3. LIFE CYCLE PHASES AND ARTIFACTS OF THE PROCESS

3.0 INTRODUCTION

- Characteristic of a successful software development process is the well-defined separation between "research and development" activities and "production" activities. Most unsuccessful project:; 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:

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

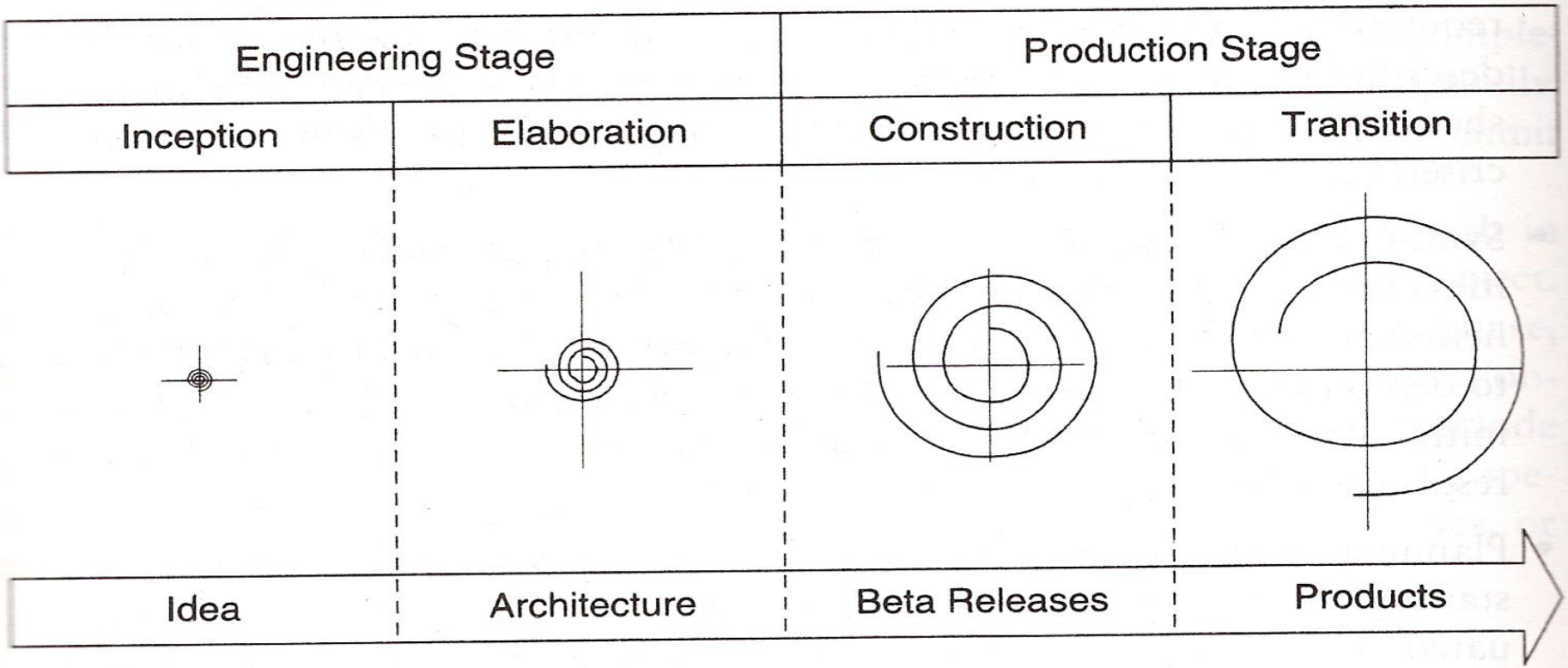
3.1 ENGINEERING AND PRODUCTION STAGES

- 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:
  - The **Engineering stage,** driven by less predictable but smaller teams doing design and synthesis activities.
  - The **Production stage,** driven by more predictable but larger team~ doing construction, test, and deployment activities.

TABLE 5-1: *The two stages of the Life Cycle: Engineering and Production.*

LIFE-CYCLE ASPECT	ENGINEERING STAFF 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.**



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

3.2 INCEPTION, ELEBORATION, CONSTRUCTION, TRANSITION PHASES

3.2.1 INCEPTION PHASE

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

❑ Primary Objectives

- Establishing the project's software scope and boundary condition, including all 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 scenarios.
- Estimating the cost and schedule for the entire project (including detailed estimates for the elaboration phase).
- Estimating potential risks (sources of un predictability)

❑ 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**

3.2.2 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 call 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 and risk.
- **Primary Objectives**
- Base lining the architecture as rapidly as practical (establishing a configuration-managed snapshot in which all changes are rationalized, tracked, and maintained)
- Base lining the vision
- Base lining 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?

3.2.3 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 eth 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?

3.2.4 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:
- Beta testing to validate the new system against user expectations
- Beta testing and parallel operation relative to a legacy system it is replacing
- conversion of operational databases
- Training of user 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 produce 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.
- Primary Evaluation Criteria
  - Is the user satisfied
  - Are actual resource expenditures versus planned expenditures acceptable

3.3 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 associate files). The artifact sets are shown in Figure 6-1.

Requirement Set	Design Set	Implementation Set	Deployment Set
1. Vision document 2. Requirements Model(s)	1. Design Model(s) 2. Test Model 3. Software architecture description	1. Source code baselines 2. Associated compile-time files 3. Component executables.	1. Integrated product executable baselines 2. Associated run-time files 3. User Manual
Management Set			
Planning Artifacts		**Operational Artifacts**	
1. Work breakdown structure 2. Business case 3. Release specifications 4. Software development plan		5. Release descriptions 6. Status assessment 7. Software change order database 8. Deployment documents 9. Environment	

3.3.1 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.**

3.3.2 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:

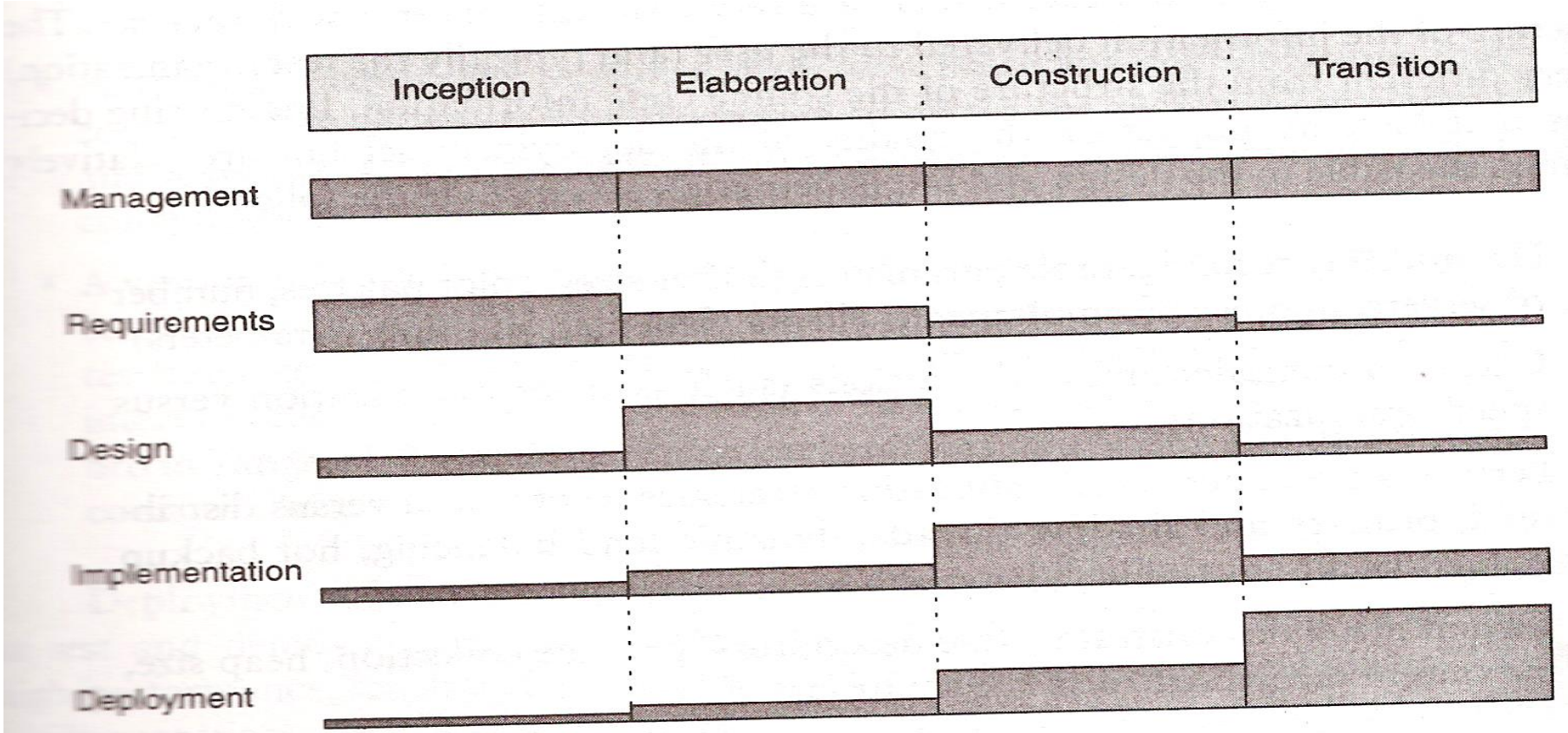


FIGURE 6-2. *Life-cycle focus on artifact sets*

- 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.

❑ 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 or 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.

3.3.3 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.**

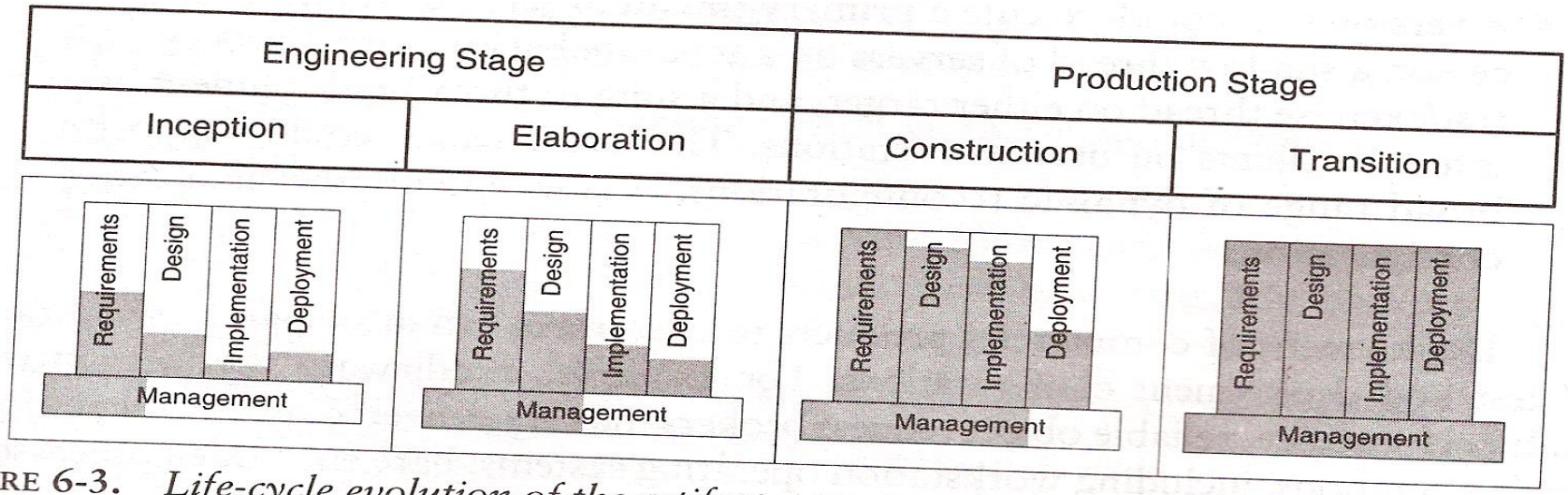


FIGURE 6-3. *Life-cycle evolution of the artifact sets*

- **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.**

3.3.4 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 non deliverable 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.

3.4 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 progress. Figure 6-4 provides a default outline for a business case.

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***

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 shelf ware) and understanding and acceptance by managers and practitioners alike. Figure 6-5 provides a default outline for a software development plan.

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. Follow through 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.

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***

❑ 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.

❑ 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 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 m1ifacts 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.

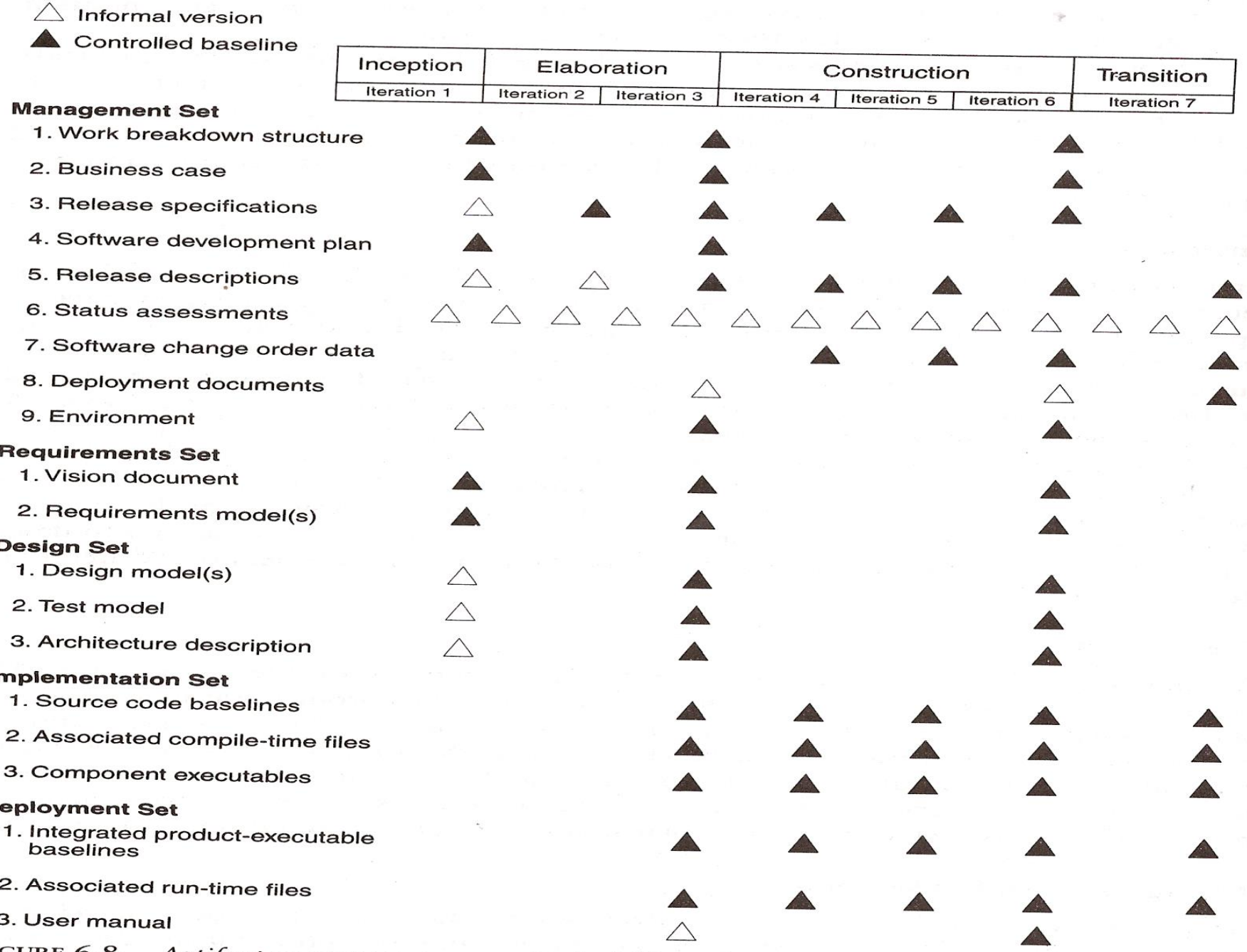


FIGURE 6-8. *Artifact sequences across a typical life cycle*

3.5 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 eth development organization. A project vision is meant to be changeable as understanding evolves of the requirements, architecture, plans and technology. A good visions document should change slowly. Figure 6-9 provides a default outline for a visions 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 et 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 fro 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 Use Manual

- The software user manual provides the user with the reference documentation necessary to support 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 interfaced, this manual should be developed early in the life cycle because it is a necessary mechanism for communication 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.

3.6 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, organization is 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.

MODEL QUESTIONS

1. Justify the dividing of the four phases of software life-cycle into engineering and production stages.
2. What are the primary objectives of the four phases of software life-cycle?
3. What are the essential activities in inception and elaboration phases?
4. What are the essential activities in construction and transition phases?
5. How do you evaluate the completion of each of the four phases in software life cycle?
6. Give an overview of the artifact sets. Also, explain the artifacts in management set.
7. Discuss briefly about the various sets that are included in the engineering set.
8. Write notes on the following.
    a. Artifact evolution over the software life-cycle.
    b. Test Artifacts.
9. Distinguish between implementation asset and deployment set.
10. Discuss the planning artifacts of a management set.
11. How an operational artifact of a management set differs from planning artifacts? Explain.
12. Draw and explain the artifact sequences across the software life-cycle.
13. Discuss in detail about the various engineering artifacts in software project management.
14. Explain pragmatic artifacts of software project management.

3.5 MODEL BASED SOFTWARE ARCHITECTURE

ARCHITECTURE: A MANAGEMENT PERSPECTIVE

- The most critical technical product of a software project is its architecture: the infrastructure, control and date interfaces that permit software components to co-operate as a system and software designers to co-operate efficiently as a tem. When the communications media include multiple languages and inter group 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 difference aspects of architecture.
  - *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.
  - *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).

- *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 form 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 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 fro 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.**

ARCHITECTURE: A TECHNICAL PERSPECTIVE

- An architecture framework is defined in the 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 relationship among the design, component and deployment views.
  - Component: Describes the structure of the implementation set.
  - Deployment: Describes the structures 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.

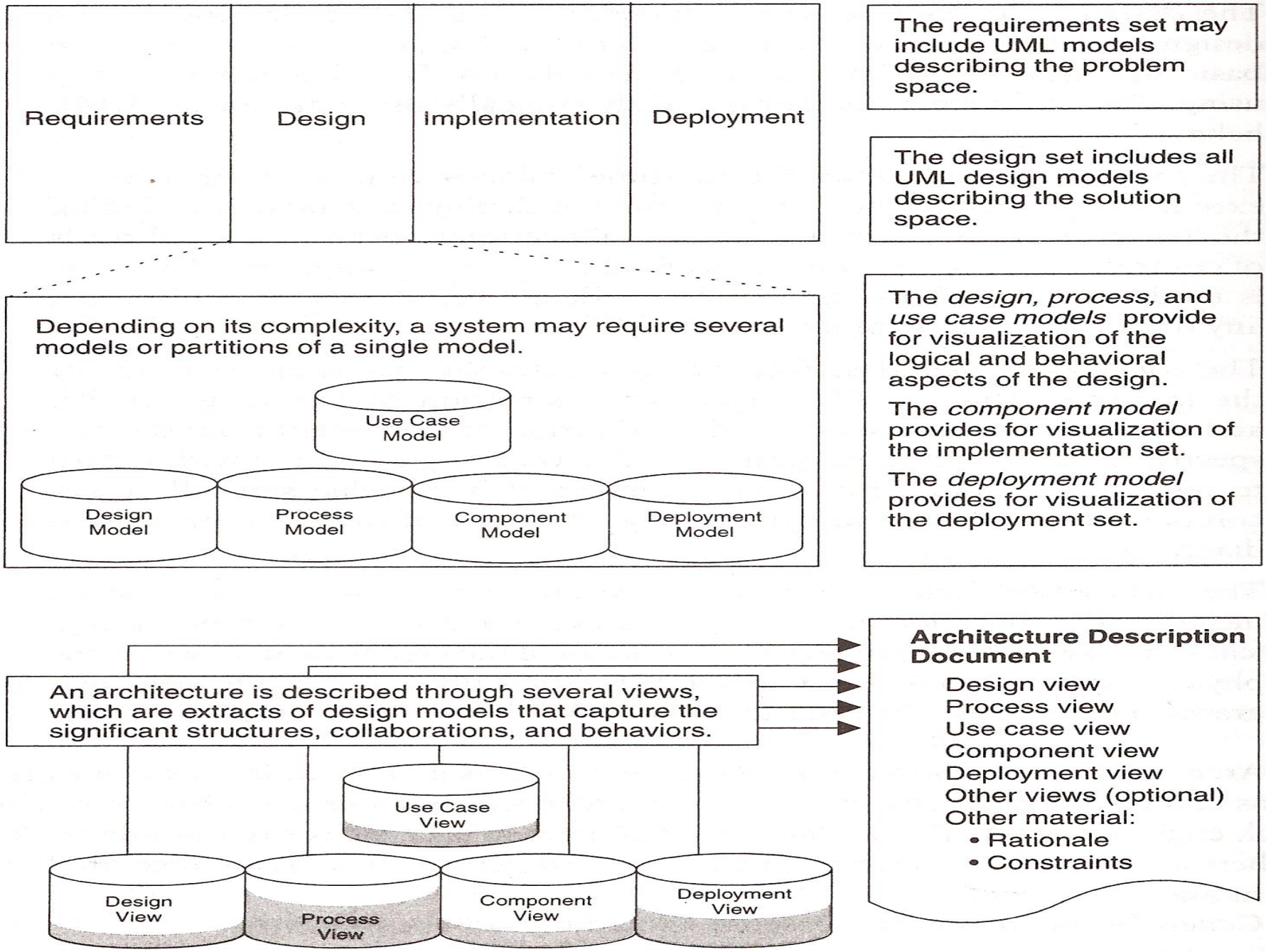


FIGURE 7-1. *Architecture, an organized and abstracted view into the design models*

- 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 calls 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 process and threads of control), inter process 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 including 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
  - **Development:** executable components sufficient to demonstrate the critical us cases and the risk associated with achieving the system qualities.

MODEL QUESTIONS

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