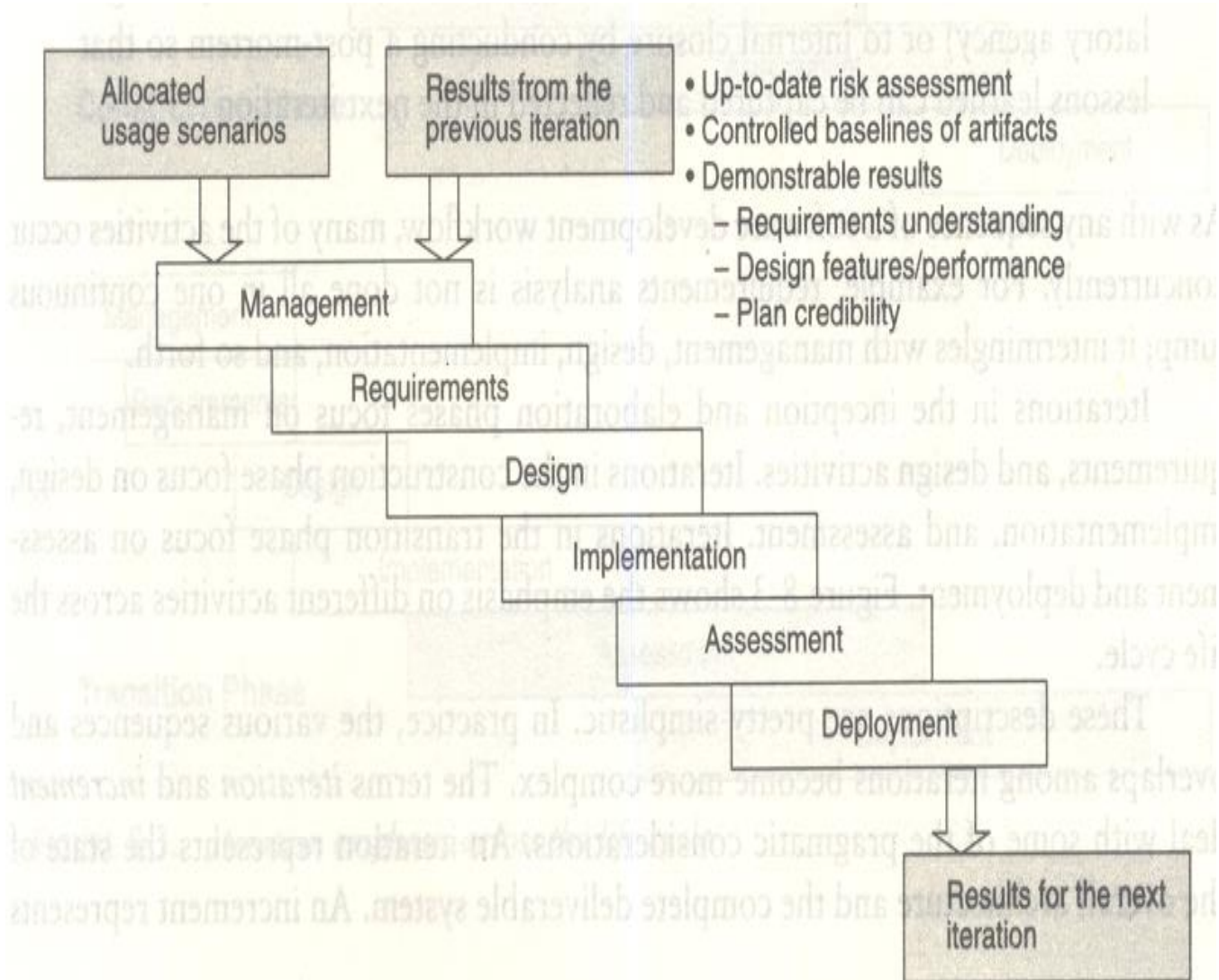


FIGURE 8-2. *The workflow of an iteration*

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

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

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

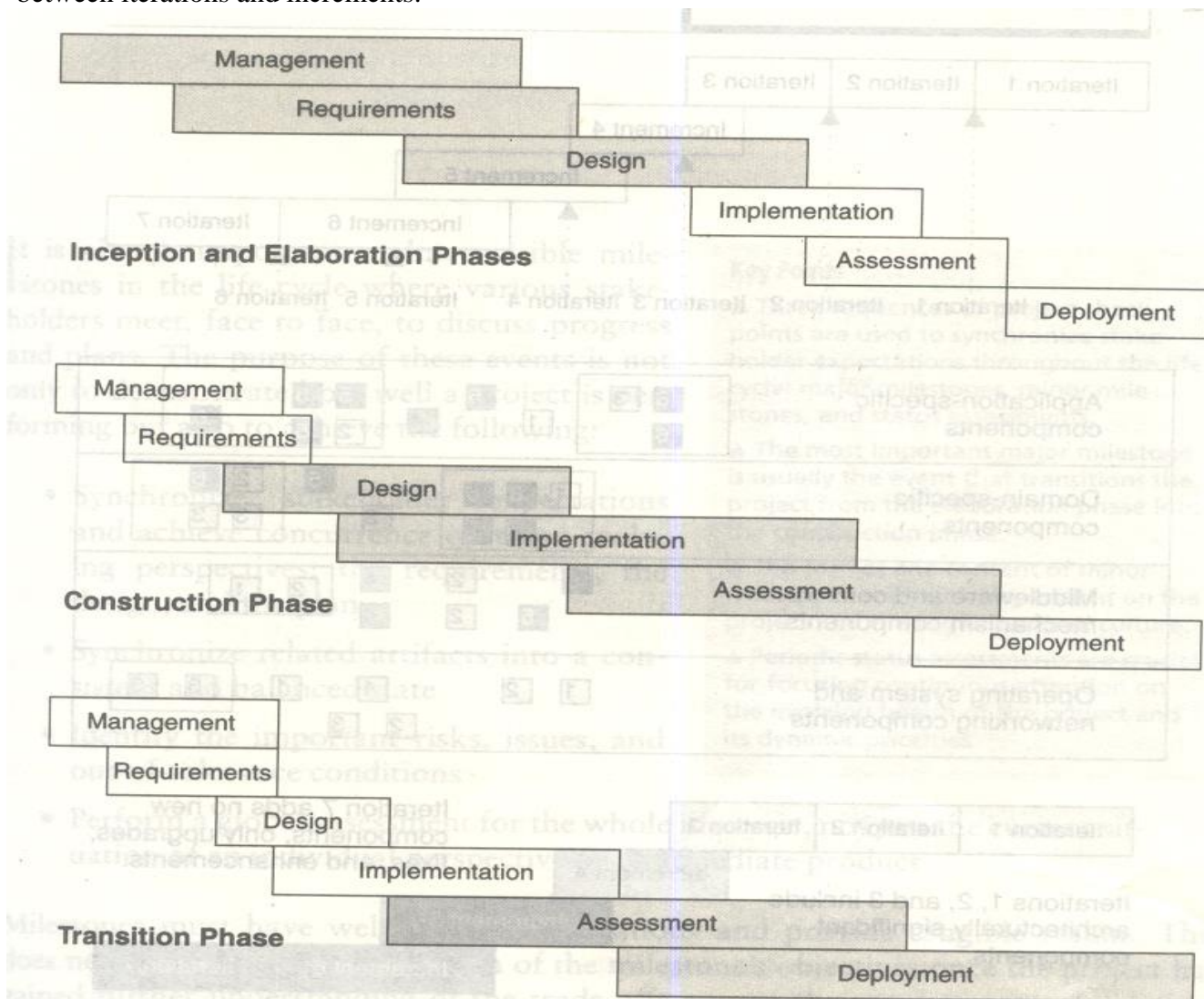


FIGURE 8-3. Iteration emphasis across the life cycle

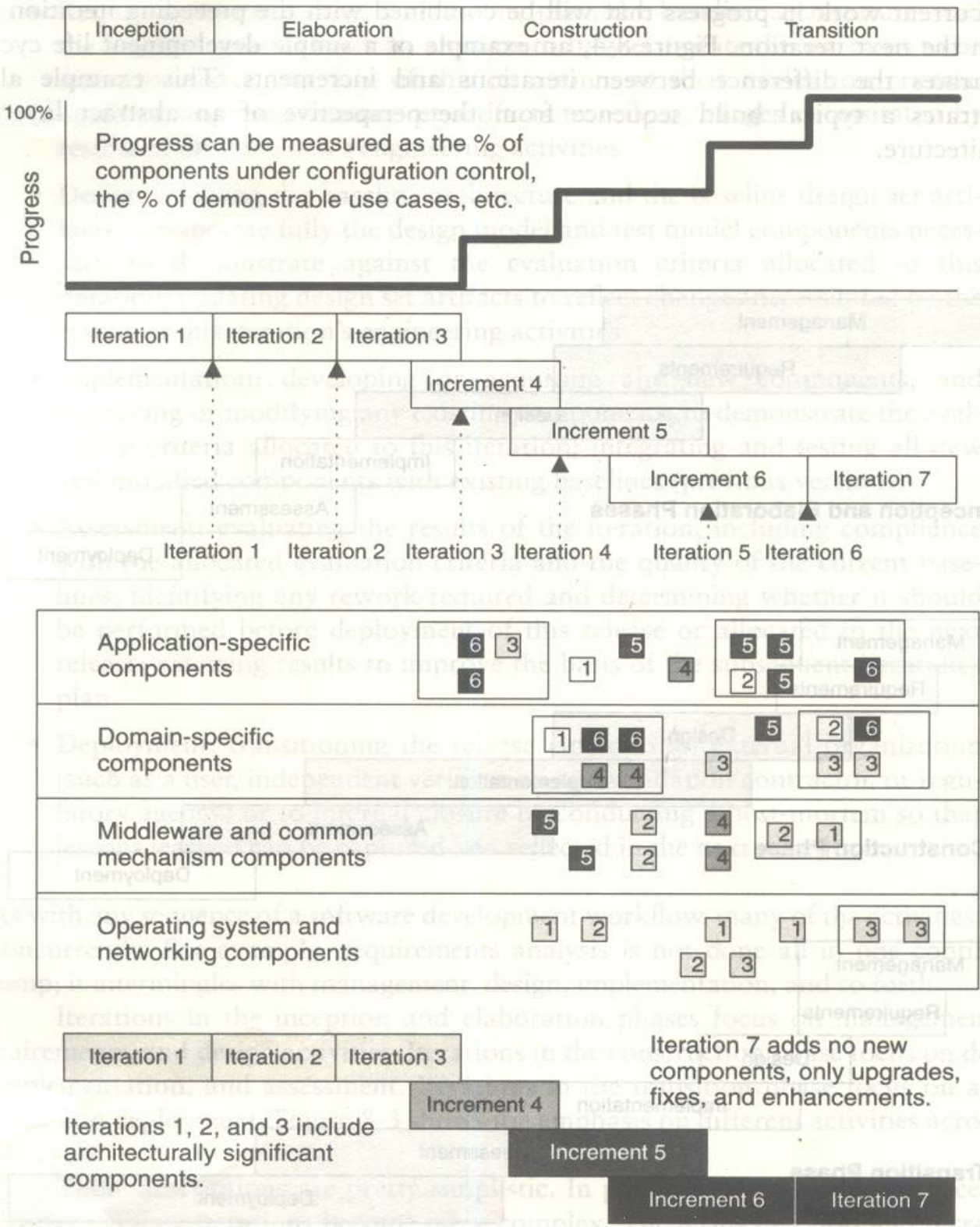


FIGURE 8-4. A typical build sequence associated with a layered architecture

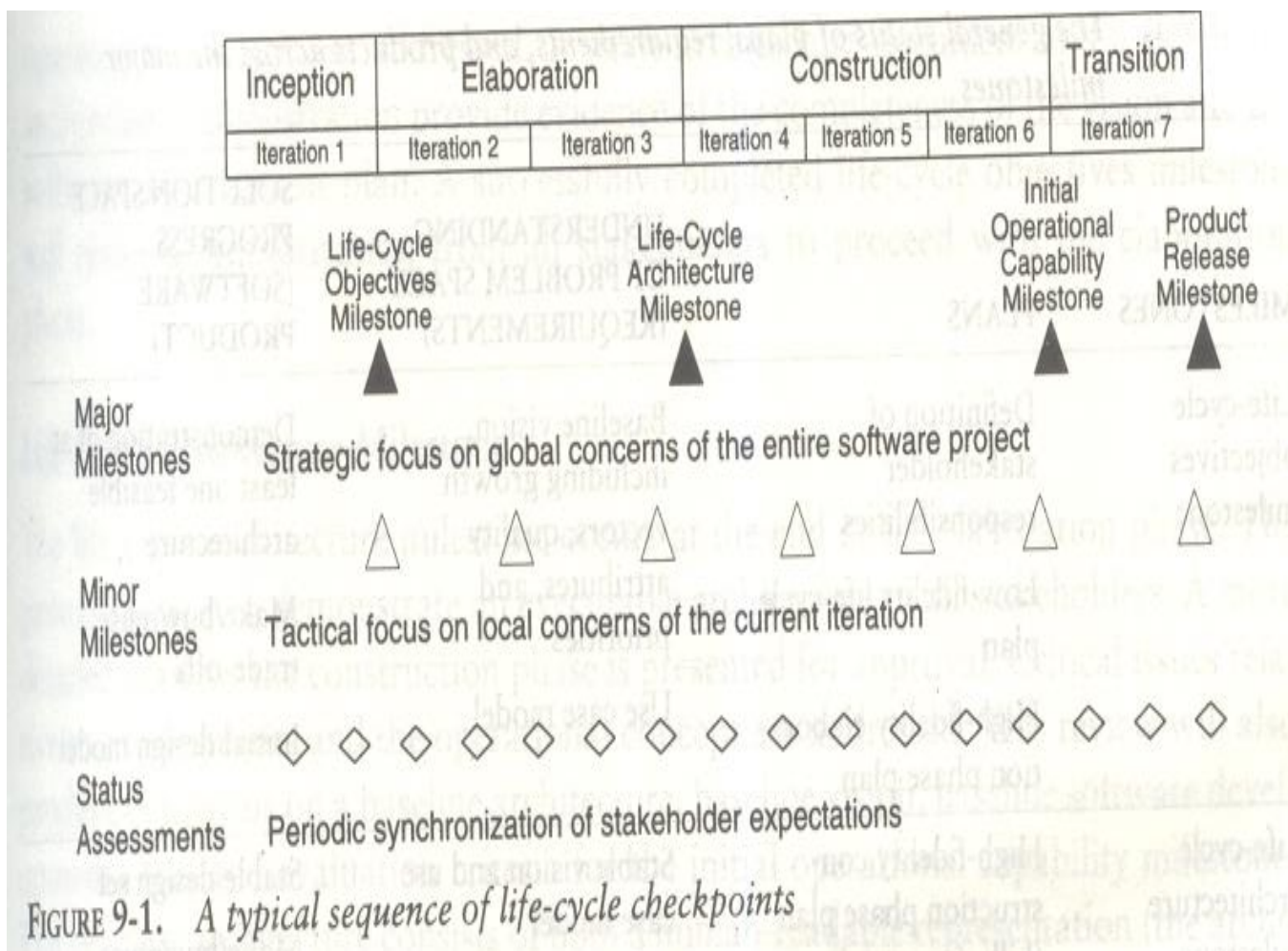
9. Checkpoints of the process

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

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

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

Figure 9-1 illustrates a typical sequence of project checkpoints for a relatively large project.



9.1 MAJOR MILESTONES

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

- Customers: schedule and budget estimates, feasibility, risk assessment, requirements understanding, progress, product line compatibility
- Users: consistency with requirements and usage scenarios, potential for accommodating growth, quality attributes
- Architects and systems engineers: product line compatibility, requirements changes, trade-off analyses, completeness and consistency, balance among risk, quality, and usability
- Developers: sufficiency of requirements detail and usage scenario descriptions, frameworks for component selection or development, resolution of development risk, product line compatibility, sufficiency of the development environment
- Maintainers: sufficiency of product and documentation artifacts, understandability, interoperability with existing systems, sufficiency of maintenance environment
- Others: possibly many other perspectives by stakeholders such as regulatory agencies, independent verification and validation contractors, venture capital investors, subcontractors, associate contractors, and sales and marketing teams

Table 9-1 summarizes the balance of information across the major milestones.

TABLE 9-1. *The general status of plans, requirements, and products across the major milestones*

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

Life-Cycle Objectives Milestone

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

Life-Cycle Architecture Milestone

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

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

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

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

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

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

Initial Operational Capability Milestone

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

Product Release Milestone

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

9.2 MINOR MILESTONES

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

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

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

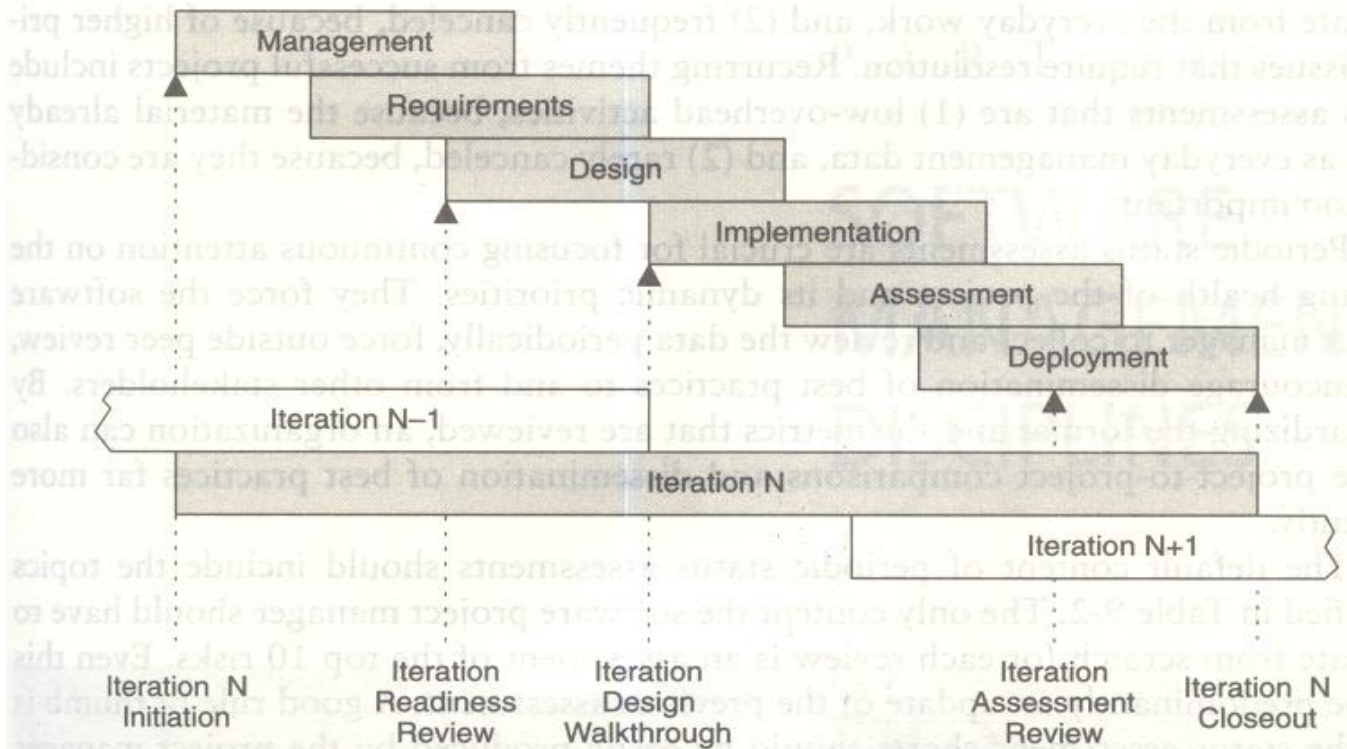


FIGURE 9-4. Typical minor milestones in the life cycle of an iteration

9.3 PERIODIC STATUS ASSESSMENTS

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

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

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

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

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

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

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

10. Iterative process planning

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

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

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

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

10.1.1 CONVENTIONAL WBS ISSUES

Conventional work breakdown structures frequently suffer from three fundamental flaws.

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

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

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

Conventional work breakdown structures are project-specific, and cross-project comparisons are usually difficult or impossible. With no standard WBS structure, it is extremely difficult to compare plans, financial data, schedule data, organizational efficiencies, cost trends, productivity trends, or quality trends across multiple projects.

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

Management
 System requirement and design
 Subsystem 1
 Component 11
 Requirements
 Design
 Code
 Test
 Documentation
 ...(similar structures for other components)
 Component 1N
 Requirements
 Design
 Code
 Test
 Documentation
 ...(similar structures for other subsystems)
 Subsystem M
 Component M1

Requirements
Design
Code
Test
Documentation
...(similar structures for other components)
Component MN
Requirements
Design
Code
Test
Documentation
Integration and test
Test planning
Test procedure preparation
Testing
Test reports
Other support areas
Configuration control
Quality assurance
System administration

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

Figure 10-2 Default work breakdown structure

A Management

AA Inception phase management

- AAA Business case development**
- AAB Elaboration phase release specifications**
- AAC Elaboration phase WBS specifications**
- AAD Software development plan**
- AAE Inception phase project control and status assessments**

AB Elaboration phase management

- ABA Construction phase release specifications**
- ABB Construction phase WBS baselining**
- ABC Elaboration phase project control and status assessments**

AC Construction phase management

- ACA Deployment phase planning**
- ACB Deployment phase WBS baselining**
- ACC Construction phase project control and status assessments**

AD Transition phase management

- ADA Next generation planning**
- ADB Transition phase project control and status assessments**

B Environment

BA Inception phase environment specification

BB Elaboration phase environment baselining

- BBA Development environment installation and administration**
- BBB Development environment integration and custom toolsmithing**
- BBC SCO database formulation**

BC Construction phase environment maintenance

- BCA Development environment installation and administration**
- BCB SCO database maintenance**

BD Transition phase environment maintenance

- BDA Development environment maintenance and administration**
- BDB SCO database maintenance**
- BDC Maintenance environment packaging and transition**

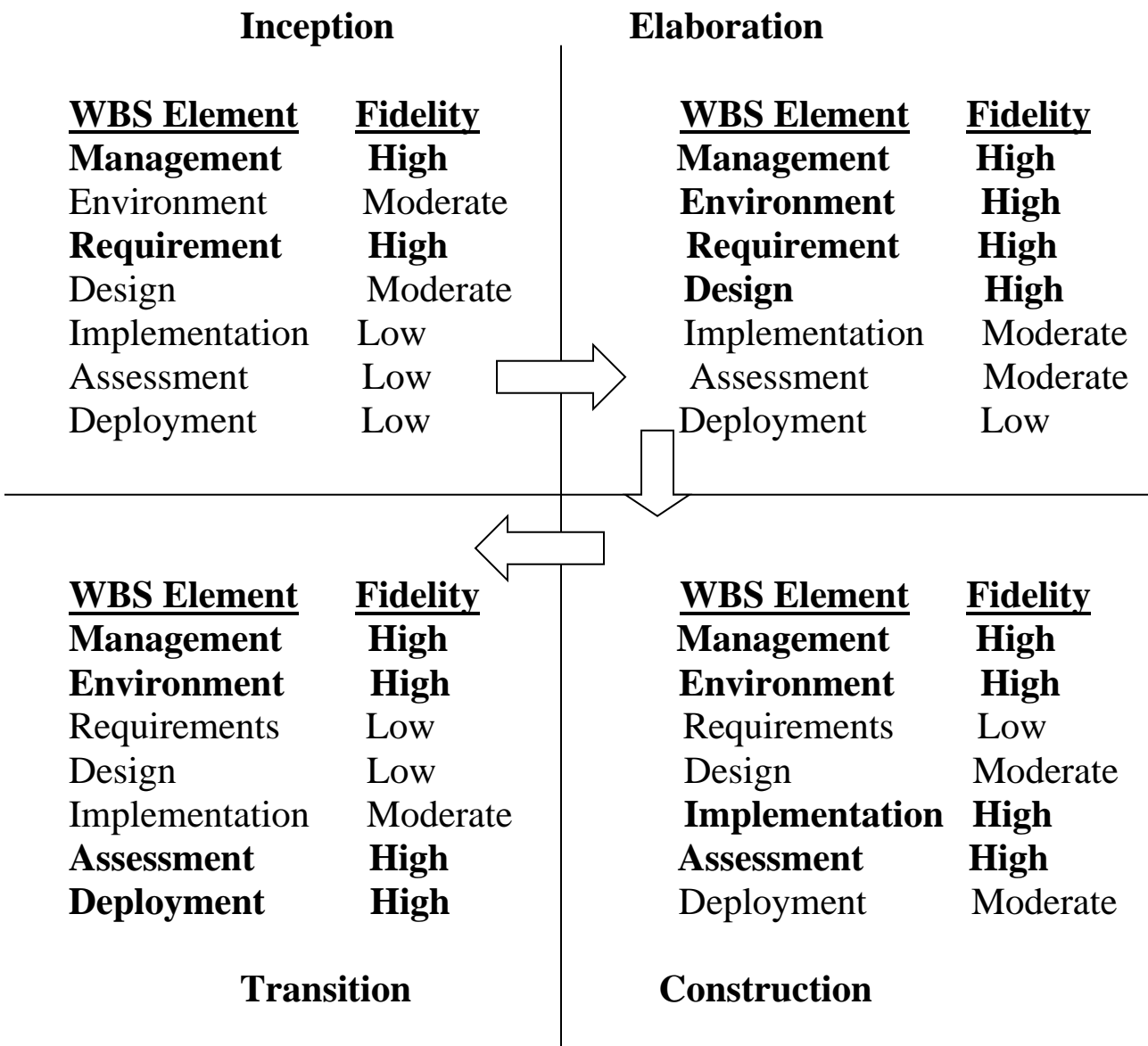
C Requirements

CA Inception phase requirements development

- CCA Vision specification**
- CAB Use case modeling**

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

Figure 10-3 Evolution of planning fidelity in the WBS over the life cycle



10.2 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 circumstances. Two simple planning guidelines should be considered when a project plan is being initiated or assessed. The first guideline, detailed in Table 10-1, prescribes a default allocation of costs among the first-level WBS elements. The second guideline, detailed in Table 10-2, prescribes the allocation of effort and schedule across the lifecycle phases.

10-1 Web budgeting defaults

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

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

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

10.3 THE COST AND SCHEDULE ESTIMATING PROCESS

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

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

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

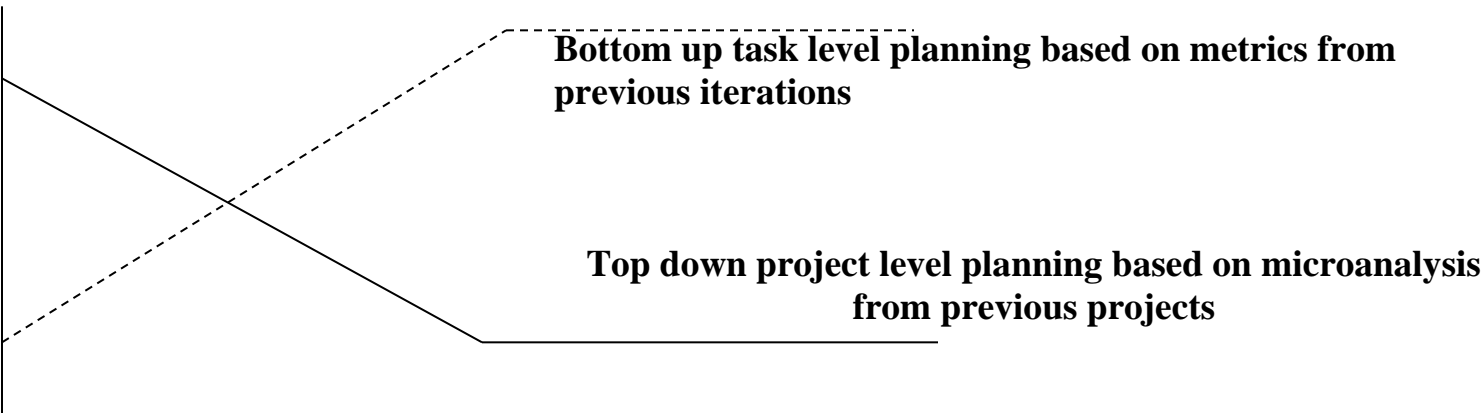
1. The lowest level WBS elements are elaborated into detailed tasks
2. Estimates are combined and integrated into higher level budgets and milestones.
3. Comparisons are made with the top-down budgets and schedule milestones.

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

These two planning approaches should be used together, in balance, throughout the life cycle of the project. During the engineering stage, the top-down perspective will dominate because there is usually not enough depth of understanding nor stability in the detailed task sequences to perform credible bottom-up planning. During the production stage, there should be enough precedent experience and planning fidelity that the bottom-up planning perspective will dominate. Top-down approach should be well tuned to the project-

specific parameters, so it should be used more as a global assessment technique. Figure 10-4 illustrates this life-cycle planning balance.

Figure 10-4 Planning balance throughout the life cycle



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

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

10.4 THE ITERATION PLANNING PROCESS

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

- Inception iterations. The early prototyping activities integrate the foundation components of a candidate architecture and provide an executable framework for elaborating the critical use cases of the system. This framework includes existing components, commercial components, and custom prototypes sufficient to demonstrate a candidate architecture and sufficient requirements understanding to establish a credible business case, vision, and software development plan.
- Elaboration iterations. These iterations result in architecture, including a complete framework and infrastructure for execution. Upon completion of the architecture iteration, a few critical use cases should

be demonstrable: (1) initializing the architecture, (2) injecting a scenario to drive the worst-case data processing flow through the system (for example, the peak transaction throughput or peak load scenario), and (3) injecting a scenario to drive the worst-case control flow through the system (for example, orchestrating the fault-tolerance use cases).

- Construction iterations. Most projects require at least two major construction iterations: an alpha release and a beta release.
- Transition iterations. Most projects use a single iteration to transition a beta release into the final product.

The general guideline is that most projects will use between four and nine iterations. The typical project would have the following six-iteration profile:

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

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

10.5 PRAGMATIC PLANNING

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

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

Unit – Important Questions

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

UNIT - V

Project Organizations and Responsibilities: Line-of-Business Organizations, Project Organizations, evolution of Organizations.

Process Automation: Automation Building blocks, The Project Environment.

Project Organizations and Responsibilities:

- **Organizations** engaged in software Line-of-Business need to support projects with the infrastructure necessary to use a common process.
- **Project** organizations need to allocate artifacts & responsibilities across project team to ensure a balance of global (architecture) & local (component) concerns.
- **The organization** must evolve with the WBS & Life cycle concerns.
- **Software lines of business & product teams have different motivation.**
- **Software lines of business** are motivated by return of investment (ROI), new business discriminators, market diversification & profitability.
- **Project teams** are motivated by the cost, Schedule & quality of specific deliverables

1) Line-Of-Business Organizations:

The main features of default organization are as follows:

- Responsibility for process definition & maintenance is specific to a cohesive line of business.
- Responsibility for process automation is an organizational role & is equal in importance to the process definition role.
- Organizational role may be fulfilled by a single individual or several different teams.

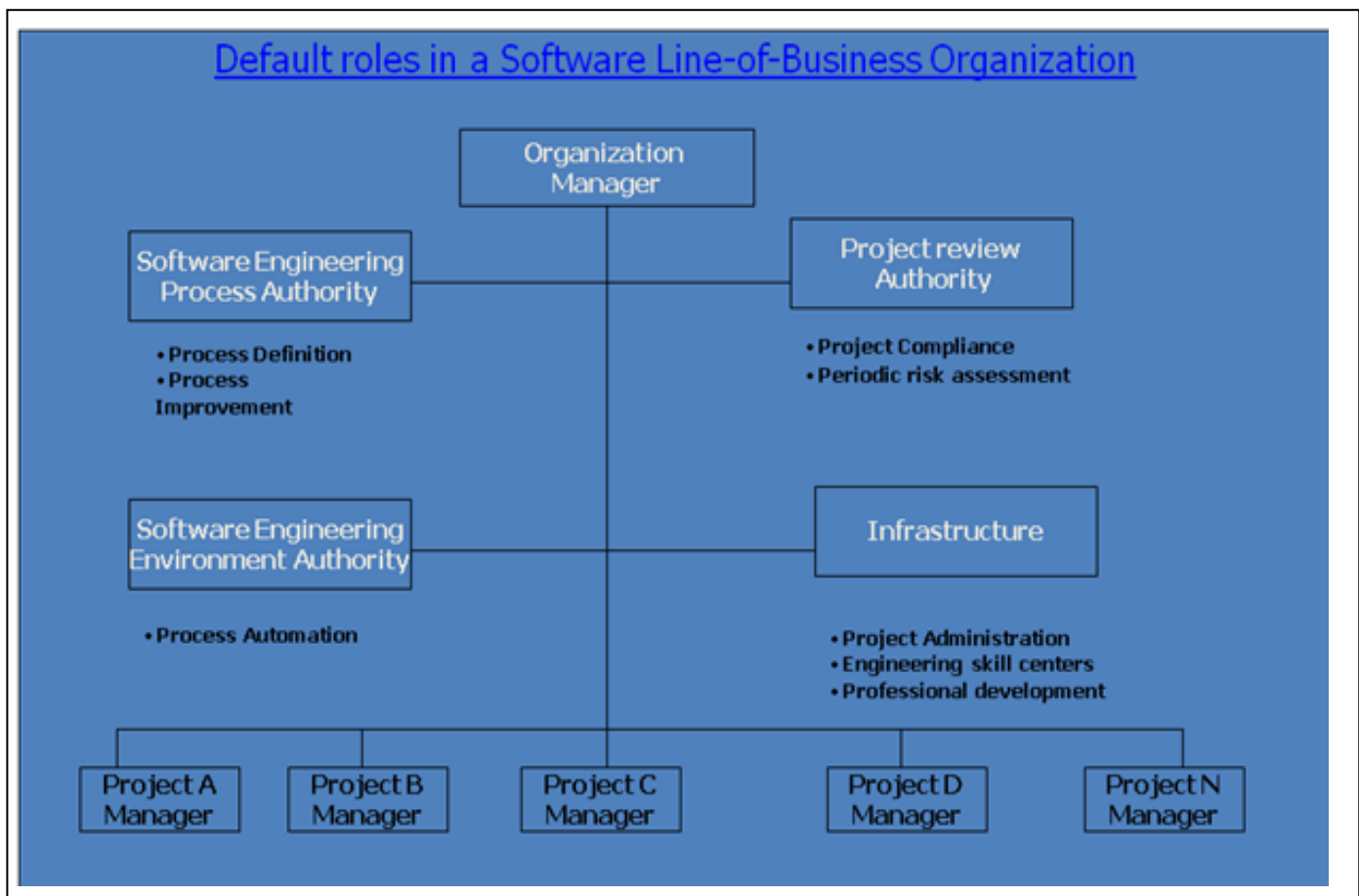


Fig: Default roles in a software Line-of-Business Organization.

Software Engineering Process Authority (SEPA)

The SEPA facilitates the exchange of information & process guidance both to & from project practitioners

This role is accountable to General Manager for maintaining a current assessment of the organization's process maturity & its plan for future improvement

Project Review Authority (PRA)

The PRA is the single individual responsible for ensuring that a software project complies with all organizational & business unit software policies, practices & standards

A software Project Manager is responsible for meeting the requirements of a contract or some other project compliance standard

Software Engineering Environment Authority(SEEA)

The SEEA is responsible for automating the organization's process, maintaining the organization's standard environment, Training projects to use the environment & maintaining organization-wide reusable assets

The SEEA role is necessary to achieve a significant ROI for common process.

Infrastructure

An organization's infrastructure provides human resources support, project-independent research & development, & other capital software engineering assets.

2) Project organizations:

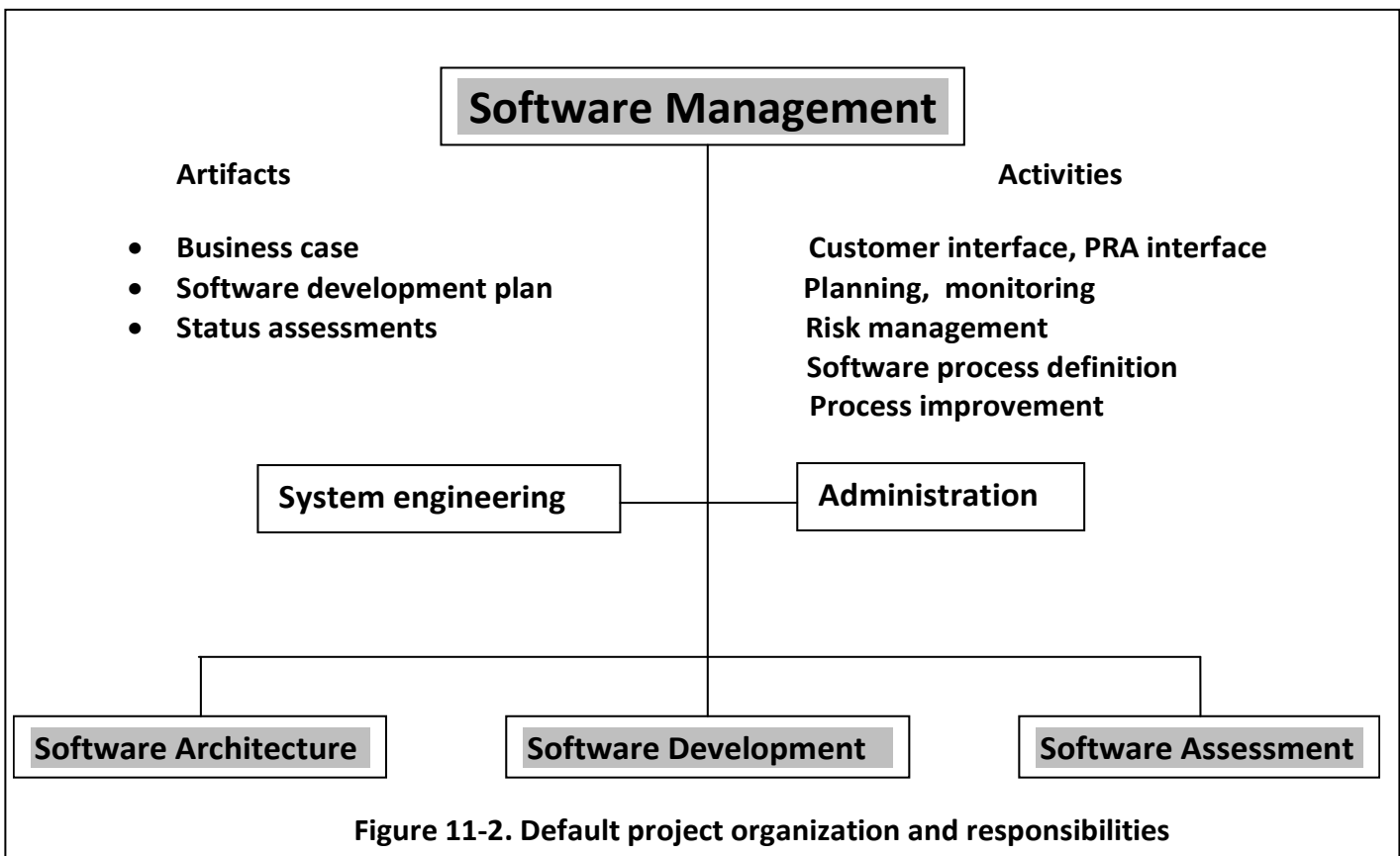
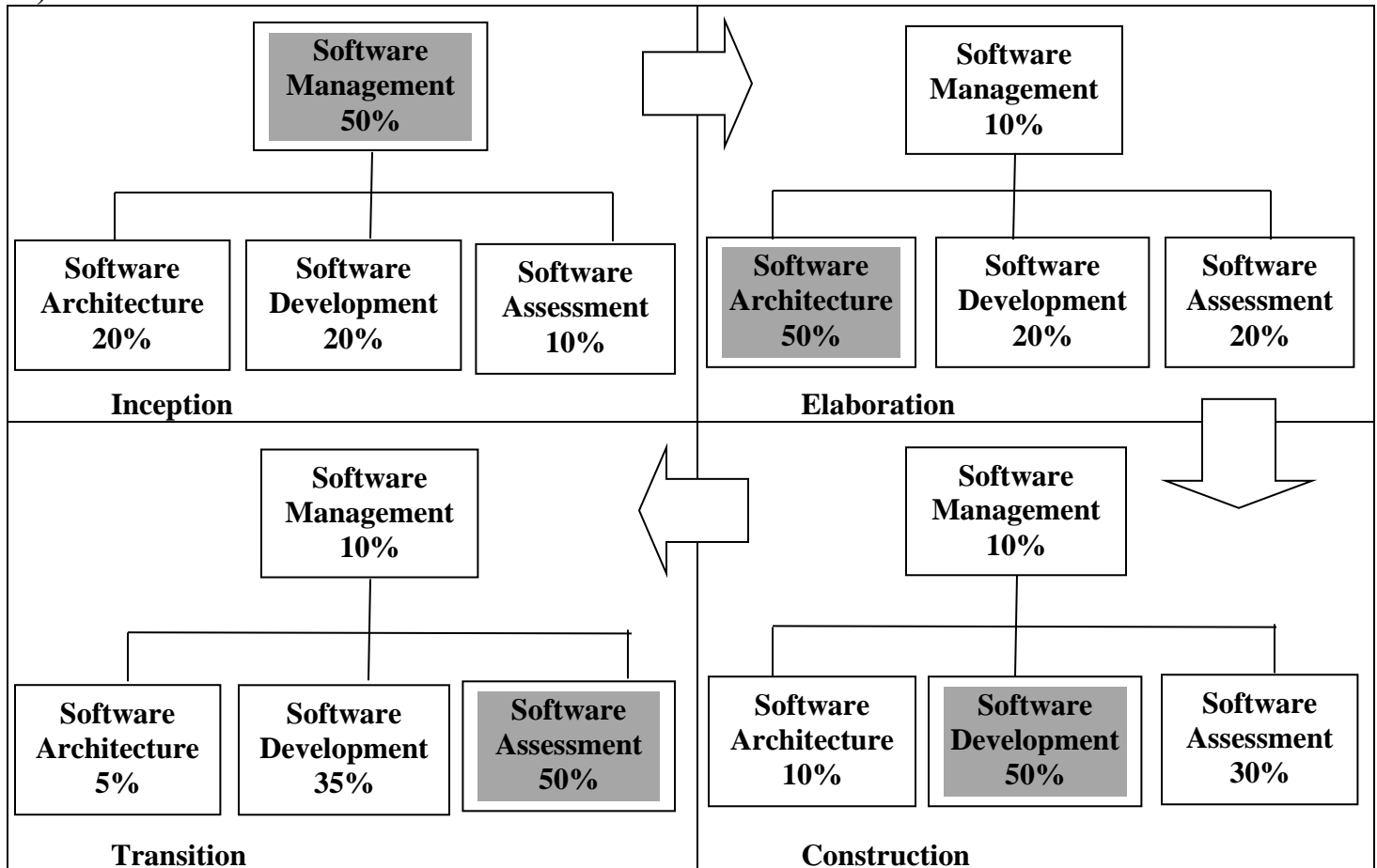


Figure 11-2. Default project organization and responsibilities

- The above figure shows a default project organization and maps project-level roles and responsibilities.
- The main features of the default organization are as follows:
- **The project management team** is an active participant, responsible for producing as well as managing.

- **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 from development.
- **Quality** is everyone's into all activities and checkpoints.
- Each team takes responsibility for a different quality perspective.

3) EVOLUTION OF ORGANIZATIONS:



<p>Inception: Software management: 50% Software Architecture: 20% Software development: 20% Software Assessment (measurement/evaluation):10%</p>	<p>Elaboration: Software management: 10% Software Architecture: 50% Software development: 20% Software Assessment (measurement/evaluation):20%</p>
<p>Construction: Software management: 10% Software Architecture: 10% Software development: 50% Software Assessment (measurement/evaluation):30%</p>	<p>Transition: Software management: 10% Software Architecture: 5% Software development: 35% Software Assessment (measurement/evaluation):50%</p>

The Process Automation:

Introductory Remarks:

The environment must be the first-class artifact of the process.

Process automation & change management is critical to an iterative process. If the change is expensive then the development organization will resist it.

Round-trip engineering & integrated environments promote change freedom & effective evolution of technical artifacts.

Metric automation is crucial to effective project control.

External stakeholders need access to environment resources to improve interaction with the development team & add value to the process.

The three levels of process which requires a certain degree of process automation for the corresponding process to be carried out efficiently.

Metaproces (Line of business): The automation support for this level is called an infrastructure.

Macroproces (project): The automation support for a project's process is called an environment.

Microproces (iteration): The automation support for generating artifacts is generally called a tool.

Tools: Automation Building blocks:

Many tools are available to automate the software development process. Most of the core software development tools map closely to one of the process workflows

<u>Workflows</u>	<u>Environment Tools & process Automation</u>
Management	Workflow automation, Metrics automation
Environment	Change Management, Document Automation
Requirements	Requirement Management
Design	Visual Modeling
Implementation	-Editors, Compilers, Debugger, Linker, Runtime
Assessment	-Test automation, defect Tracking
Deployment	defect Tracking

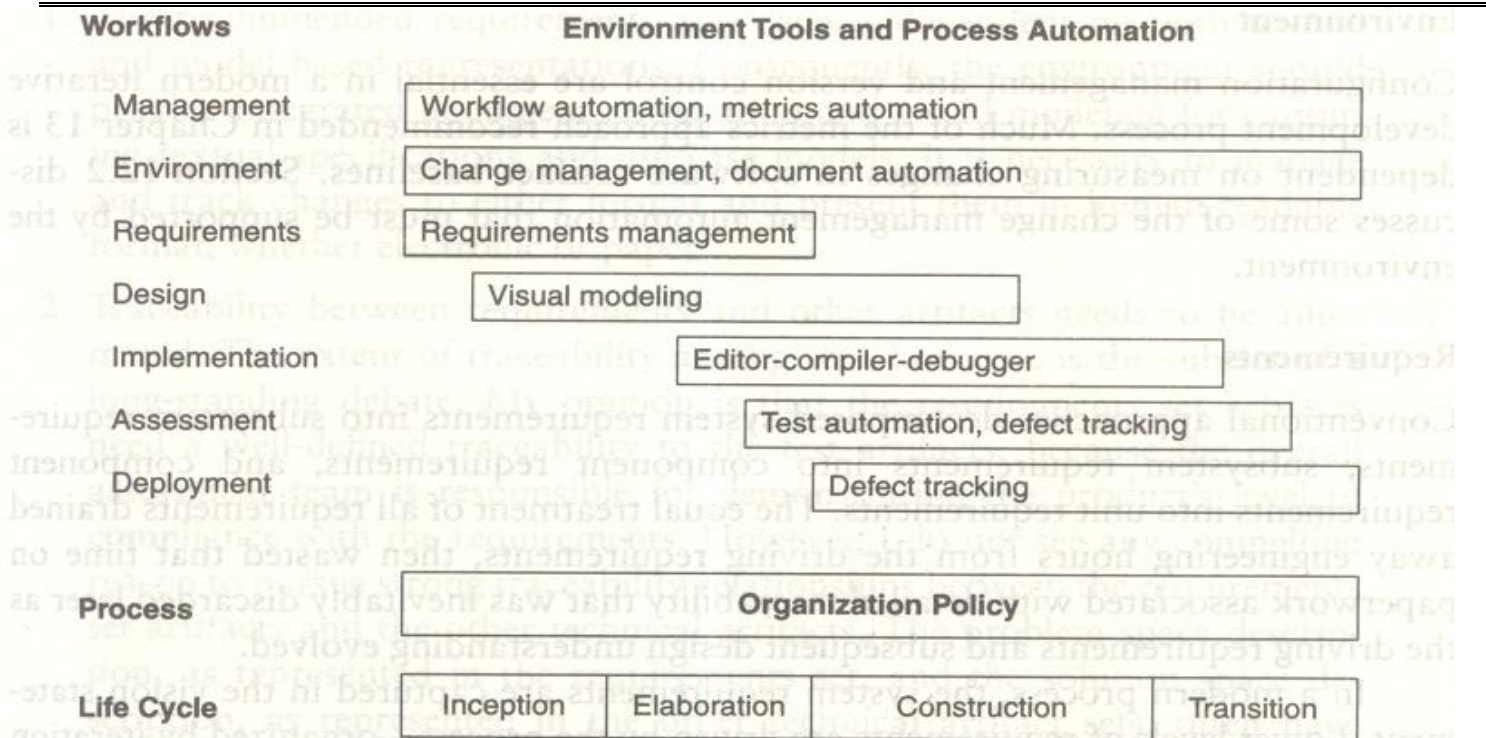


FIGURE 12-1. Typical automation and tool components that support the process workflows

The Project Environment:

The project environment artifacts evolve through three discrete states.

(1) Prototyping Environment. (2) Development Environment. (3) Maintenance Environment.

The **Prototype Environment** includes an architecture test bed for prototyping project architecture to evaluate trade-offs during inception & elaboration phase of the life cycle.

The **Development environment** should include a full suite of development tools needed to support various Process workflows & round-trip engineering to the maximum extent possible.

The **Maintenance Environment** should typically coincide with the mature version of the development.

There are four important environment disciplines that are critical to management context & the success of a modern iterative development process.

Round-Trip engineering

Change Management

Software Change Orders (SCO)

Configuration baseline Configuration Control Board

Infrastructure

Organization Policy

Organization Environment

Stakeholder Environment.

Round Trip Environment

Tools must be integrated to maintain consistency & traceability.

Round-Trip engineering is the term used to describe this key requirement for environment that support iterative development.

As the software industry moves into maintaining different information sets for the engineering artifacts, more automation support is needed to ensure efficient & error free transition of data from one artifacts to another.

Round-trip engineering is the environment support necessary to maintain Consistency among the engineering artifacts.

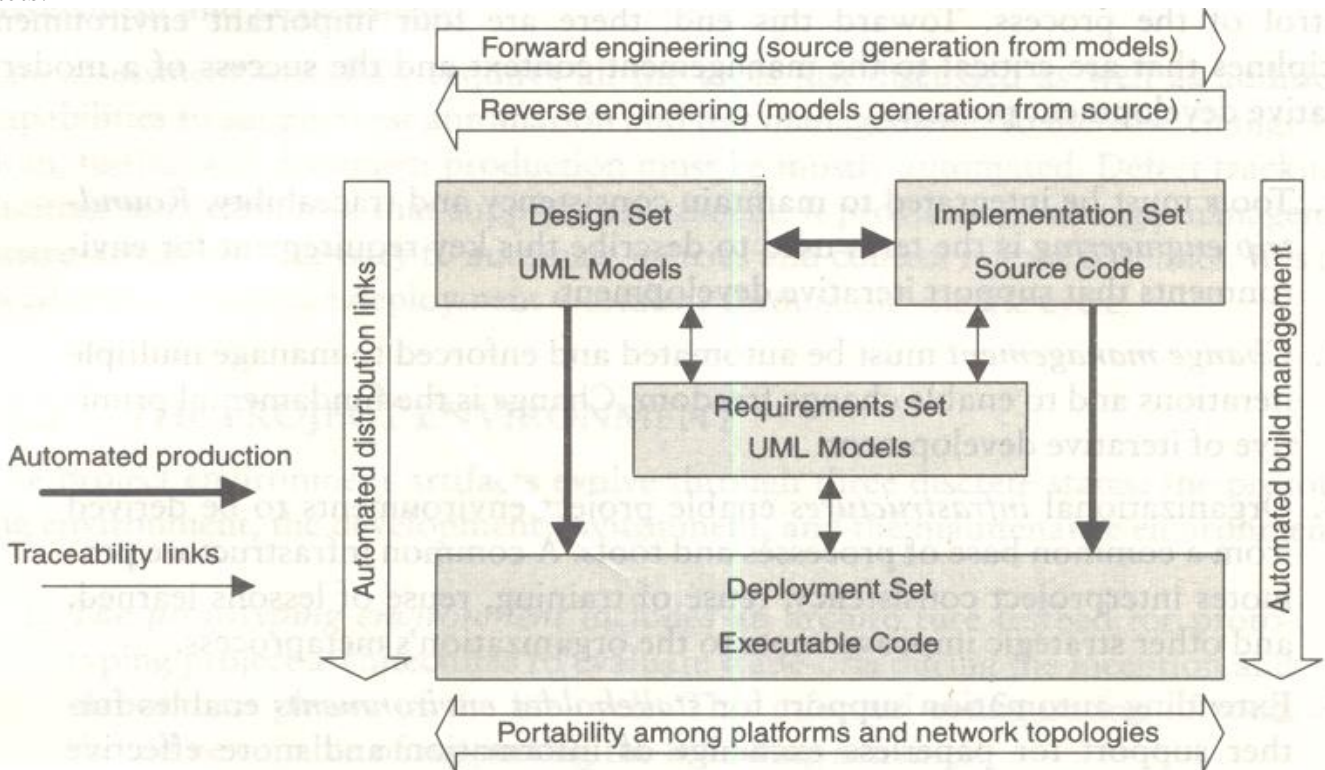


FIGURE 12-2. Round-trip engineering

Change Management

Change management must be automated & enforced to manage multiple iterations & to enable change freedom. Change is the fundamental primitive of iterative Development.

I. Software Change Orders

The atomic unit of software work that is authorized to create, modify or obsolete components within a configuration baseline is called a software change orders (SCO)

The basic fields of the SCO are Title, description, metrics, resolution, assessment & disposition

Title: _____

Description	Name: _____ Date: _____ Project: _____
Metrics	Category: _____ (0/1 error, 2 enhancement, 3 new feature, 4 other)
Initial Estimate	Actual Rework Expended
Breakage: _____	Analysis: _____ Test: _____
Rework: _____	Implement: _____ Document: _____
Resolution	Analyst: _____ Software Component: _____
Assessment	Method: _____ (inspection, analysis, demonstration, test)
Disposition	Tester: _____ Platforms: _____ Date: _____
Disposition	State: _____ Release: _____ Priority _____
Disposition	Acceptance: _____ Date: _____
Disposition	Closure: _____ Date: _____

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

Change management

II. Configuration Baseline

A configuration baseline is a named collection of software components & Supporting documentation that is subjected to change management & is upgraded, maintained, tested, statuses & obsolesced a unit

There are generally two classes of baselines

External Product Release

Internal testing Release

Three levels of baseline releases are required for most Systems

1. Major release (N)
2. Minor Release (M)
3. Interim (temporary) Release (X)

Major release represents a new generation of the product or project

A minor release represents the same basic product but with enhanced features, performance or quality.

Major & Minor releases are intended to be external product releases that are persistent & supported for a period of time.

An interim release corresponds to a developmental configuration that is intended to be transient.

Once software is placed in a controlled baseline all changes are tracked such that a distinction must be made for the cause of the change. Change categories are

Type 0: Critical Failures (must be fixed before release)

Type 1: A bug or defect either does not impair (Harm) 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 the update to the environment

Type 4: Changes that are not accommodated by the other categories.

Change Management

III Configuration Control Board (CCB)

A CCB is a team of people that functions as the decision

Authority on the content of configuration baselines

A CCB includes:

1. Software managers

2. Software Architecture managers

3. Software Development managers

4. Software Assessment managers

5. Other Stakeholders who are integral to the maintenance of the controlled software delivery system?

Infrastructure

The organization infrastructure provides the organization's capital assets including two key artifacts - Policy & Environment

I Organization Policy:

A Policy captures the standards for project software development processes

The organization policy is usually packaged as a handbook that defines the life cycles & the process primitives such as

-
- Major milestones
 - Intermediate Artifacts
 - Engineering repositories
 - Metrics
 - Roles & Responsibilities
-

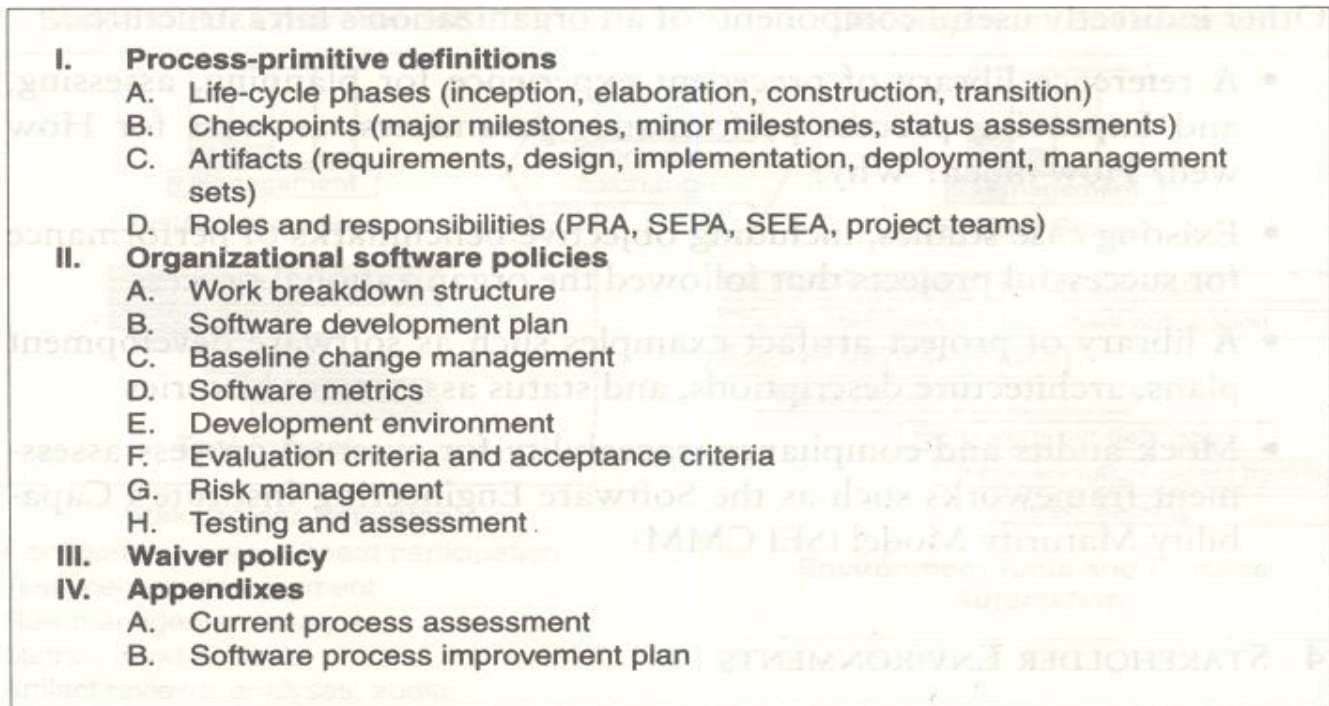


FIGURE 12-5. *Organization policy outline*

Infrastructure

II Organization Environment

The Environment that captures an inventory of tools which are building blocks from which project environments can be configured efficiently & economically

Stakeholder Environment

Many large scale projects include people in external organizations that represent other stakeholders participating in the development process they might include

-
- Procurement agency contract monitors
 - End-user engineering support personnel
 - Third party maintenance contractors
 - Independent verification & validation contractors
 - Representatives of regulatory agencies & others.
-

These stakeholder representatives also need to access to development resources so that they can contribute value to overall effort. These stakeholders will be access through on-line
An on-line environment accessible by the external stakeholders allow them to participate in the process a follows

Accept & use executable increments for the hands-on evaluation.

Use the same on-line tools, data & reports that the development organization uses to manage & monitor the project

Avoid excessive travel, paper interchange delays, format translations, paper * shipping costs & other overhead cost

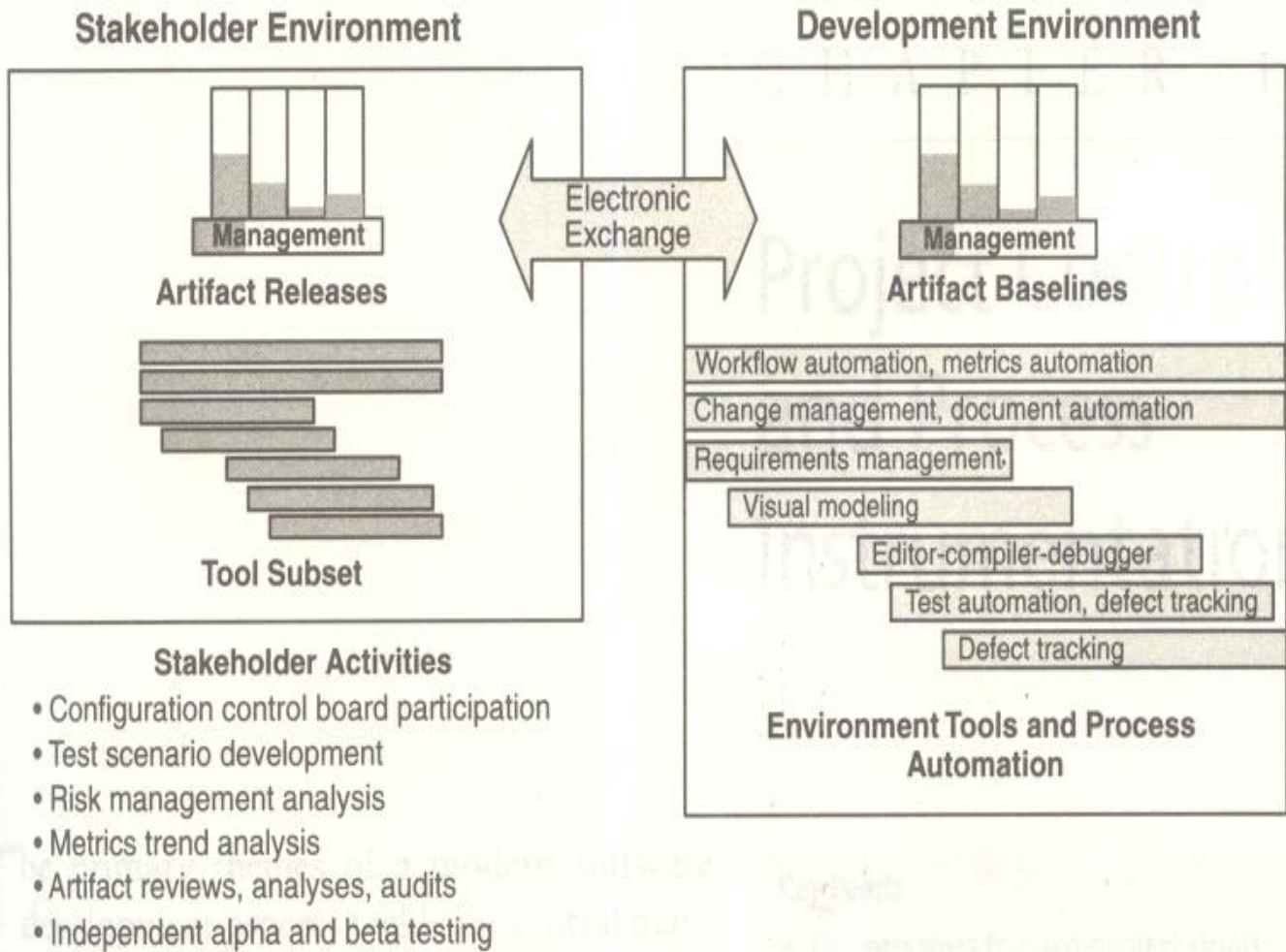


FIGURE 12-6. *Extending environments into stakeholder domains*

PROJECT CONTROL & PROCESS INSTRUMENTATION

INTERODUCTION: Software metrics are used to implement the activities and products of the software development process. Hence, the quality of the software products and the achievements in the development process can be determined using the software metrics.

Need for Software Metrics:

- Software metrics are needed for calculating the cost and schedule of a software product with great accuracy.
- Software metrics are required for making an accurate estimation of the progress.
- The metrics are also required for understanding the quality of the software product.

1.1 INDICATORS:

An indicator is a metric or a group of metrics that provides an understanding of the software process or software product or a software project. A software engineer assembles measures and produce metrics from which the indicators can be derived.

Two types of indicators are:

- (i) Management indicators.
- (ii) Quality indicators.

1.1.1 Management Indicators

The management indicators i.e., technical progress, financial status and staffing progress are used to determine whether a project is on budget and on schedule. The management indicators that indicate financial status are based on earned value system.

1.1.2 Quality Indicators

The quality indicators are based on the measurement of the changes occurred in software.

1.2 SEVEN CORE METRICS OF SOFTWARE PROJECT

Software metrics instrument the activities and products of the software development/integration process. Metrics values provide an important perspective for managing the process. The most useful metrics are extracted directly from the evolving artifacts.

There are seven core metrics that are used in managing a modern process.

Seven core metrics related to project control:

Management Indicators

- Work and Progress
- Budgeted cost and expenditures
- Staffing and team dynamics

Quality Indicators

- Change traffic and stability
- Breakage and modularity
- Rework and adaptability
- Mean time between failures (MTBF) and maturity

1.2.1 MANAGEMENT INDICATORS:

1.2.1.1 Work and progress

This metric measure the work performed over time. Work is the effort to be accomplished to complete a certain set of tasks. The various activities of an iterative development project can be measured by defining a planned estimate of the work in an objective measure, then tracking progress (work completed overtime) against that plan.

The default perspectives of this metric are:

Software architecture team: - Use cases demonstrated.

Software development team: - SLOC under baseline change management, SCOs closed

Software assessment team: - SCOs opened, test hours executed and evaluation criteria meet.

Software management team: - milestones completed.

The below figure shows expected progress for a typical project with three major releases

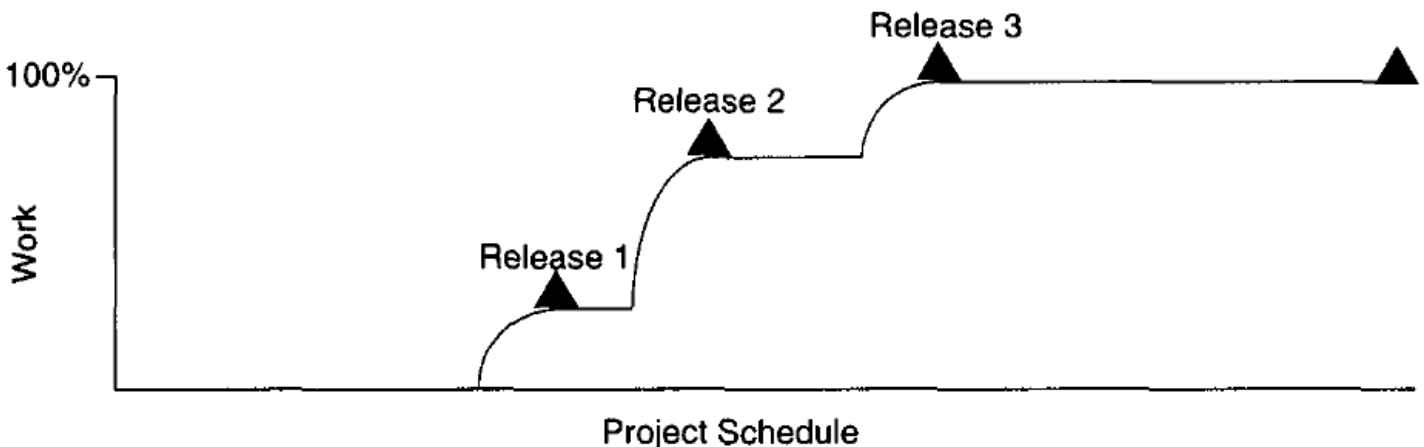


Fig: work and progress

1.2.1.2 Budgeted cost and expenditures

This metric measures cost incurred over time. Budgeted cost is the planned expenditure profile over the life cycle of the project. To maintain management control, measuring cost expenditures over the project life cycle is always necessary. Tracking financial progress takes on an organization - specific format. Financial performance can be measured by the use of an earned value system, which provides highly detailed cost and schedule insight. The basic parameters of an earned value system, expressed in units of dollars, are as follows:

Expenditure Plan - It is the planned spending profile for a project over its planned schedule. Actual progress - It is the technical accomplishment relative to the planned progress underlying the spending profile.

Actual cost: It is the actual spending profile for a project over its actual schedule.

Earned value: It is the value that represents the planned cost of the actual progress.

Cost variance: It is the difference between the actual cost and the earned value.

Schedule variance: It is the difference between the planned cost and the earned value. Of all parameters in an earned value system, actual progress is the most subjective

Assessment: Because most managers know exactly how much cost they have incurred and how much schedule they have used, the variability in making accurate assessments is centred in the actual progress assessment. The default perspectives of this metric are cost per month, full-time staff per month and percentage of budget expended.

1.2.1.3 Staffing and team dynamics

This metric measures the personnel changes over time, which involves staffing additions and reductions over time. An iterative development should start with a small team until the risks in the requirements and architecture have been suitably resolved. Depending on the overlap of iterations and other project specific circumstances, staffing can vary. Increase in staff can slow overall project progress as new people consume the productive team of existing people in coming up to speed. Low attrition of good people is a sign of success. The default perspectives of this metric are people per month added and people per month leaving. These three management indicators are responsible for technical progress, financial status and staffing progress.

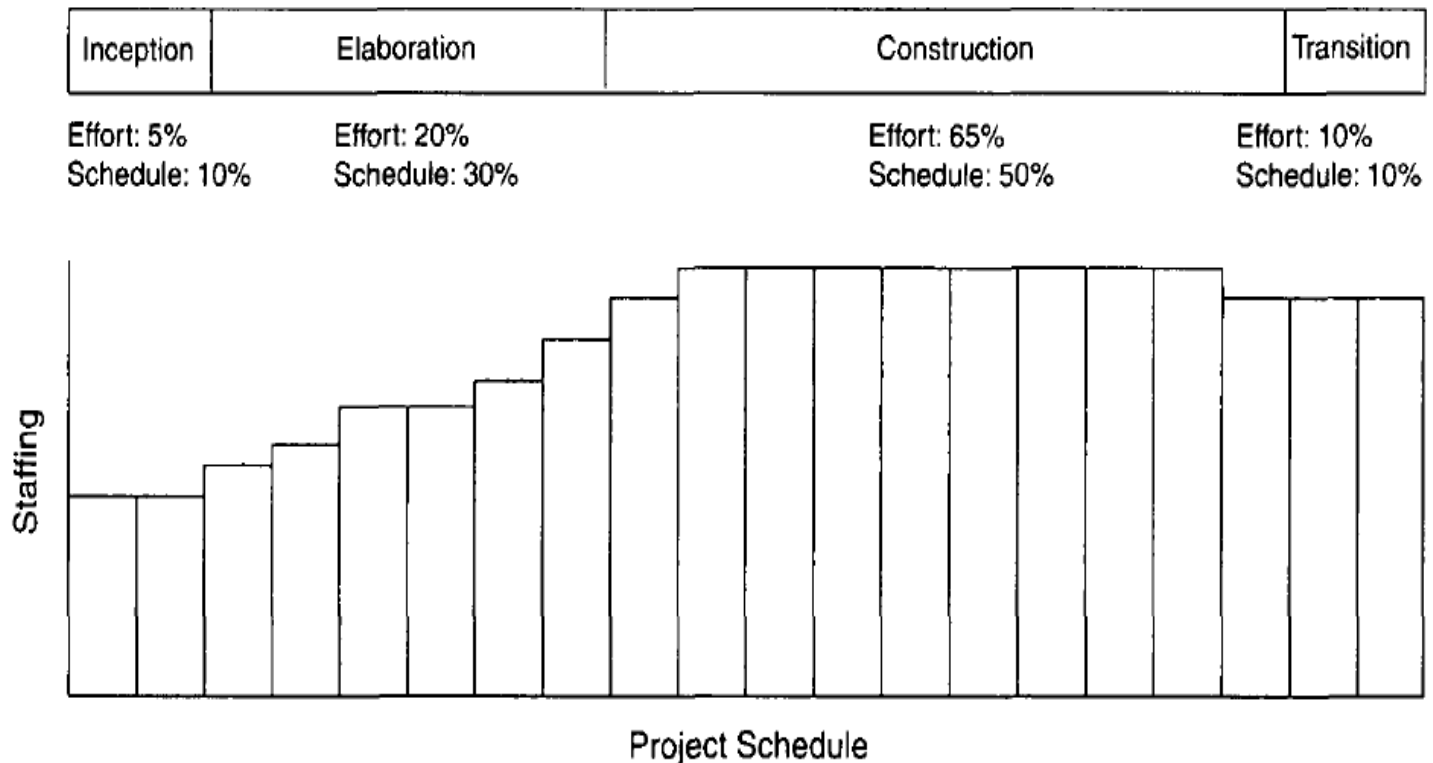


Fig: staffing and Team dynamics

1.2.2 QUALITY INDICATORS:

1.2.2.1 Change traffic and stability:

This metric measures the change traffic over time. The number of software change orders opened and closed over the life cycle is called change traffic. Stability specifies the relationship between opened versus closed software change orders. This metric can be collected by change type, by release, across all releases, by term, by components, by subsystems, etc.

The below figure shows stability expectation over a healthy project's life cycle

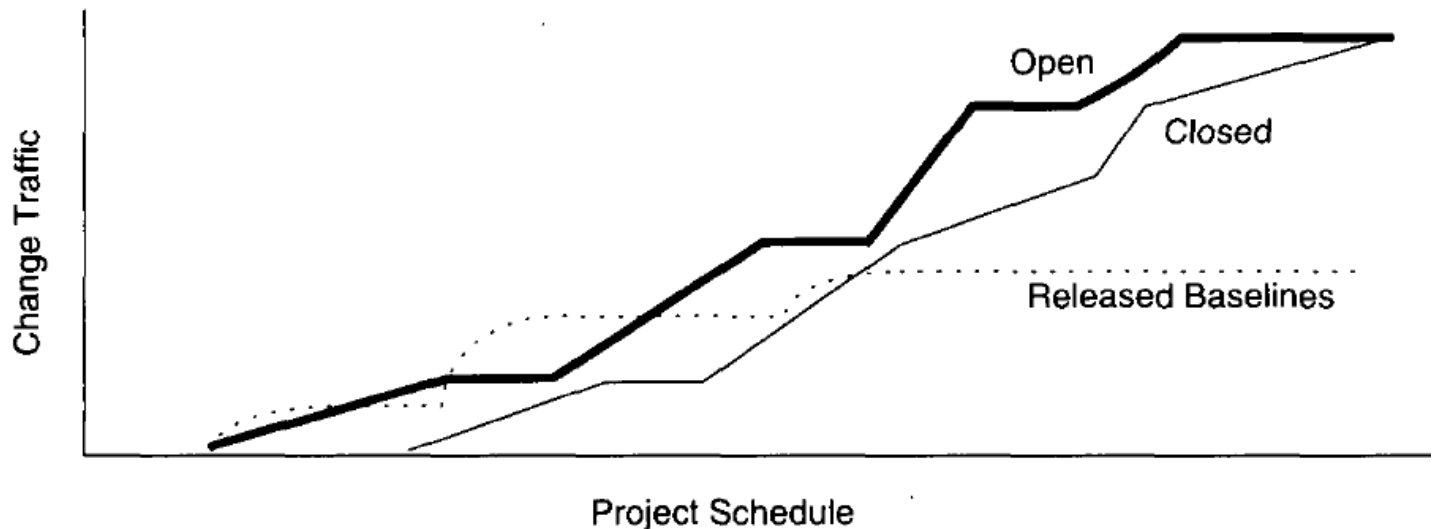


Fig: Change traffic and stability

1.2.2.2 Breakage and modularity

This metric measures the average breakage per change over time. Breakage is defined as the average extent of change, which is the amount of software baseline that needs rework and measured in source lines of code, function points, components, subsystems, files or other units. Modularity is the average breakage trend over time. This metric can be collected by revoke SLOC per change, by change type, by release, by components and by subsystems.

1.2.2.3 Rework and adaptability:

This metric measures the average rework per change over time. Rework is defined as the average cost of change which is the effort to analyse, resolve and retest all changes to software baselines. Adaptability is defined as the rework trend over time. This metric provides insight into rework measurement. All changes are not created equal. Some changes can be made in a staff-hour, while others take staff-weeks. This metric can be collected by average hours per change, by change type, by release, by components and by subsystems.

1.2.2.4 MTBF and Maturity:

This metric measures defect rather over time. MTBF (Mean Time Between Failures) is the average usage time between software faults. It is computed by dividing the test hours by the number of type 0 and type 1 SCOs. Maturity is defined as the MTBF trend over time. Software errors can be categorized into two types deterministic and nondeterministic. Deterministic errors are also known as Bohr-bugs and nondeterministic errors are also called as Heisen-bugs. Bohr-bugs are a class of errors caused when the software is stimulated in a certain way such as coding errors. Heisen-bugs are software faults that are coincidental with a certain probabilistic occurrence of a given situation, such as design errors. This metric can be collected by failure counts, test hours until failure, by release, by components and by subsystems. These four quality indicators are based primarily on the measurement of software change across evolving baselines of engineering data.

1.3 LIFE -CYCLE EXPECTATIONS:

There is no mathematical or formal derivation for using seven core metrics properly. However, there were specific reasons for selecting them:

The quality indicators are derived from the evolving product rather than the artifacts.

They provide inside into the waste generated by the process. Scrap and rework metrics are a standard measurement perspective of most manufacturing processes.

They recognize the inherently dynamic nature of an iterative development process. Rather than focus on the value, they explicitly concentrate on the trends or changes with respect to time.

The combination of insight from the current and the current trend provides tangible indicators for management action.

Table 13-3. the default pattern of life cycle evolution

Metric	Inception	Elaboration	Construction	Transition
Progress	5%	25%	90%	100%
Architecture	30%	90%	100%	100%
Applications	<5%	20%	85%	100%
Expenditures	Low	Moderate	High	High
Effort	5%	25%	90%	100%
Schedule	10%	40%	90%	100%
Staffing	Small team	Ramp up	Steady	Varying
Stability	Volatile	Moderate	Moderate	Stable
Architecture	Volatile	Moderate	Stable	Stable
Applications	Volatile	Volatile	Moderate	Stable
Modularity	50%-100%	25%-50%	<25%	5%-10%
Architecture	>50%	>50%	<15%	<5%
Applications	>80%	>80%	<25%	<10%

Adaptability	Varying	Varying	Benign.	Benign
Architecture	Varying	Moderate	Benign	Benign
Applications	Varying	Varying	Moderate	Benign
Maturity	Prototype	Fragile	Usable	Robust
Architecture	Prototype	Usable	Robust	Robust
Applications	Prototype	Fragile	Usable	Robust

1.4 METRICS AUTOMATION:

Many opportunities are available to automate the project control activities of a software project. A Software Project Control Panel (SPCP) is essential for managing against a plan. This panel integrates data from multiple sources to show the current status of some aspect of the project. The panel can support standard features and provide extensive capability for detailed situation analysis. SPCP is one example of metrics automation approach that collects, organizes and reports values and trends extracted directly from the evolving engineering artifacts.

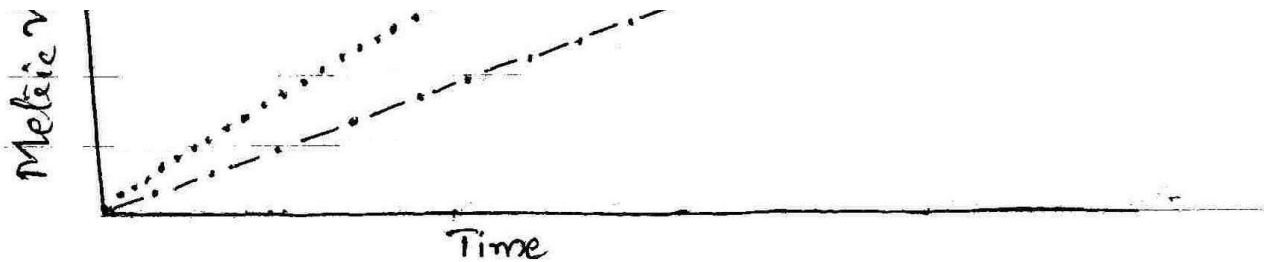
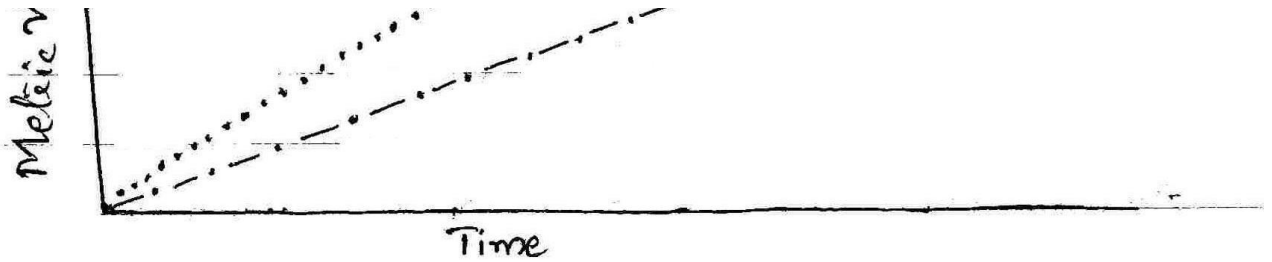
SPCP:

To implement a complete SPCP, the following are necessary.

- Metrics primitives - trends, comparisons and progressions
- A graphical user interface.
- Metrics collection agents
- Metrics data management server
- Metrics definitions - actual metrics presentations for requirements progress, implementation progress, assessment progress, design progress and other progress dimensions.
- Actors - monitor and administrator.

Monitor defines panel layouts, graphical objects and linkages to project data. Specific monitors called roles include software project managers, software development team leads, software architects and customers. Administrator installs the system, defines new mechanisms, graphical objects and linkages. The whole display is called a panel. Within a panel are graphical objects, which are types of layouts such as dials and bar charts for information. Each graphical object displays a metric. A panel contains a number of graphical objects positioned in a particular geometric layout. A metric shown in a graphical object is labelled with the metric type, summary level and insurance name (line of code, subsystem, server1). Metrics can be displayed in two modes – value, referring to a given point in time and graph referring to multiple and consecutive points in time. Metrics can be displayed with or without control values. A control value is an existing expectation either absolute or relative that is used for comparison with a dynamically changing metric. Thresholds are examples of control values.

The basic fundamental metrics classes are trend, comparison and progress.



The format and content of any project panel are configurable to the software project manager's preference for tracking metrics of top-level interest. The basic operation of an SPCP can be described by the following top-level use case.

- i. Start the SPCP
- ii. Select a panel preference
- iii. Select a value or graph metric
- iv. Select to superimpose controls
- v. Drill down to trend
- vi. Drill down to point in time.
- vii. Drill down to lower levels of information
- viii. Drill down to lower level of indicators.

10 Mark Questions

1. Define metric. Discuss seven core metrics for project control and process instrumentation with suitable examples?
2. List out the three management indicators that can be used as core metrics on software projects. Briefly specify meaning of each?
3. Explain the various characteristics of good software metric. Discuss the metrics Automation using appropriate example?
4. Explain about the quality indicators that can be used as core metrics on software projects.
5. Explain Management Indicators with suitable example?
6. Define MTBF and Maturity. How these are related to each other?
7. Briefly explain about Quality Indicators?
8. Write short notes on Pragmatic software metrics?