UNIT-III
PART - B

Artifacts of the Process: The Artifact Sets, Management Artifacts, Engineering Artifacts, and Pragmatic Artifacts.

Introduction

- Conventional software projects focused on the sequential development of software artifacts:

➢ Build the requirements

➢ Construct a design model traceable to the requirements

➢ Build an implementation traceable to the design model

➢ Compile and test the implementation for development

The Artifact Sets

- To make the development of a complete software system manageable, distinct collections of information are organized into artifacts sets.
- Each set comprises related artifacts that are persistent and in a uniform representation format.
- While a set represents a complete aspect of the system, **an artifact** represents cohesive information that are typically is developed and reviewed as a single entity.

The Artifact Sets

- 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 files)
  - Deployment (machine-processable language and associated files)

Overview of the artifact sets

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 2. Business case 3. Release specifications 4. Software development plan	5. Release descriptions 6. Status assessments 7. Software change order database 8. Deployment documents 9. Environment

The Management Set

- The management set captures the artifacts associated with process planning and execution.
- These artifacts use adhoc 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.

The Management Set

- Management set artifacts are evaluated, assessed, and measured through a combination of the following:
  - Relevant stakeholders review
  - Analysis of changes between the current version of the artifact and previous versions
  - Major milestone demonstrations of the balance among all artifacts and the accuracy of the business case and vision artifacts

The Engineering Sets

- The engineering sets consisting of
  - The requirements set
  - The design set
  - The implementation set
  - The deployment set
- The primary mechanism for evaluating the evolving quality of each artifact set is the transitioning of information from set to set, there by maintaining a balance of understanding among the requirements, design, implementation, and deployment artifacts.

Requirements set

- Structured text is used for the vision statement, which documents the project scope that supports the contract between the funding authority and the project team.
- UML notation is used for engineering representations of requirements models (use case models, domain models).
- **The requirements set is the primary engineering context for evaluating the other three engineering artifact sets and is the basis for test cases.**

Requirements set

- Requirements artifacts are evaluated, assessed, and measured through a combination of the following:
  - Analysis of consistency with the release specification 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
  - Subjective review of the 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 represents the component 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 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
  - Subjective review of other dimensions of quality

Implementation set

- The implementation set includes source code that represents the tangible implementation of components and any executables necessary for stand-alone testing of components.
- 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 to evaluate the consistency and completeness among artifacts 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 result
  - Analysis of changes between the current version of the implementation set and previous versions
  - 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.
- These machine language notations represents the product components in the target from intended for distribution to users.

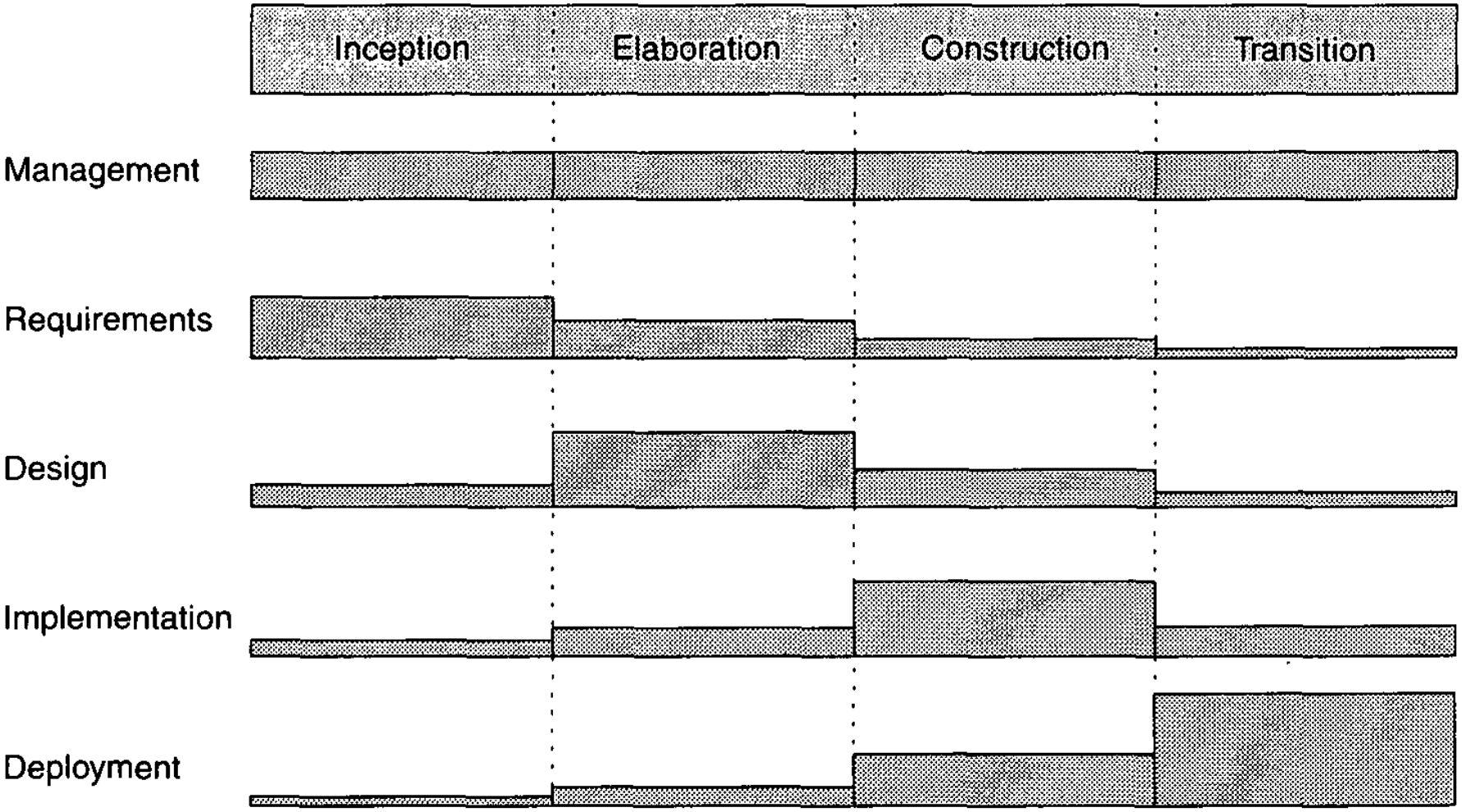
Deployment set

- 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
  - Testing against the defined usage scenarios in the user manual such as installation, user-oriented dynamic reconfiguration, mainstreams usage , and anomaly management
  - Analysis of changes between the current version of the deployment set and previous versions.
  - Subjective review of other dimensions of quality

Deployment set

- 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 shown in below figure, 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 a fairly constant level across the life cycle.

Life-cycle focus on artifact sets



Deployment set

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

- **Management:** scheduling, workflow, defect tracking, change, management, documentation, spreadsheet, resources management and presentation tools
- **Requirements:** requirements management tools
- **Design:** visual modelling tools
- **Implementation:** complier/debugger tools, code analysis tools, test coverage analysis tools and test management tools
- **Deployment:** test coverage and test automation tools, network management tools, commercial components (operating systems, GUIs ,DBMSs, networks ,middleware) , and installation tools

Overview of the artifact sets

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 2. Business case 3. Release specifications 4. Software development plan	5. Release descriptions 6. Status assessments 7. Software change order database 8. Deployment documents 9. Environment

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.
- Below figure provides a default outline for a business case.

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

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 and understanding and acceptance by managers and practitioners alike.
- Below figure provides a default outline for a software development plan.

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)

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.
- Below figure provides a default outline for a release specification.

Release Specifications

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

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.
- Below figure provides a default outline for a release description.

Release Descriptions

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

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, and follow-through.

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.

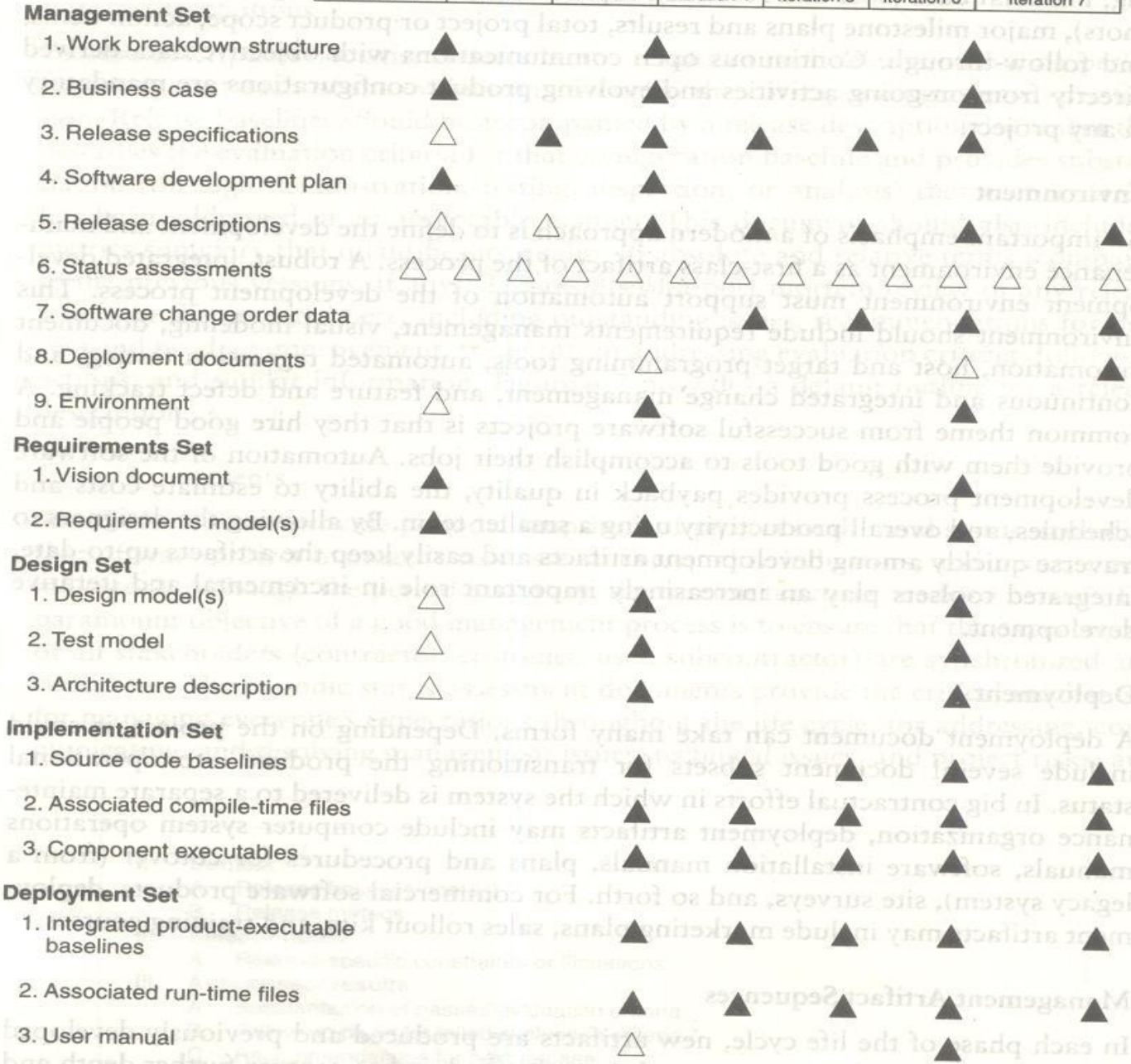
Management Artifact Sequences

- 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.
- Below figure 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



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.

1. 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.
- Below figure provides a default outline for a vision document.
- The vision statement should include a description of what will be included as well as those features considered but not included.

Typical vision document outline

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)

2. 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.
- Below figure provides a default outline for an architecture description.

Typical architecture description outline

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**

3. 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.

Pragmatic Artifacts

- **Pragmatic Artifacts** is the conventional document-driven approaches squandered incredible amounts of engineering time on developing, polishing, formatting, reviewing, updating, and distributing documents.
- It is an approach that redirects this effort of documentation to simply improve rigor and easily understanding of source of data or information.
- With use of smart browsing and navigation tools, it also allows an on-line review of source of the native information.
- This idea increases various cultural issues. Some of them are given below.

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 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".

Pragmatic Artifacts

- **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.

Pragmatic Artifacts

- **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 abbreviation should be used only where they are well-accepted jargon in the context of the component's usage.
  - No matter what language or tools are used, there is no reason to abbreviate and encrypt modelling or programming language source identifies.

Pragmatic Artifacts

- **Useful documentation is self-defining: It is documentation that gets used**
  - Above all, building self-documenting engineering artifacts gives the development organization the "right" to work solely in the engineering notations and avoid separate documents to describe all the details of a model, component, or test procedure.

Pragmatic Artifacts

- **Paper is tangible; electronic artifacts are too easy to change**
  - One reason some stakeholders prefer paper documents is that once they are delivered, they are tangible, static, and persistent.
  - On-line and Web-based artifacts can be changed easily and are viewed with more skepticism because of their inherent volatility.