ESPM UNIT II

The Old way and the NEW way: Principles of Conventional Software Engineering, Principles of Modern Software Management, Transitioning to an Iterative Process.
Life Cycle Phases: Engineering and Production Stages, Inception, Elaboration, Construction, Transition Phases.
Artifacts of the Process: The Artifact Sets. Management Artifacts, Engineering Artifacts, Programmatic Artifacts.

PRINCIPLES OF CONVENTIONAL SOFTWARE ENGINEERING

Based on many years of software development experience, the software industry proposed so many principles (nearly 201 by – Davis‟s). Of which Davis‟s top 30 principles are:

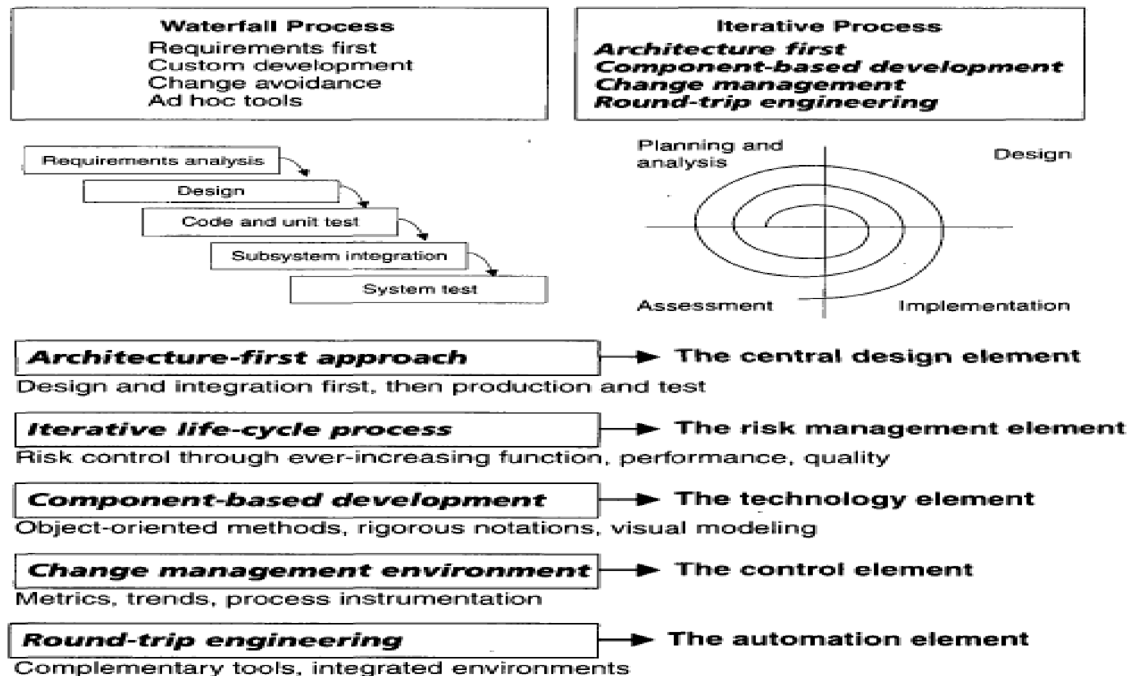
1. **Make quality #1:** Quality must be quantified and mechanisms put into place to motivate its achievement.
2. **High-quality software is possible:** Techniques that have been demonstrated to increase quality include involving the customer, prototyping, simplifying design, conducting inspections, and hiring the best people.
3. **Give products to customers early:** No matter how hard you try to learn users needs during the requirements phase, the most effective way to determine real needs is to give users a product and let them play with it.
4. **Determine the problem before writing the requirements:** When faced with what they believe is a problem, most engineers rush to offer a solution. Before you try to solve a problem, be sure to explore all the alternatives and don't be blinded by the obvious solution.
5. **Evaluate Design Alternatives:** After the requirements are agreed upon, you must examine a variety of architectures and algorithms. You certainly do not want to use "architecture" simply because it was used in the requirements specification.
6. **Use an appropriate process model:** Each project must select a process that makes the most sense for that project on the basis of corporate culture, willingness to take risks, application area, volatility of requirements, and the extent to which requirements are well understood.
7. **Use different languages for different phases:** Our industry's eternal thirst for simple solutions to complex problems has driven many to declare that the best development method is one that uses the same notation throughout the life cycle.
8. **Minimize Intellectual Distance:** To minimize intellectual distance, the software's structure should be as close as possible to the real –world structure.
9. **Put techniques before tools:** An undisciplined software engineer with a tool becomes a dangerous, undisciplined software Engineer.
10. **Get it right before you make it faster:** It is far easier to make a working program run faster than it is to make a fast program work. Don't worry about optimization during initial coding.
11. **Inspect Code:** Inspecting the detailed design and code is a much better way to find errors than testing.
12. **Good Management is more important than good technology:** Good management motivates people to do their best, but there are no universal "right" styles of management.
13. **People are the key to success:** Highly skilled people with appropriate experience, talent, and training are key.
14. **Follow with Care:** Just because everybody is doing something does not make it right for you. It may be right, but you must carefully assess its applicability to your environment.
15. **Take responsibility:** When a bridge collapses we ask, "What did the engineers do wrong?" Even when software fails, we rarely ask this. The fact is that in any engineering discipline, the best methods can be used to produce awful designs, and the most antiquated methods to produce elegant designs
16. **Understand the customer's priorities:** It is possible the customer would tolerate 90% of the functionality delivered late if they could have 10% of it on time.
17. **The more thy see, the more they need:** The more functionality (or performance) you provide a user, the more functionality (or performance) the user wants.
18. **Plan to throw one away:** One of the most important critical success factors is whether or not a product is entirely new. Such brand-new applications, architectures, interfaces, or algorithms rarely work the first time.
19. **Design for Change:** The architectures, components and specification techniques you use must accommodate change.
20. **Design without documentation is not design:** I have often heard software engineers say, “I have finished the design. All that is left is the documentation.”

21. **Use tools, but be realistic:** Software tools make their users more efficient.
22. **Avoid tricks:** Many programmers love to crate programs with tricks constructs that perform a function correctly, but in an obscure way. Show the world how smart you are by avoiding tricky code.
23. **Encapsulate:** Information-hiding is a simple, proven concept that results in software that is easier to test and much easier to maintain.
24. **Use coupling and cohesion:** Coupling and cohesion are the best ways to measure software's inherent maintainability and adaptability.
25. **Use the McCabe complexity measure:** Although there are many metrics available to report the inherent complexity of software, none is as intuitive and easy to use as Total McCabe‟s.
26. **Don't test your own software:** Software developers should never be the primary testers of their own software.
27. **Analyze causes for errors:** It is far more cost-effective to reduce the effect of an error by preventing it than it is to find and fix it. One way to do this is to analyze the causes of errors as they are detected.
28. **Realize that software's entropy increases:** Any software system that undergoes continuous change will grow in complexity and will become more and more disorganized.
29. **People and time are not interchangeable:** Measuring a project solely by person-months makes little sense.
30. **Expect Excellence:** Your employees will do much better if you have high expectations for them.

PRINCIPLES OF MODERN' SOFTWARE MANAGEMENT

Top 10 principles of modern software management are:

1. **Base the process on an architecture-first approach:** This requires that a demonstrable balance be achieved among the driving requirements, the architecturally significant design decisions, and the life-cycle plans before the resources are committed for full-scale development.
2. **Establish an iterative life-cycle process that confronts risk early that confronts risk early:** With today's sophisticated software systems, it is not possible to define the entire problem, design the entire solution, build the software, then test the end product in sequence. Instead, an iterative process that refines the problem understanding, an effective solution, and an effective plan over several iterations encourages a balanced treatment of all stakeholder objectives. Major risks must be addressed early to increase predictability and avoid expensive downstream scrap and rework.
3. **Transition design methods to emphasize component-based development:** Moving from a line-of-code mentality to a component-based mentality is necessary to reduce the amount of human-generated source code and custom development.



The top five principles of a modern process

4. **Establish a change Management Environment:** the dynamics of iterative development, including concurrent workflows by different teams working on shared artifacts, necessitates objectively controlled baselines.
5. **Enhance change freedom through tools that support round-trip Engineering:** Round trip engineering is the environment support necessary to automate and synchronize engineering information in different formats (such as requirements specifications, design models, source code, executable code, test cases).
6. **Capture design artifacts in rigorous, model-based notation:** A model based approach (such as UML) supports the evolution of semantically rich graphical and textural design notations.
7. **Instrument the process for objective quality control and progress assessment:** Life-cycle assessment of the progress and the quality of all intermediate products must be integrated into the process.
8. **Use a demonstration-based approach:** to assess intermediate artifacts.
9. **Plan intermediate releases in groups of usage scenarios with evolving levels or detail:** It is essential that the software management process drive toward early and continuous demonstrations within the operational context of the system, namely its use cases.
10. **Establish a configurable process that is economically scalable:** No single process suitable for all software developments.

Modern process approaches for solving conventional problems

CONVENTIONAL PROCESS: TOP 10 RISKS	IMPACT	MODERN PROCESS: INHERENT RISK RESOLUTION FEATURES
1. Late breakage and excessive scrap/rework	Quality, cost, schedule	Architecture-first approach Iterative development Automated change management Risk-confronting process
2. Attrition of key personnel	Quality, cost, schedule	Successful, early iterations Trustworthy management and planning
3. Inadequate development resources	Cost, schedule	Environments as first-class artifacts of the process Industrial-strength, integrated environments Model-based engineering artifacts Round-trip engineering
4. Adversarial stakeholders	Cost, schedule	Demonstration-based review Use-case-oriented requirements/testing
5. Necessary technology insertion	Cost, schedule	Architecture-first approach Component-based development
6. Requirements creep	Cost, schedule	Iterative development Use case modeling Demonstration-based review
7. Analysis paralysis	Schedule	Demonstration-based review Use-case-oriented requirements/testing
8. Inadequate performance	Quality	Demonstration-based performance assessment Early architecture performance feedback
9. Overemphasis on artifacts	Schedule	Demonstration-based assessment Objective quality control
10. Inadequate function	Quality	Iterative development Early prototypes, incremental releases

TRANSITIONING TO AN ITERATIVE PROCESS

- Modern software development processes have moved away form the conventional waterfall model, in which each stage of the development process is dependent on completion of the previous stage.
- The economic benefits inherent in transitioning from the conventional waterfall model to an iterative development process are significant but difficult to quantify.

- As one benchmark of the expected economic impact of process improvement, consider the process exponent parameters of the COCOMO II mode. This exponent can range from 1.01 (virtually no diseconomy of scale) to 1.26 (significant diseconomy of scale).

The following paragraphs map the process exponent parameters of CO COMO II to my top 10 principles of a modern process:

1. **Application Precedentedness:** domain experience is a critical factor in understanding how to plan and execute a software development project. Early iterations in the life cycle establish precedents from which the product, the process and the plans can be elaborated in evolving levels of detail.
2. **Process flexibility:** Development of modern software is characterized by such a broad solution space and so many interrelated concerns that there is a paramount need for continuous incorporation of changes. A configurable process that allows a common framework to be adapted across a range of projects is necessary to achieve a software return on investment.
3. **Architecture Risk Resolution**: Architecture-first development is a crucial theme underlying a successful iterative development process. A project team develops and stabilizes architecture before developing all the components that make up the entire suite of applications components. An Architecture-first and component-based development approach forces tile infrastructure, common mechanisms, and control mechanisms to be elaborated early in the life cycle and drives all component make/buy decisions into the architectureprocess.
4. **Team Cohesion**: Successful teams are cohesive, and cohesive teams are successful. Successful teams and cohesive teams share common objectives and priorities. Advances in technology (such as programming languages, UML, and visual modeling) have enabled more rigorous and understandable notations for communicating software engineering information, particularly in the requirements and design artifacts that previously were ad hoc and based completely on paper exchange. These model-based formats have also enabled the round-trip engineering support needed to establish change freedom sufficient for evolving design representations.
5. **Software Process Maturity**: The Software Engineering Institute's Capability Maturity Model (CMM) is a well-accepted benchmark for software process assessment. One of key themes is that truly mature processes are enabled through an integrated environment that provides the appropriate level of automation to instrument the process for objection quality control.

LIFE CYCLE PHASES

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

Engineering and Production Stages

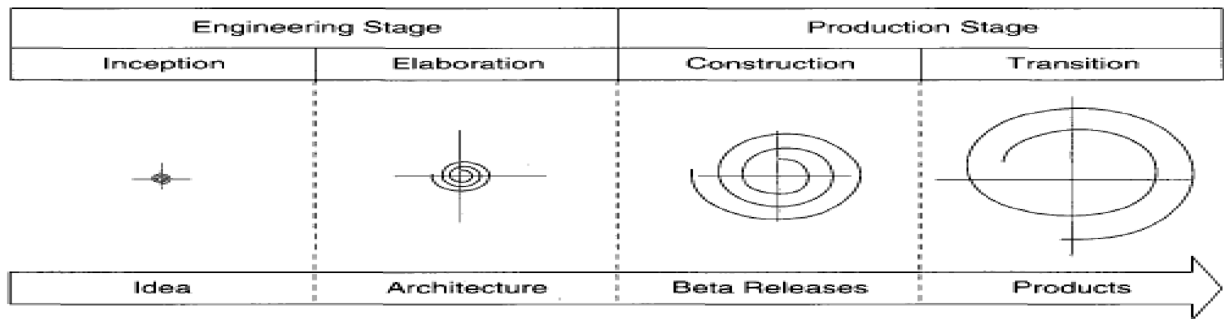
To achieve economies of scale and higher returns on investment, we must move toward a software manufacturing process driven by technological improvements in process automation and component based development. Two stages of the life cycle are:

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

The two stages of the life cycle: engineering and production

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

- The transition between engineering and production is a crucial event for the various stakeholders. The production plan has been agreed upon, and there is a good enough understanding of the problem and the solution that all stakeholders can make a firm commitment to go ahead with production.
- Engineering stage is decomposed into two distinct phases, inception and elaboration, and the production stage into construction and transition. These four phases of the life-cycle process are loosely mapped to the conceptual framework of the spiral model as shown in Figure:



The phases of the life-cycle process

INCEPTION PHASE

- The goal of this 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?
- Are actual resource expenditures versus planned expenditures acceptable?

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 reasonabletime

❑ **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?

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?
- Is this product baseline stable enough to be deployed in the user community?
- Are the stakeholders ready for transition to the user community?
- Are actual resource expenditures versus planned expenditures acceptable?

TRANSITION PHASE

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

1. Beta testing to validate the new system against user expectations.
2. Beta testing and parallel operation relative to a legacy system it is replacing.
3. Conversion of operational databases.
4. Training of user and maintainers- 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 Assessment of deployment baselines against the complete vision and acceptance criteria in the requirements set.

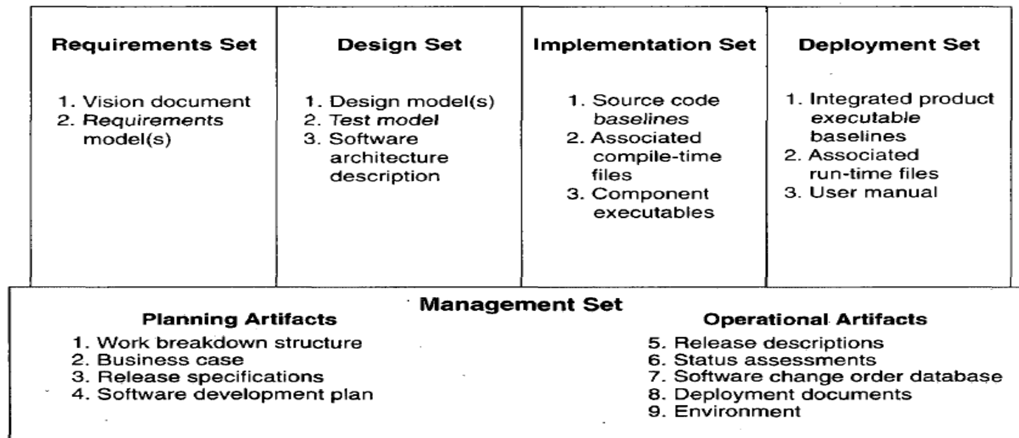
❑ **Primary Evaluation Criteria**

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

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:
  1. Management (ad hoc textual formats),
  2. Requirements (organized text and models of the problem space,
  3. Design (models of the solution space),
  4. Implementation (human-readable programming, language and associated source files), and
  5. Deployment (machine-process able languages and associate files).

The artifact sets are shown in the following figure:



Overview of the artifact sets

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

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.

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:

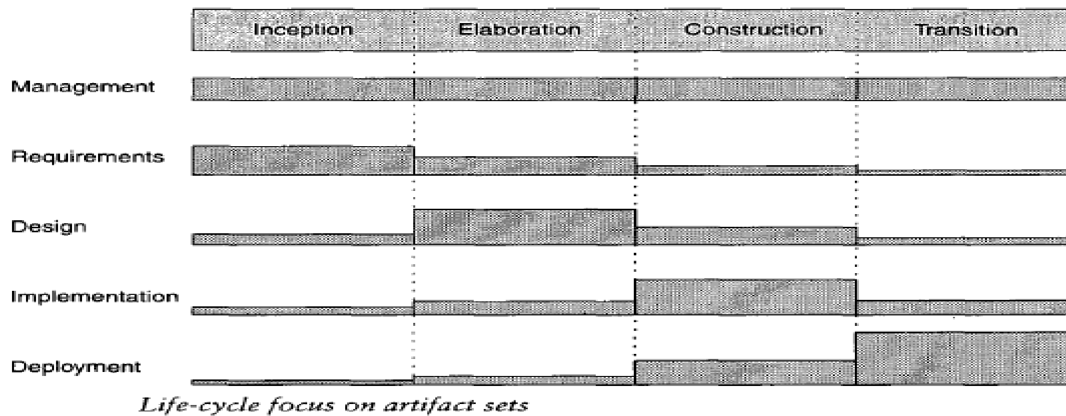
- Analysis of the internal consistency and quality of the design model
- Analysis of consistency with the requirements models
- Translation into implementation and deployment sets and notations (for example, traceability, source code generation, compilation, linking) to evaluate the consistency and completeness 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.



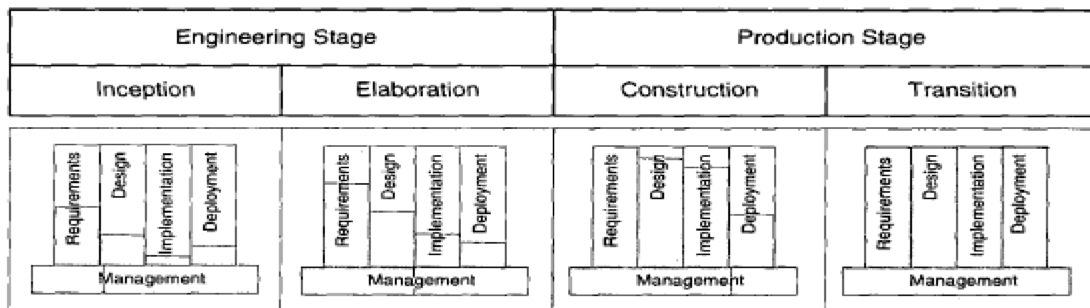
Life-cycle focus on artifact sets

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 (OS, GUIs, RDBMS, networks, middleware), and installation tools.

Artifact Evolution over the Life Cycle

Each state of development represents a certain amount of precision in the final system description. Early in the lif e 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 following figure:



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.

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.
    - **Management Set**: The release specifications and release descriptions capture the objectives, evaluation criteria, and results of an intermediate milestone.
    - **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.
    - **Design Set**: A test model for non deliverable components needed to test the product baselines is captured in the design set.
    - **Implementation Set**: Self-documenting source code representations for test components and test drivers provide the equivalent of test procedures and test scripts.
    - **Deployment Set:** Executable versions of test components, test drivers, and data files are provided.

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.

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

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.

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

Typical business case 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.

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

Typical release specification 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.

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)

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.

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

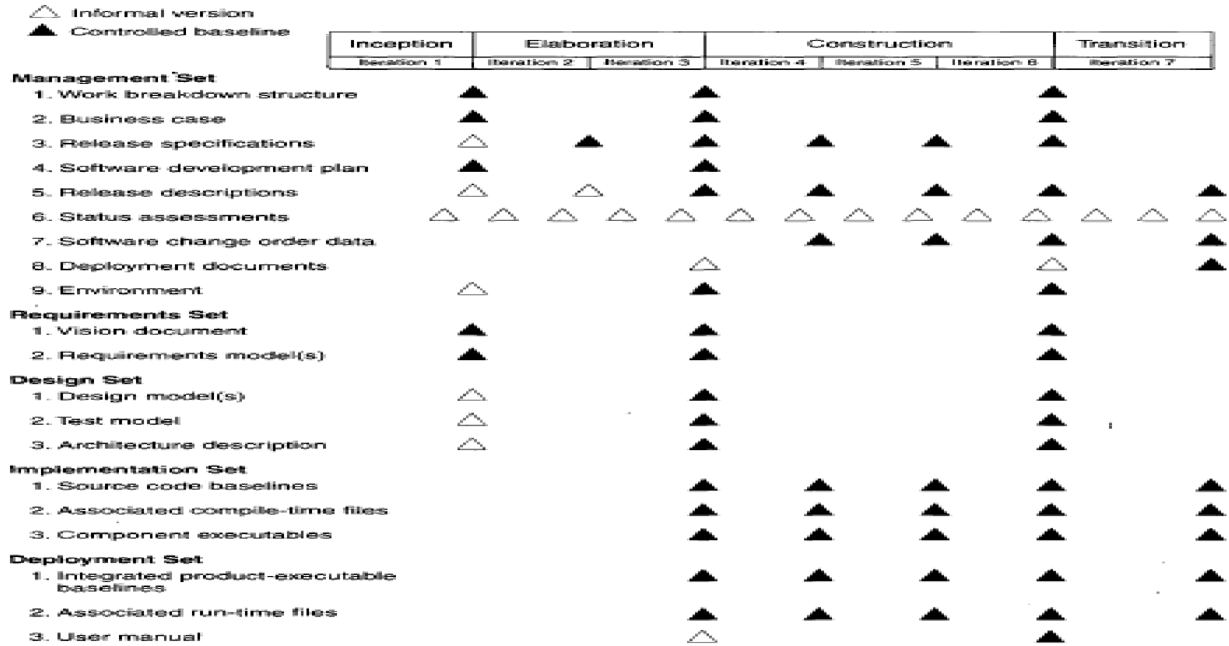
Typical release description outline

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. The following figure identifies a typical sequence of artifacts across the life-cycle phases.



Artifact sequences across a typical life cycle

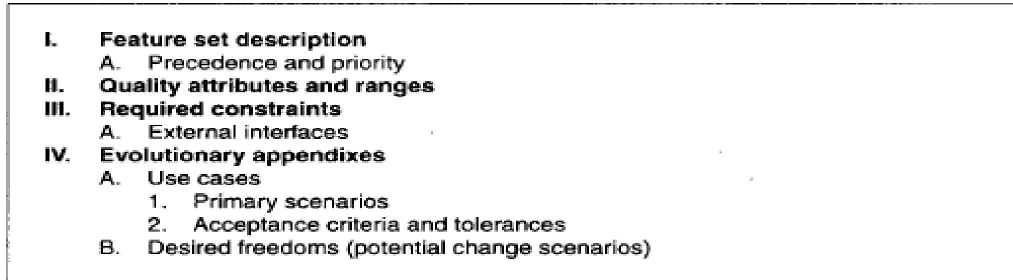
4.3. ENGINEERING ARTIFACTS

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

Vision document

- The vision document provides a complete vision for the software system under development and supports the contract between the funding authority and 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.

The following figure 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)

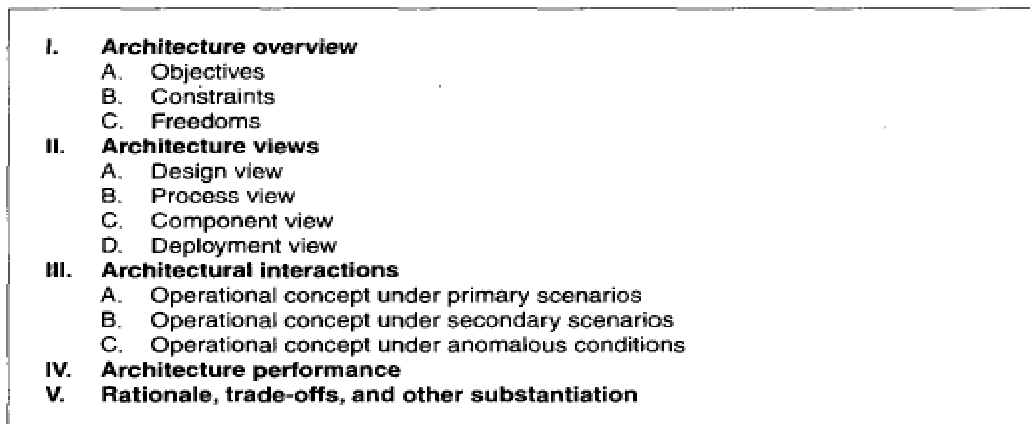
Typical vision document outline

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. The following figure provides a default outline form an architecture description.

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.



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**

Typical architecture description outline

PRAGMATIC ARTIFACTS

a) **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."
b) **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.
c) **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.
d) **Useful documentation is self-defining:** It is documentation that gets used.
e) **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.