UNIT-III

LIFE-CYCLE PHASES

INTRODUCTION:

- If there is a well defined separation between “research and development” activities and “production” activities then the software is said to be in successful development process.

- Most of the software’s fail due to the following characteristics ,

1) An overemphasis on research and development.
2) An overemphasis on production.

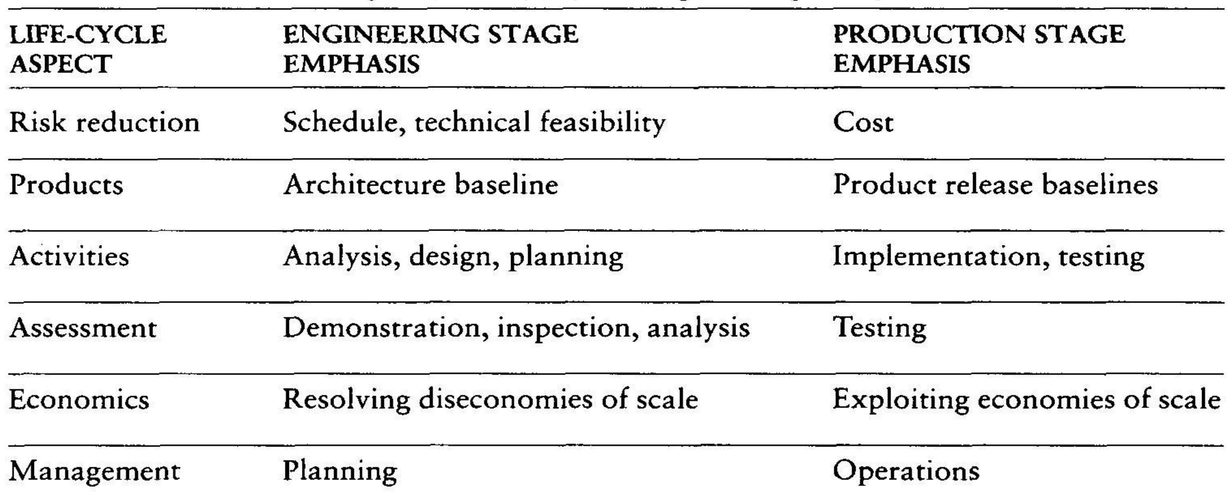
ENGINEERING AND PRODUCTION STAGES :

To achieve economics of scale and higher return on investment, we must move toward a software manufacturing process which is determined by technological improvements in process automation and component based development.

There are two stages in the software development process

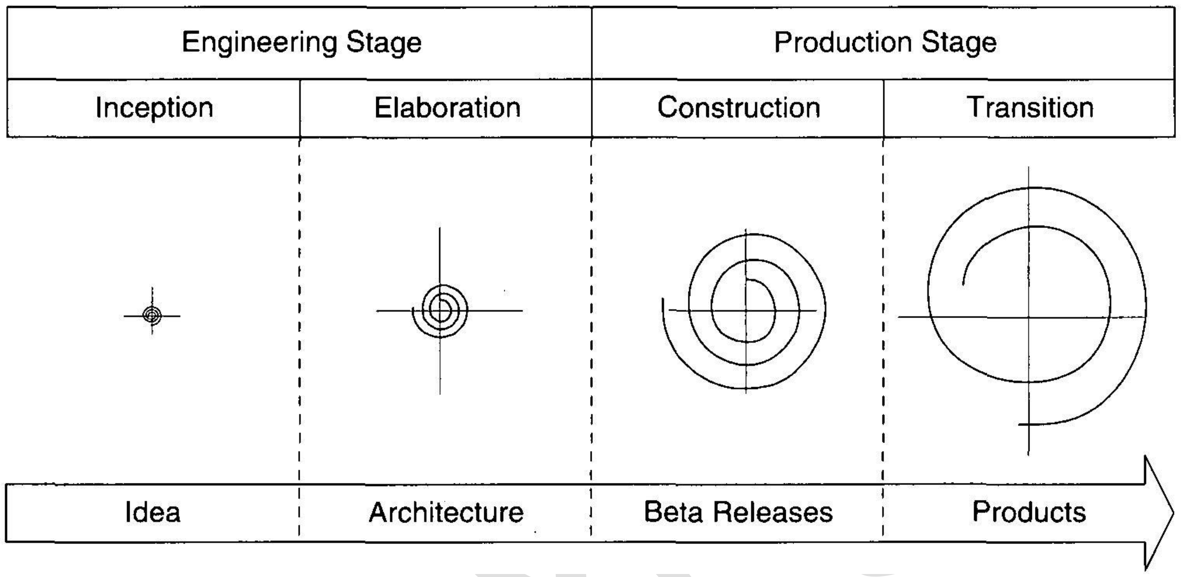
1) **The engineering stage:** Less predictable but smaller teams doing design and production activities. This stage is decomposed into two distinct phases *inception* and *elaboration*.
2) **The production stage:** More predictable but larger teams doing construction, test, and deployment activities. This stage is also decomposed into two distinct phases *construction* and *transition*.

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

These four phases of lifecycle process are loosely mapped to the conceptual framework of the spiral model is as shown in the following figure**.**



- In the above figure the size of the spiral corresponds to the inactivity of the project with respect to the breadth and depth of the artifacts that have been developed.
- This inertia manifests itself in maintaining artifact consistency, regression testing, documentation, quality analyses, and configuration control.
- Increased inertia may have little, or at least very straightforward, impact on changing any given discrete component or activity.
- However, the reaction time for accommodating major architectural changes, major requirements changes, major planning shifts, or major organizational perturbations clearly increases in subsequent phases.

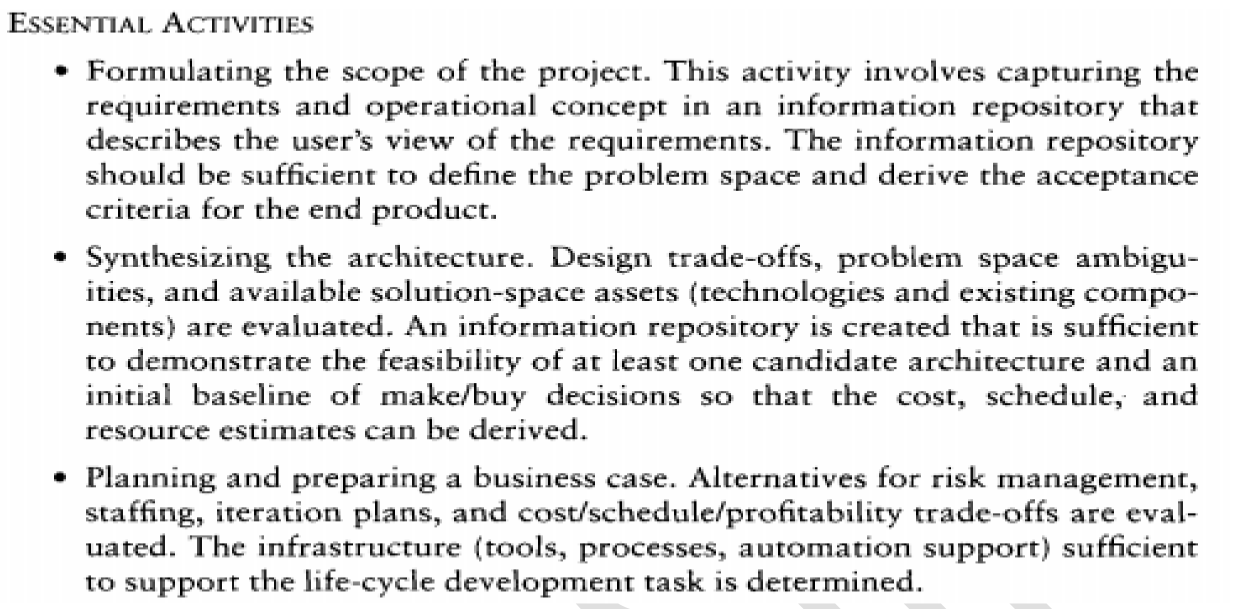
INCEPTION PAHSE:

The main goal of this phase is to achieve agreement among stakeholders on the life-cycle objectives for the project.

PRIMARY OBJECTIVES

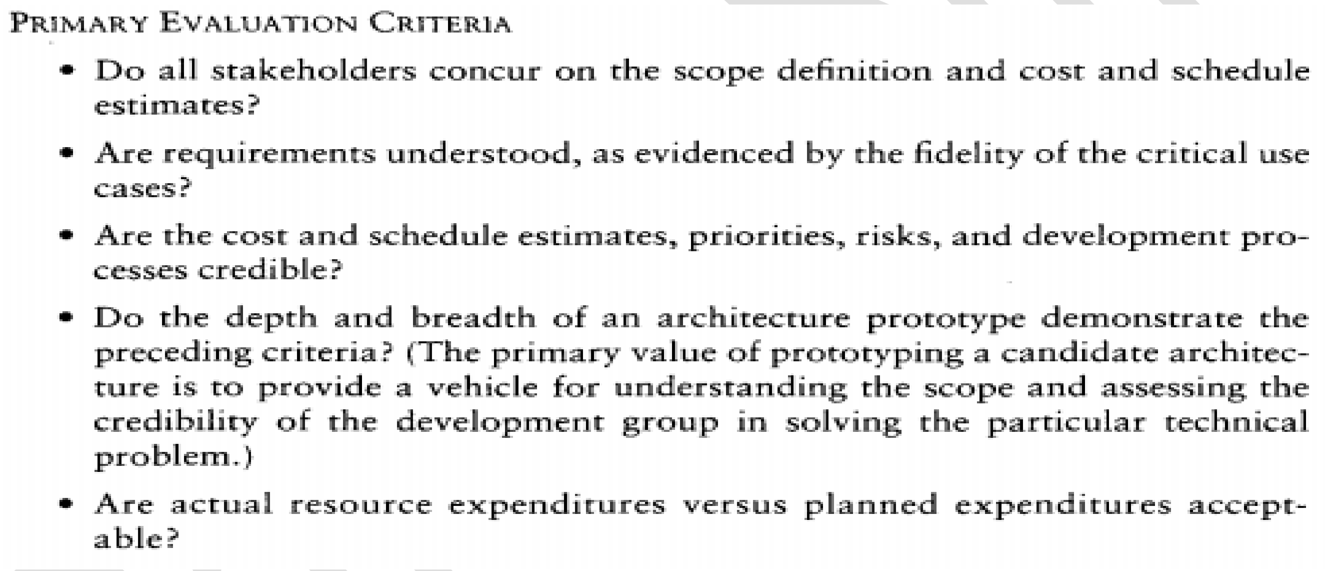
1) Establishing the project's scope and boundary conditions
2) Distinguishing the critical use cases of the system and the primary scenarios of operation
3) Demonstrating at least one candidate architecture against some of the primary scenarios

4) Estimating cost and schedule for the entire project
5) Estimating pot
6) ential risks



ESSENTIAL ACTIVITIES

- Formulating the scope of the project. This activity involves capturing the requirements and operational concept in an information repository that describes the user's view of the requirements. The information repository should be sufficient to define the problem space and derive the acceptance criteria for the end product.
- Synthesizing the architecture. Design trade-offs, problem space ambiguities, and available solution-space assets (technologies and existing components) are evaluated. 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. The infrastructure (tools, processes, automation support) sufficient to support the life-cycle development task is determined.



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 a 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?

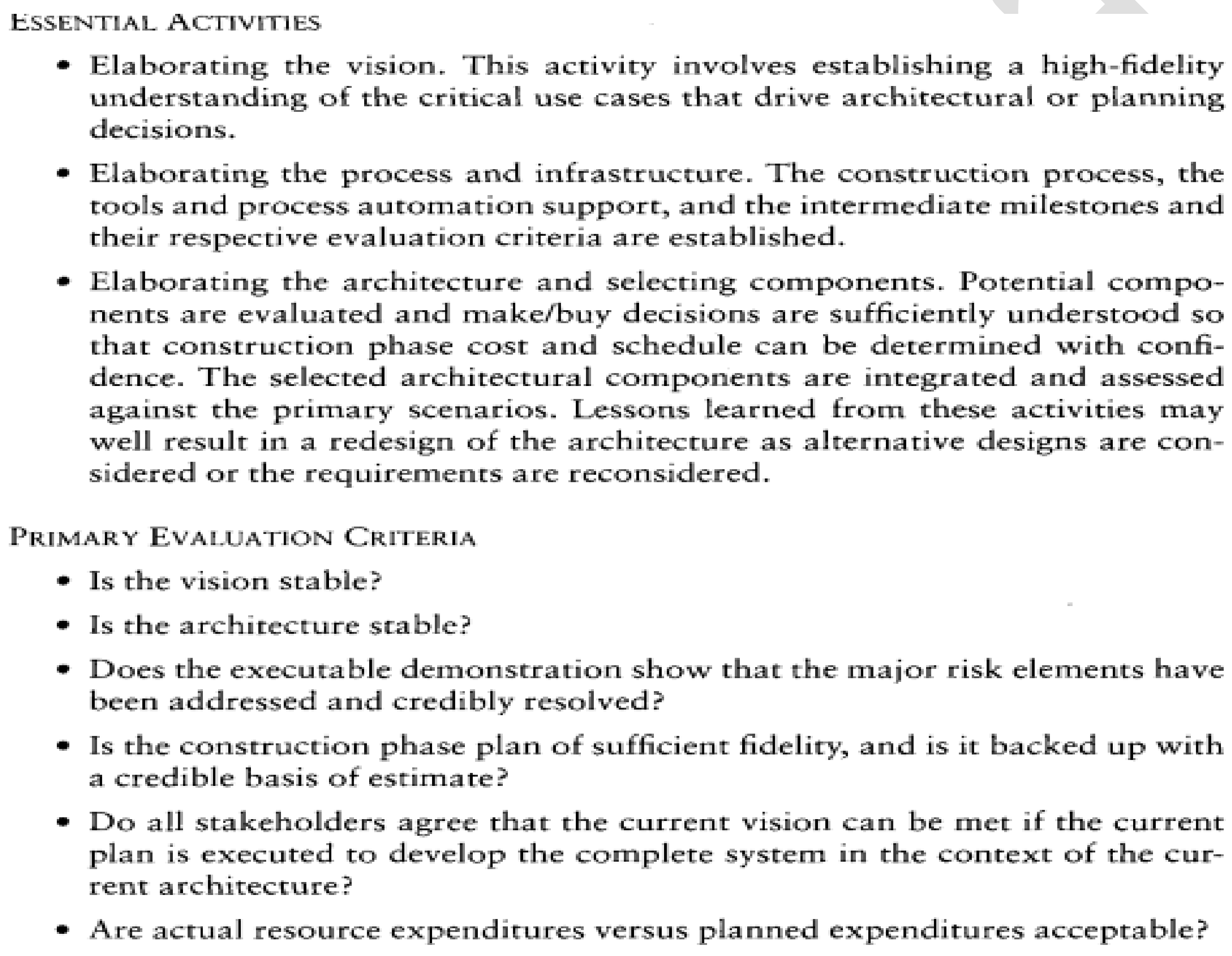
ELABORATION PHASE

- It is the most critical phase among the four phases.
- Depending upon the scope, size, risk, and freshness of the project, an executable architecture prototype is built in one or more iterations.
- At most of the time the process may accommodate changes, the elaboration phase activities must ensure that the architecture, requirements, and plans are stable. And also the cost and schedule for the completion of the development can be predicted within an acceptable range.

PRIMARY OBJECTIVES

1) Base lining the architecture as rapidly as practical
2) Base lining the vision
3) Base lining a high-reliability plan for the construction phase

4) Demonstrating that the baseline architecture will support the vision at a reasonable cost in a reasonable time.



ESSENTIAL ACTIVITIES

- Elaborating the vision. This activity involves establishing a high-fidelity understanding of the critical use cases that drive architectural or planning decisions.
- Elaborating the process and infrastructure. The construction process, the tools and process automation support, and the intermediate milestones and their respective evaluation criteria are established.
- Elaborating the architecture and selecting components. Potential components are evaluated and make/buy decisions are sufficiently understood so that construction phase cost and schedule can be determined with confidence. The selected architectural components are integrated and assessed against the primary scenarios. Lessons learned from these activities may well result in a redesign of the architecture as alternative designs are considered or the requirements are reconsidered.

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?

CONSTRUCTION PHASE

During this phase all the remaining components and application features are integrated into the application, and all features are thoroughly tested. Newly developed software is integrated where ever required.

- If it is a big project then parallel construction increments are generated.

PRIMARY OBJECTIVES

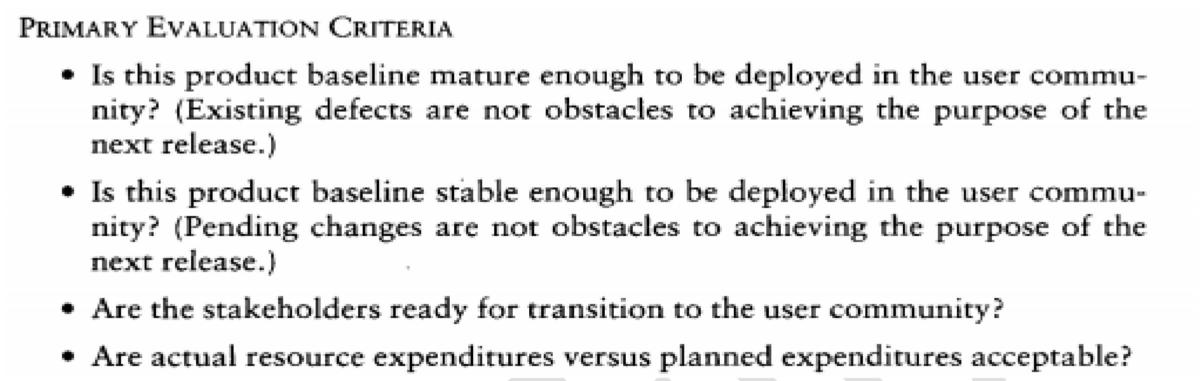
1) Minimizing development costs

2) Achieving adequate quality as rapidly as practical
3) Achieving useful version (alpha, beta, and other releases) as rapidly as practical

ESSENTIAL ACTIVITIES

1) Resource management, control, and process optimization
2) Complete component development and testing evaluation criteria

3) Assessment of product release 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?

TRANSITION PHASE

Whenever a project is grown-up completely and to be deployed in the end-user domain this phase is called transition phase. It includes the following activities:

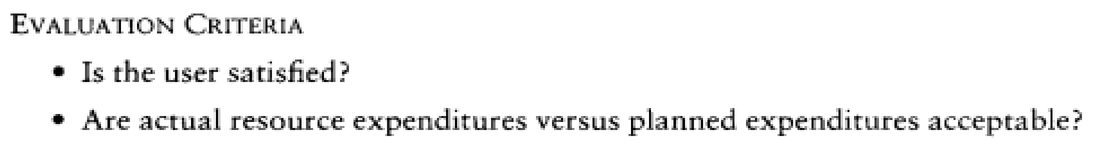
1) Beta testing to validate the new system against user expectations
2) Beta testing and parallel operation relative to a legacy system it is replacing
3) Conversion of operational databases
4) Training of users and maintainers

PRIMARY OBJECTIVES

1) Achieving user self-supportability
2) Achieving stakeholder concurrence
3) Achieving final product baseline as rapidly and cost-effectively as practical

ESSENTIAL ACTIVITIES

1) Synchronization and integration of concurrent construction increments into consistent deployment baselines
2) Deployment-specific engineering
3) Assessment of deployment baselines against the complete vision and acceptance criteria in the requirement set.


EVALUATION CRITERIA

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

<u>Artifacts of the Process</u>

- Conventional s/w projects focused on the sequential development of s/w artifacts:
- Build the requirements

- Construct a design model traceable to the requirements &
- Compile and test the implementation for deployment.
-This process can work for small-scale, purely custom developments in which the design representation, implementation representation and deployment representation are closely aligned.

This approach is doesn't work for most of today's s/w systems why because of having complexity and are composed of numerous components some are custom, some reused, some commercial products.

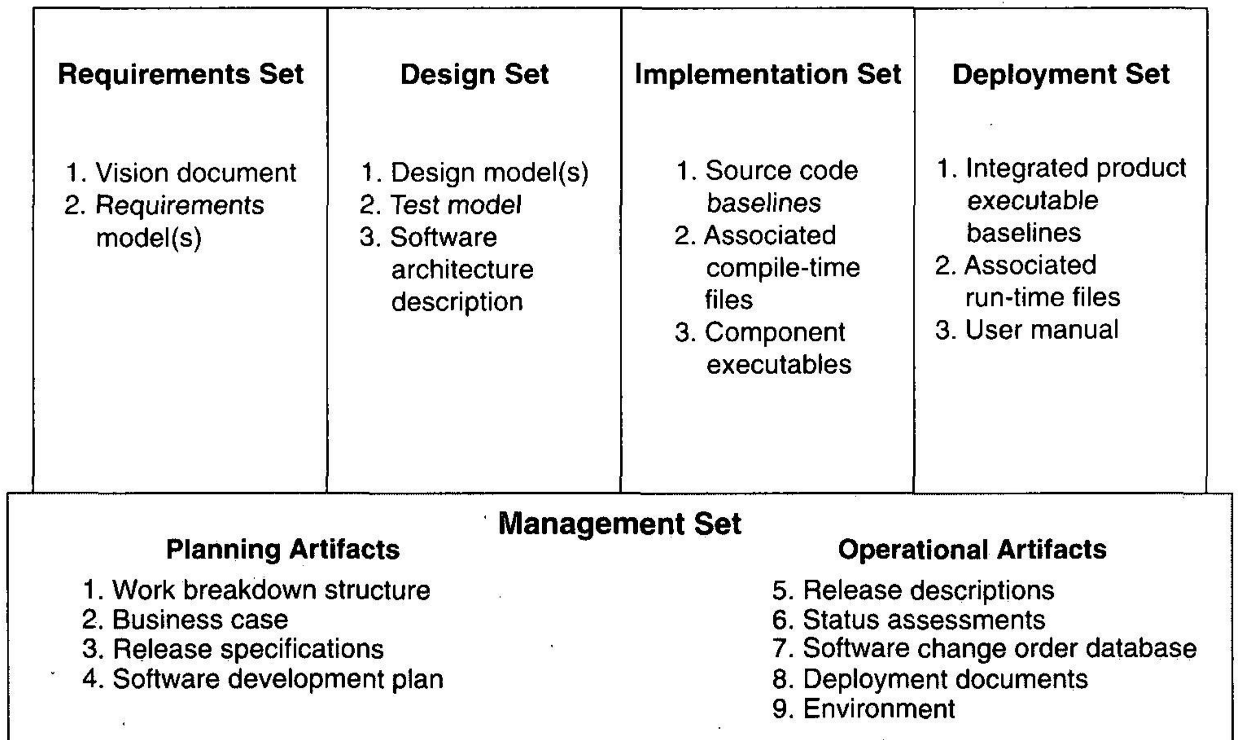
THE ARTIFACT SETS

In order to manage the development of a complete software system, we need to gather distinct collections of information and is organized into *artifact sets*.

- *Set* represents a complete aspect of the system where as *artifact* represents interrelated information that is developed and reviewed as a single entity.
- The artifacts of the process are organized into five sets:

1) Management 2) Requirements 3) Design
4) Implementation 5) Deployment

here the management artifacts capture the information that is necessary to synchronize stakeholder expectations. Where as the remaining four artifacts are captured rigorous notations that support automated analysis and browsing**.**

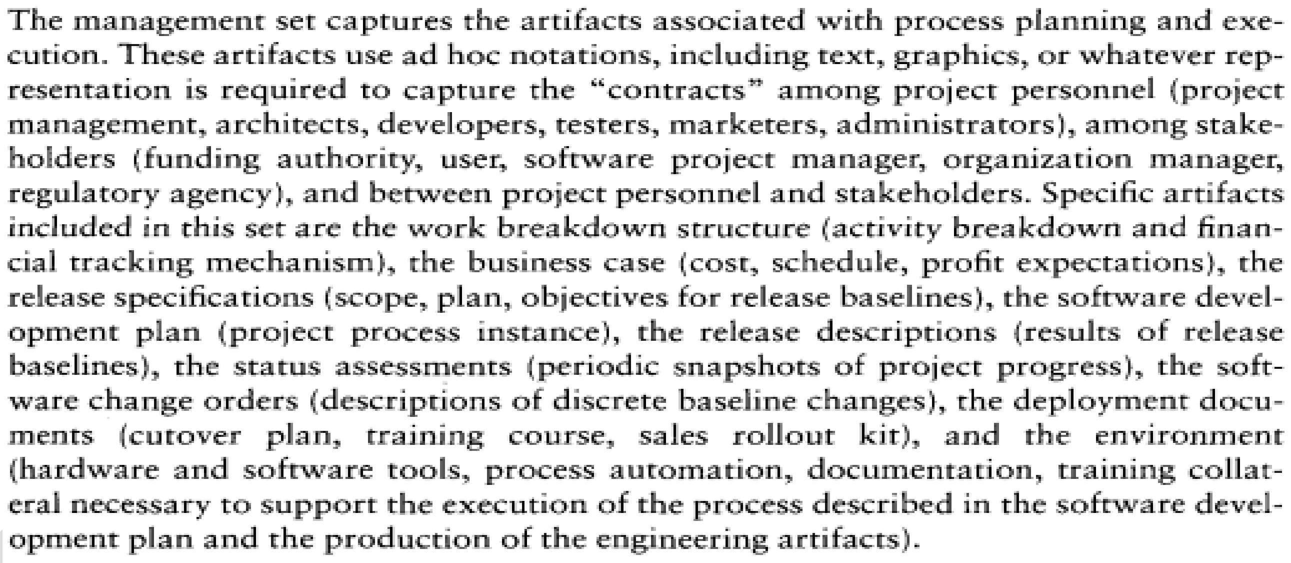


Requirements Set	Design Set	Implementation Set	Deployment Set
1. Vision document 2. Requirements model(s)	1. Design model(s) 2. Test model 3. Software architecture description	1. Source code baselines 2. Associated compile-time files 3. Component executables	1. Integrated product executable baselines 2. Associated run-time files 3. User manual

Management Set

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

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, training collateral necessary to support the execution of the process described in the software development plan and the production of the engineering artifacts).

It captures the artifacts associated with process planning and execution. These artifacts use ad hoc notation including text, graphics, or whatever representation is required to capture the "contracts" among,

- **project personnel:**

project manager, architects, developers, testers, marketers, administrators

- **stakeholders:**

funding authority, user, s/w project manager, organization manager, regulatory agency & between project personnel and stakeholders

Management artifacts are evaluated, assessed, and measured through a combination of

1) Relevant stakeholder review.
2) Analysis of changes between the current version of the artifact and previous versions.
3) Major milestone demonstrations of the balance among all artifacts.

THE ENGINEERING SETS:

1) REQUIREMENT SET:

- The requirements set is the primary engineering context for evaluating the other three engineering artifact sets and is the basis for test cases.
- **Requirement artifacts are evaluated, assessed, and measured through a combination of**

1) Analysis of consistency with the release specifications of the mgmt set.
2) Analysis of consistency between the vision and the requirement models.
3) Mapping against the design, implementation, and deployment sets to evaluate the consistency and completeness and the semantic balance between information in the different sets.
4) Analysis of changes between the current version of the artifacts and previous versions.
5) Subjective review of other dimensions of quality.

2) DESIGN SET:

- UML notations are used to engineer the design models for the solution.
- It contains various levels of abstraction and enough structural and behavioral information to determine a bill of materials.
- Design model information can be clearly and, in many cases, automatically translated into a subset of the implementation and deployment set artifacts.

The design set is evaluated, assessed, and measured through a combination of

1) Analysis of the internal consistency and quality of the design model.
2) Analysis of consistency with the requirements models.
3) Translation into implementation and deployment sets and notations to evaluate the consistency and completeness and semantic balance between information in the sets.
4) Analysis of changes between the current version of the design model and previous versions.
5) Subjective review of other dimensions of quality.

3) IMPLEMENTATION SET:

- The implementation set include source code that represents the tangible implementations of components and any executables necessary for stand-alone testing of components.

- Executables are the primitive parts that are needed to construct the end product, including custom components, APIs of commercial components.
- Implementation set artifacts can also be translated into a subset of the deployment set. Implementation sets are human-readable formats that are evaluated, assessed, and measured through a combination of

1) Analysis of consistency with design models
2) Translation into deployment set notations to evaluate consistency and completeness among artifact sets.
3) Execution of stand-alone component test cases that automatically compare expected results with actual results.
4) Analysis of changes b/w the current version of the implementation set and previous versions.
5) Subjective review of other dimensions of quality.

4) DEPLOYMENT SET:

- It 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

1) 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.
2) Testing the partitioning, replication, and allocation strategies in mapping components of the implementation set to physical resources of the deployment system.
3) Testing against the defined usage scenarios in the user manual such as installation, user-oriented dynamic reconfiguration, mainstream usage, and anomaly management.
4) Analysis of changes b/w the current version of the deployment set and previous

versions.

5) Subjective review of other dimensions of quality.

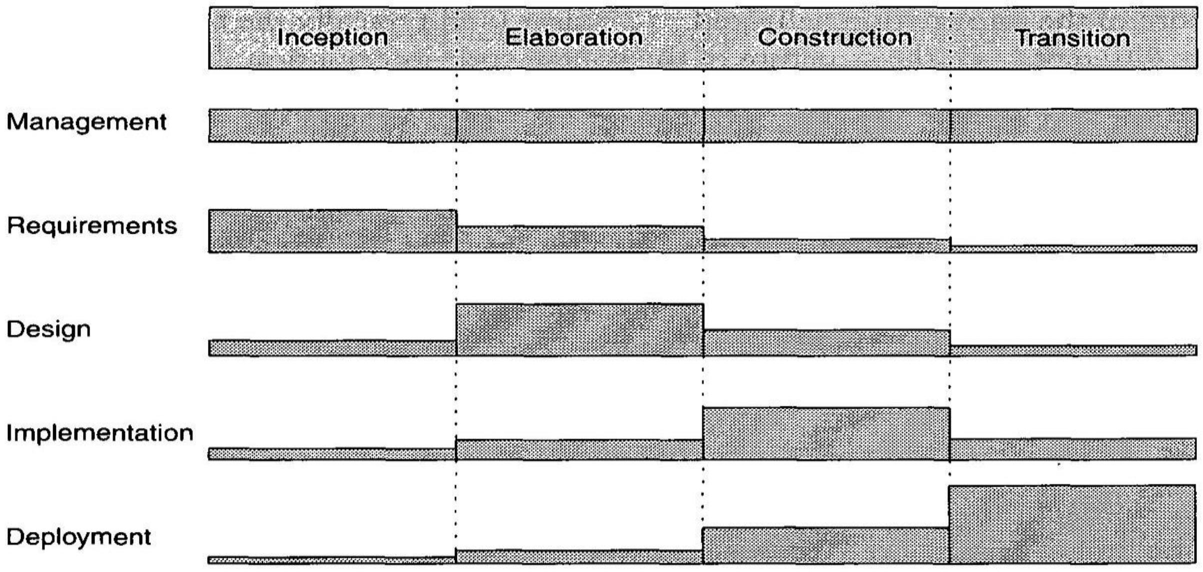
Each artifact set uses different notations to capture the relevant artifact.

1) **Management set notations** (ad hoc text, graphics, use case notation) capture the plans, process, objectives, and acceptance criteria.

2) **Requirement notation** (structured text and UML models) capture the engineering context and the operational concept.

3) **Implementation notations** (software languages) capture the building blocks of the solution in human-readable formats.

4) **Deployment notations** (executables and data files) capture the solution in machine-readable formats.



IMPLEMENTATION SET VERSUS DEPLOYMENT SET

- The structure of the information delivered to the user (testing people) is very different from the structure of the source code implementation.
- Engineering decisions that have impact on the quality of the deployment set but are relatively incomprehensible in the design and implementation sets include:

1) Dynamically reconfigurable parameters such as buffer sizes, color palettes, number of servers, number of simultaneous clients, data files, run-time parameters.

2) Effects of compiler/link optimizations such as space optimization versus speed

optimization.

3) Performance under certain allocation strategies such as centralized versus distributed, primary and shadow threads, dynamic load balancing.

4) Virtual machine constraints such as file descriptors, garbage collection, heap size, maximum record size, disk file rotations.

5) Process-level concurrency issues such as deadlock and race condition.

6) Platform-specific differences in performance or behavior.

ARTIFACTS EVOLUTION OVER THE LIFE CYCLE

- Each state of development represents a certain amount of precision in the final system description.
- Early in the lifecycle, precision is low and the representation is generally high. Eventually, the precision of representation is high and everything is specified in full detail.
- At any point in the lifecycle, the five sets will be in different states of completeness. However, they should be at compatible levels of detail and reasonably traceable to one another.
- Performing detailed traceability and consistency analyses early in the life cycle i.e. when precision is low and changes are frequent usually has a low ROI.

Inception phase: It mainly focuses on critical requirements, usually with a secondary focus on an initial deployment view, little implementation and high-level focus on the design architecture but not on design detail.

Elaboration phase: It include generation of an executable prototype, involves subsets of development in all four sets. A portion of all four sets must be evolved to some level of completion before an architecture baseline can be established.

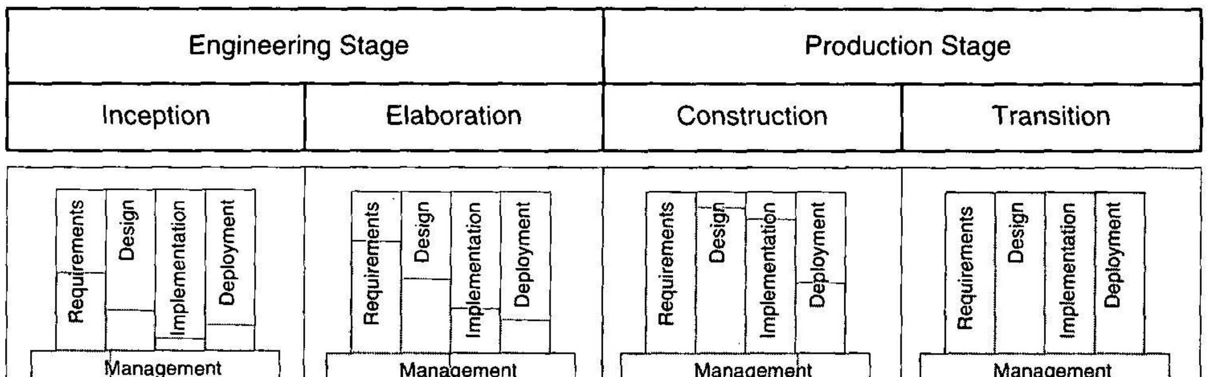


Fig: Life-Cycle evolution of the artifact sets

Construction: Its main focus on design and implementation. In the early stages the main focus is on the depth of the design artifacts. Later, in construction, realizing the design in source code and individually tested components. This stage should drive the requirements, design, and implementation sets almost to completion. Substantial work is also done on the deployment set, at least to test one or a few instances of the programmed system through alpha or beta releases.

Transition: The main focus is on achieving consistency and completeness of the deployment set in the context of another set. Residual defects are resolved, and feedback from alpha, beta, and system testing is incorporated.

TEST ARTIFACTS:

Testing refers to the explicit evaluation through execution of deployment set components under a controlled scenario with an expected and objective outcome.

- What ever the document-driven approach that was applied to software development is also followed by the software testing people.
- Development teams built requirements documents, top-level design documents, and detailed design documents before constructing any source files or executable files.
- In the same way test teams built system test plan documents, unit test plan documents, and unit test procedure documents before building any test drivers, stubs, or instrumentation.
- This document-driven approach caused the same problems for the test activities that it did for the development activities.
- One of the truly tasteful belief of a modern process is to use exactly the same sets, notations, and artifacts for the products of test activities as are used for product development.
- The test artifacts must be developed concurrently with the product from inception through deployment. *i.e. Testing 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.
- Testing is only one aspect of the evaluation workflow. Other aspects include inspection, analysis, and demonstration.
- The success of test can be determined by comparing the expected outcome to the actual outcome with well-defined mathematical precision.

MANAGEMENT ARTIFACTS:

Work Breakdown Structure

- Development of WBS is dependent on product management style , organizational culture, custom performance, financial constraints and several project specific parameters.
- The WBS is the architecture of project plan. It encapsulate change and evolve with appropriate level of details.
- A WBS is simply a hierarchy of elements that decomposes the project plan into discrete work task.
- A WBS provides the following information structure
- - A delineation of all significant tasks.
- - A clear task decomposition for assignment of responsibilities.
- - A framework for scheduling ,debugging and expenditure tracking.
- -Most systems have first level decomposition subsystem. subsystems are then decomposed into their components
- Therefore WBS is a driving vehicle for budgeting and collecting cost.
- The structure of cost accountability is a serious project planning constraints.

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. In large contractual procurements, the business case may be implemented in a full-scale proposal with multiple volumes of information. In a small-scale endeavor for a commercial product, it may be implemented in a brief plan with an attached spreadsheet. 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.

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*

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

There are two important forms of requirements. The first is the vision statement (or user need), which captures the contract between the development group and the buyer. This information should be evolving, but varying slowly, across the life cycle. It should be represented in a form that is understandable to the buyer (an ad hoc format that may include text, mockups, use cases, spreadsheets, or other formats). A use case model in the vision statement context serves to capture the operational concept in terms the user/buyer will understand.

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 6-5. *Typical release specification outline*

Software Development Plan

The software development plan (SDP) elaborates the process framework into a fully detailed plan. It is the defining document for the project's process. It must comply with the contract (if any), comply with organization standards (if any), evolve along with the design and requirements, and be used consistently across all subordinate organizations doing software development. Two indications of a useful SDP are peri-

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 6-6 *Typical software development plan 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. This document should also include a metrics summary that quantifies its quality in absolute and relative terms (compared to the previous versions, if any). The results of a post-mortem review of any release would be documented here, including outstanding issues, recommendations for process and product improvement, trade-offs in addressing evaluation criteria, follow-up actions, and similar information. Figure 6-7 provides a default outline for a release description.

Software Change Order Database :

- Managing change is one of the fundamental primitives of an iterative development process.
- This flexibility increases the content, quality, and number of iterations that a project can achieve within a given schedule.
- Once software is placed in a controlled baseline, all changes must be formally tracked and managed.
- Most of the change management activities can be automated by automating data entry and maintaining change records online.

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.

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. Although the period may vary, the forcing function needs to persist. The paramount objective of a good management process is to ensure that the expectations of all stakeholders (contractor, customer, user, subcontractor) are synchronized and consistent. The periodic status assessment documents provide the critical mechanism for managing everyone's expectations throughout the life cycle; for addressing, communicating, and resolving management issues, technical issues, and project risks; and for capturing project history. They are the periodic heartbeat for management attention.

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, and follow-through. Continuous open communications with objective data derived directly from on-going activities and evolving product configurations are mandatory in any project.

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, integrated change management, and defect tracking. A common theme from successful software projects is that they hire good people and provide them with good tools to accomplish their jobs. Automation of the software development process provides payback in quality, the ability to estimate costs and schedules, and overall productivity using a smaller team. By allowing the designers to traverse quickly among development artifacts and easily keep the artifacts up-to-date, integrated toolsets play an increasingly important role in incremental and iterative development.

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. Some artifacts are updated at each major milestone, others at each minor milestone. 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			▲	▲	▲	▲	▲

ENGINEERING ARTIFACTS

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. Whether the project is a huge military-standard development (whose vision could be a 300-page system specification) or a small, internally funded commercial product (whose vision might be a two-page white paper), every project needs a source for capturing the expectations among stakeholders. 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. The architecture can be described using a subset of the design model or as an abstraction of the design model with supplementary material, or a combination of both. As examples of these two forms of descriptions, consider the architecture of this book:

- A subset form could be satisfied by the table of contents. This description of the architecture of the book is directly derivable from the book itself.
- An abstraction form could be satisfied by a "Cliffs Notes" treatment. (Cliffs Notes are condensed versions of classic books used as study guides by some college students.) This format is an abstraction that is developed separately and includes supplementary material that is not directly derivable from the evolving product.

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. If the test team is responsible for the manual, it can be generated in parallel with development and can be evolved early as a tangible and relevant perspective of evaluation criteria. It also provides a necessary basis for test plans and test cases, and for construction of automated test suites.

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

PRAGMATIC ARTIFACTS

Conventional document-driven approaches squandered incredible amounts of engineering time on developing, polishing, formatting, reviewing, updating, and distributing documents. Why? There are several reasons that documents became so important to the process. First, there were no rigorous engineering methods or languages for requirements specification or design. Consequently, paper documents with ad hoc text and graphical representations were the default format. Second, conventional languages of implementation and deployment were extremely cryptic and highly unstructured. To present the details of software structure and behavior to other interested reviewers (testers, maintainers, managers), a more human-readable format was needed. Probably most important, software progress needed to be "credibly" assessed. Documents represented a tangible but misleading mechanism for demonstrating progress.

This philosophy raises the following cultural issues:

- **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

"I am not going to learn UML, but I am going to review design"

- **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

Organizations are forced toexchange paper documents

- **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

Use proper english words that enables understandable representations,browsable formats and reduced error rates

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

Avoid separate documents to describe all the details of a model, component or test procedure

If a document is produced but not used, eliminate it.

- **Paper is tangible; electronic artifacts are too easy to change.**

Paper documents are tangible, static and persistant. Online and web based artifacts can be changed easily and are viewed with more scepticism

I Support change management, electronic signature which replaces paper.

Short documents are usually more useful than long ones. Software is primary product, documentation is support material.