Unit II: Project Approach

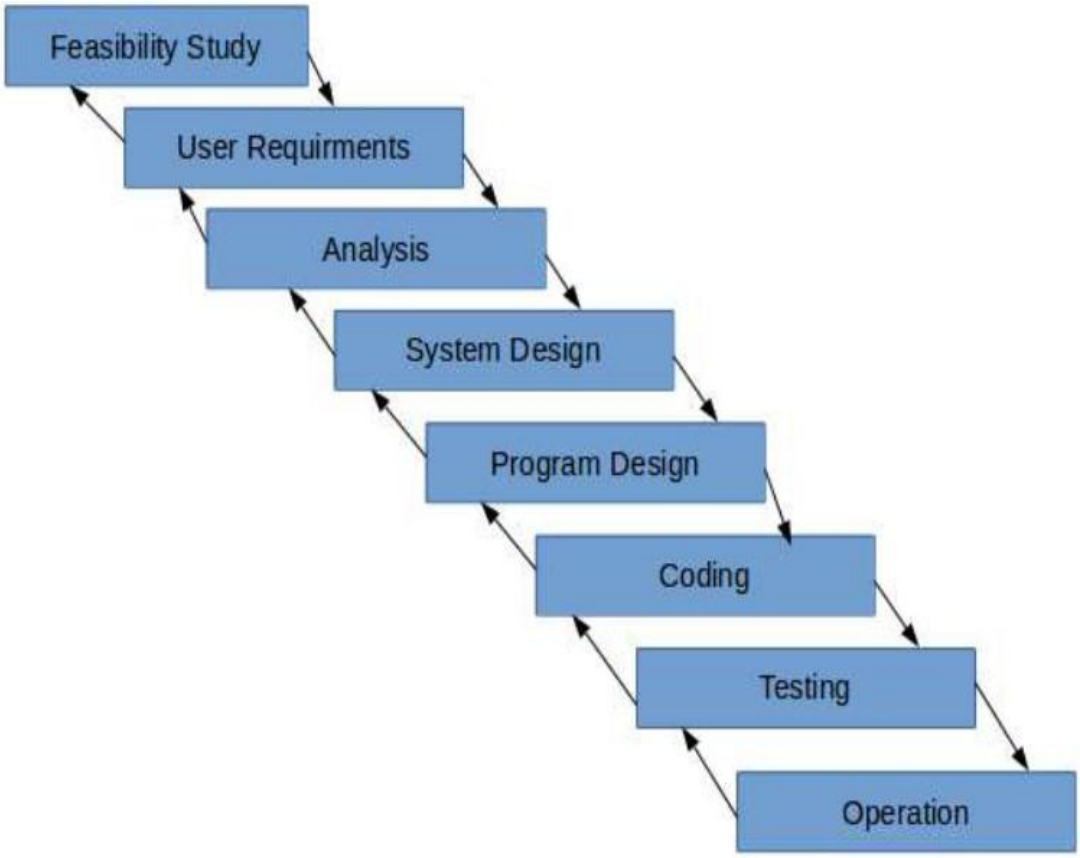
Lifecycle models, Choosing Technology, Prototyping Iterative & incremental Process Framework: Lifecycle phases, Process Artifacts, Process workflows.

Part-1

1. Lifecycle models

Waterfall Model
V Process Model
Spiral Model
Prototyping

1.1. The Waterfall Model:



It is also referred to as a **linear-sequential life cycle model**. It is very simple to understand and use. In a waterfall model, each phase must be completed fully before the next phase can begin. This type of model is basically used for the for the project which is small and there are no uncertain requirements. At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project. In this model the testing starts only after the development is complete. In **waterfall model phases** do not overlap.

Advantages of waterfall model:

- This model is simple and easy to understand and use.
- It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.

- In this model phases are processed and completed one at a time. Phases do not overlap.
- Waterfall model works well for smaller projects where requirements are very well understood.

Disadvantages of waterfall model:

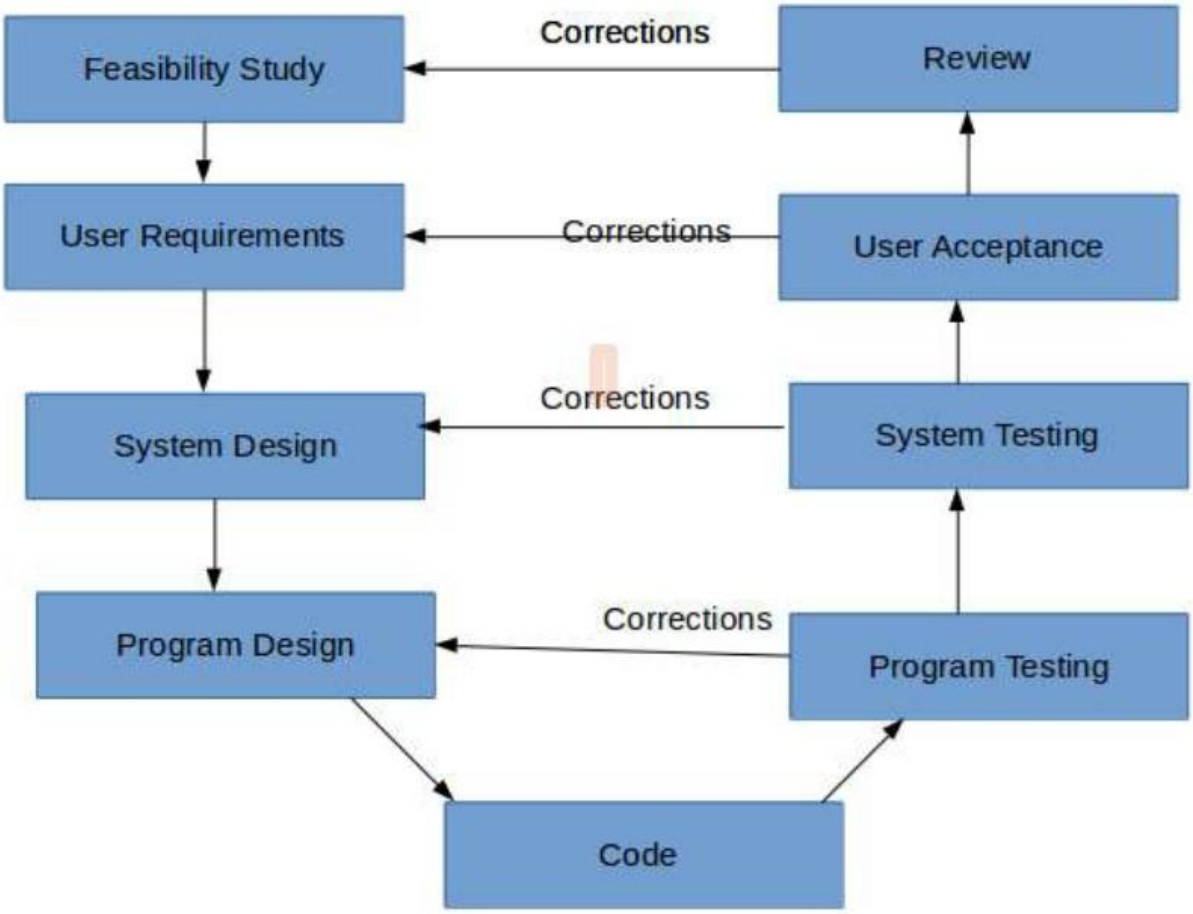
- Once an application is in the **testing** stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.

When to use the waterfall model:

- This model is used only when the requirements are very well known, clear and fixed.
- Product definition is stable.
- Technology is understood.

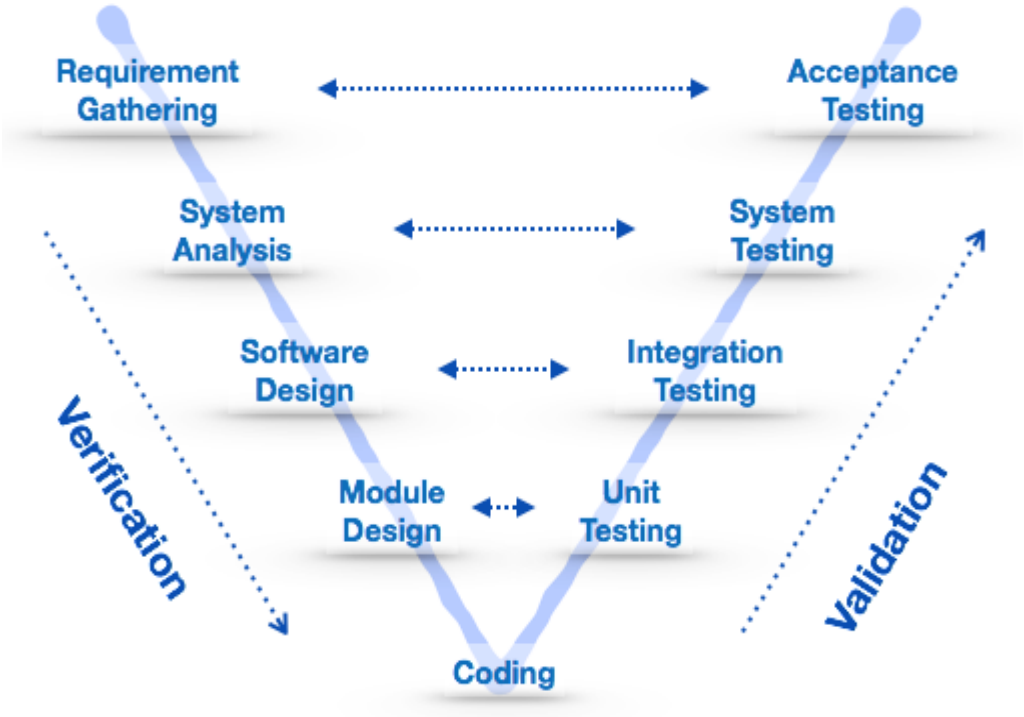
There are no ambiguous requirements

- Ample resources with required expertise are available freely
- The project is short.



1.2. <u>The V Process Model:</u>

The major drawback of waterfall model is we move to the next stage only when the previous one is finished and there was no chance to go back if something is found wrong in later stages. V-Model provides means of testing of software at each stage in reverse manner.



At every stage, test plans and test cases are created to verify and validate the product according to the requirement of that stage. For example, in requirement gathering stage the test team prepares all the test cases in correspondence to the requirements. Later, when the product is developed and is ready for testing, test cases of this stage verify the software against its validity towards requirements at this stage.

This makes both verification and validation go in parallel. This model is also known as verification and validation model.

<u>Advantages of V-model:</u>

- Simple and easy to use.
- Testing activities like planning, test designing happens well before coding. This saves a lot of time.

Hence higher chance of success over the waterfall model.

- Proactive defect tracking – that is defects are found at early stage.
- Avoids the downward flow of the defects.
- Works well for small projects where requirements are easily understood.

<u>Disadvantages of V-model:</u>

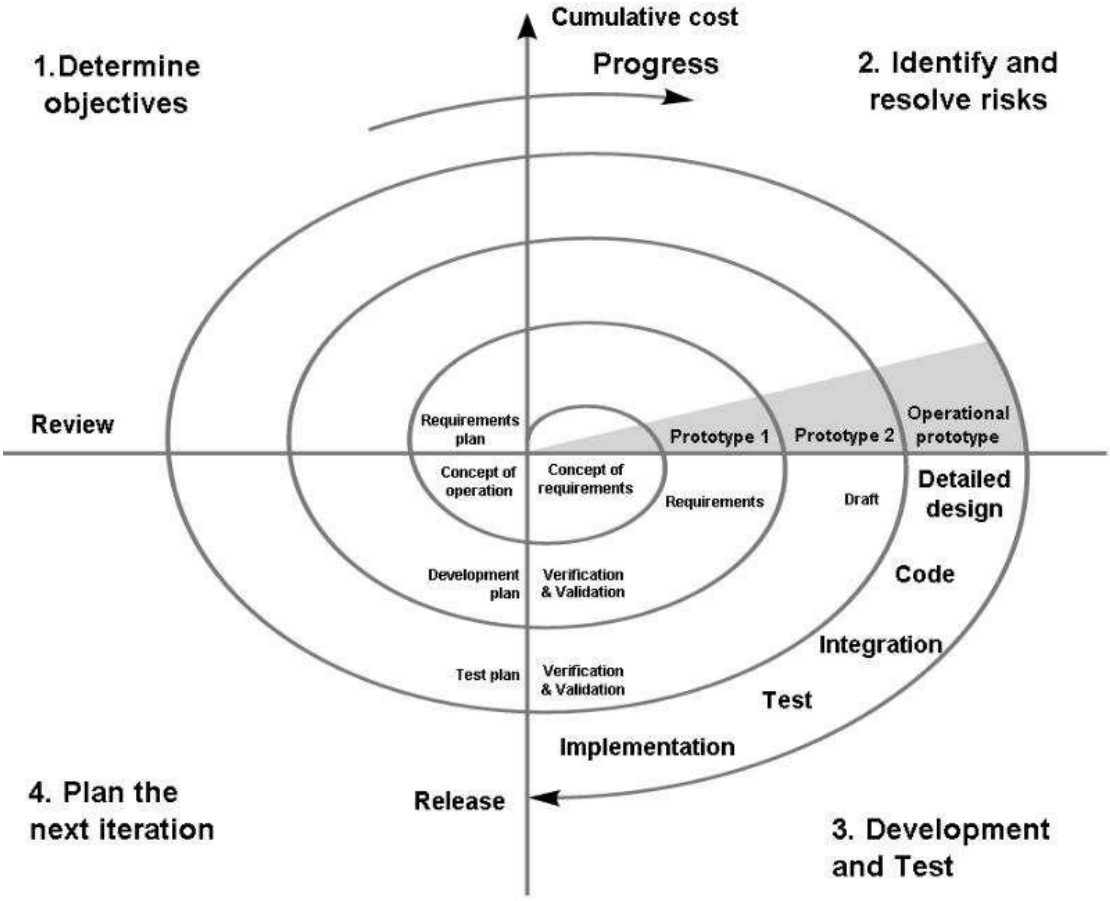
- Very rigid and least flexible.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.

- If any changes happen in midway, then the test documents along with requirement documents has to be updated.

When to use the V-model:

- The V-shaped model should be used for small to medium sized projects where requirements are clearly defined and fixed.
- The V-Shaped model should be chosen when ample technical resources are available with needed technical expertise.

1.3. The Spiral Model:



Spiral model is a combination of iterative development process model and Sequential linear development model i.e. waterfall model with very high emphasis on risk analysis. It allows for incremental releases of the product, or incremental refinement through each iteration around the spiral.

Advantages of Spiral model:

- High amount of risk analysis.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the **software life cycle**.

Disadvantages of Spiral model:

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project‘s success is highly dependent on the risk analysis phase.
- Doesn‘t work well for smaller projects.

When to use Spiral model:

- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex
- New product line

2. Choosing Technology:

As well as the products and activities, the choosen technology will affect:

- The training requirements for development staff.
- The types of staff to be recruited.
- The development environment – both hardware and software.
- System maintenance arrangements.

2.1. Steps of Project Analysis:

Identify project as either Objective-driven or Product-Driven:

- Project whose requirement is to meet certain objectives which could be met in a number of ways, is objective-based project.
- Project whose requirement is to create a product, the details of which have been specified by the client, is product-based project.

Analyse other Project Characteristics:

The following questions can be carefully asked:

- *Is a data oriented or process oriented system to be implemented?*
- *Will the software that is to be produced be a general tool or application specific?*
- *Are there specific tools available for implementing the particular type of application?*
- *Is the system to be created safety critical?*
- *Is the system designed primarily to carry out predefined services or to be engaging and entertaining?*
- *What is the nature of software/hardware environment in which the system will operate?*
- *Does it involve concurrent processing?*
- *Will the system to be produced make heavy use of computer graphics?*

3. Prototyping:

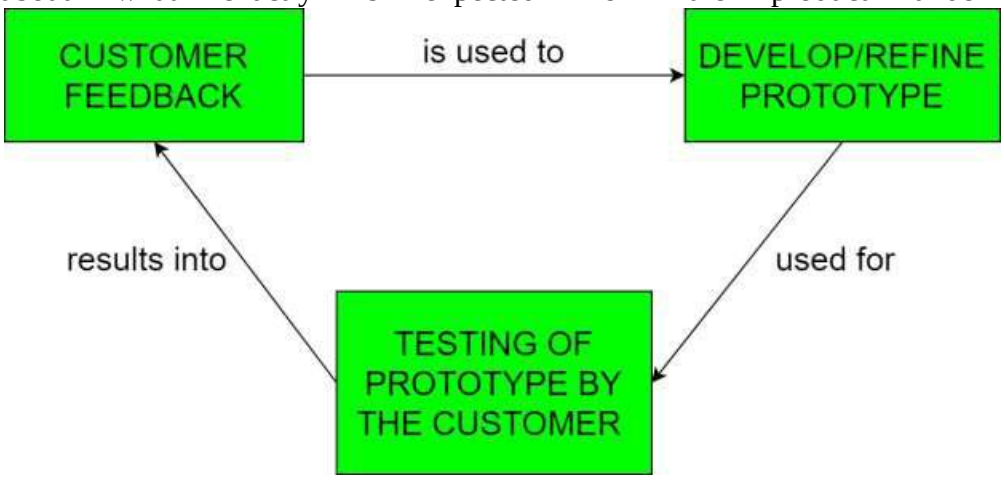
Prototype is a working model of software with some limited functionality.

Prototyping is used to allow the users evaluate developer proposals and try them out before implementation. It also helps understand the requirements which are user specific and may not have been considered by the developer during product design.

What is Software Prototyping?

The Software Prototyping refers to building software application prototypes which displays the functionality of the product under development, but may not actually hold the exact logic of the original software.

Software prototyping is becoming very popular as a software development model, as it enables to understand customer requirements at an early stage of development. It helps get valuable feedback from the customer and helps software designers and developers understand about what exactly is expected from the product under development.



Software Prototyping Types:

There are different types of software prototypes used in the industry. Following are the major software prototyping types used widely −

1) **Throwaway/Rapid Prototyping:**

Throwaway prototyping is also called as rapid or close ended prototyping. Its name refers to the ease and speed with which a prototype can be modified to try different ideas with the user audience and incorporate their feedback. It may go through several cycles of feedback, modification, and evaluation during that time. When all the stakeholders are satisfied, it becomes a reference for the designers and developers to use. After the sprint is completed, the prototype is discarded and a new one is built for the next sprint.

2) **Evolutionary Prototyping:**

Evolutionary prototyping also called as breadboard prototyping . an evolutionary prototype is a functional piece of software, not just a simulation. Evolutionary prototyping starts with a product that meets only the system requirements that are understood. It won't do everything the customer requires, but it makes a good starting point. New features and functions can be added as those requirements become clear to the stakeholders. That's the "evolutionary" nature of this prototype.

3) Incremental Prototyping:

In incremental prototyping, separate small prototypes are built in parallel. The individual prototypes are evaluated and refined separately, and then merged into a comprehensive whole, which can then be evaluated for consistency in look, feel, behavior, and terminology.

4) Extreme Prototyping:

Extreme prototyping is more common for web application development. Web applications are composed of:

- Presentation layer
  - Displayed in the user's browser
- Services layer
  - Communications services
  - Business logic
  - Authentication and authorization
  - Other back-end services

Extreme prototyping is conducted in three phases:

1. **Build** HTML wireframes to simulate the presentation layer. These web pages have limited interactivity.
2. **Transform** the wireframes to fully functional HTML pages, tying them to a simulated services layer.
3. **Code** and implement the services layer.

----------o----------

Part-2

Process Framework: Lifecycle phases, Process Artifacts, Process workflows.

4. Lifecycle 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.

4.1. ENGINEERING AND PRODUCTION STAGES:

To achieve economies 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.

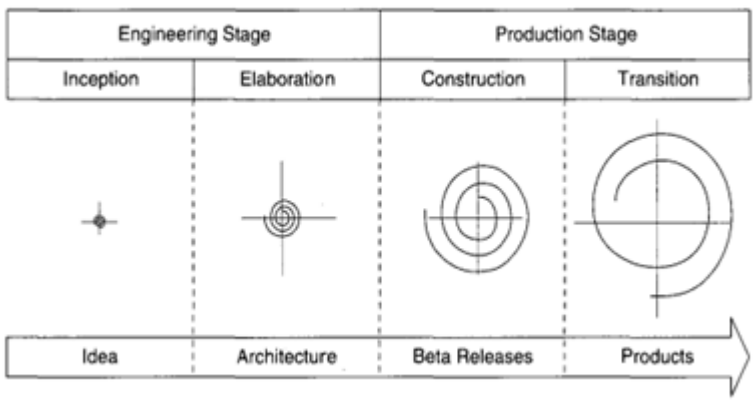
2) **The production stage**: More predictable but larger teams doing construction, test, and deployment activities.

- 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 this 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.

4.2. Inception Phase:

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

Inception Phase involves establishing goals and gathering the requirements needed for the software development. It involves the cost estimation and identifying the risk factors. In the inception phase, we mainly work on the scope of the project and architecture. Feasibility analysis is also an important part of the inception phase.

Primary Objectives: (by end of phase…)

- Establish project software scope and boundary conditions Includes operational concept,

Acceptance criteria, and a clear understanding of what is and what is not intended in the product

- Discriminating the critical use cases (core functionalities) of system and the primary scenarios that will drive the activities
- Demonstrate at least one candidate architecture against some of the primary scenarios (walkthrough it…)
- Estimate the cost and schedule for the entire project (including detailed estimates for the elaboration phase)
- Estimate potential risks (sources of unpredictability)

Essential Activities:

Formulating the scope of the project: This activity involves capturing the requirements and concept of operations in an information repository that describes users'view of the requirements. Repository contains information used to define problem space

Synthesizing the Architecture: Design trade-offs, problem space ambiguities and available solution space assets are evaluated. Repository contains enough information to demonstrate at least a single candidate architecture

Planning and preparing a business case: Alternatives for Risk management strategies, staffing, general iteration plans, cost/schedule/profitability tradeoffs are all evaluated.

Elaboration Phase:

It is easy to argue that Elaboration Phase is clearly the most important phase of all four Phases. At end of this phase the engineering‖ is considered complete. In these phases almost all use cases are designed. During the Elaboration Phase a prototype for gathering requirements and to demonstrate proof of concept is accommodated, and an analysis model is constructed, and a baseline executable architecture is established and demonstrated. In this phase, Risks have been addressed and strategies understood.

Primary Objectives (at end of phase…)

- Base-lining the architecture as rapidly as possible
- Base-lining the Vision.
- Base-lining a detailed plan for Construction
- Demonstrating the baseline architecture such that it clearly supports the vision

Essential Activities:

Elaborating the Vision: This activity involves establishing a high understanding of the critical usecaes.

Elaborating the process and infrastructure support: 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.

4.3. Construction Phase:

Its main focus on design and implementation. In the early stages the main focus is on the depth of the design artefacts. 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.

Primary Objectives:

- Develop the system rapidly but with high quality – that is, construction (programming and unit testing) implements the design; realize the design.
- Take advantage of the process, versioning, reviews, assessment, etc. to minimize costs due to needless rework and scrap.
- Develop alpha, beta, or what have you releases for Transition phase.

Essential Activities:

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

Primary Evaluation Criteria (at end)

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

4.4. Transition Phase:

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.

PreviousNext:

This phase could include any of the following activities:

- Beta testing to validate new system against expectations
- Beta testing in parallel with legacy system to be replaced
- Conversion of operational data bases
- Training of users and mainainteners.

Primary Objectives:

- Achieving users self-supportability.
- Achieving final product baselines as rapidly and cost effectively.

Essential Activities:

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

5. Process Artifacts:

An **artifact** is one of many kinds of tangible by-product produced during the development of software. Some artifacts (e.g., use cases, class diagrams, and other UML models, requirements and design documents) help describe the function, architecture, and design of software. Other artifacts are concerned with the process of development itself—such as project plans, business cases, and risk assessments.

Al development of software artifacts: build the requirement, construct a design model, compile and test the implementation for deployment. This process works for small scale and purely custom developments.

Conventional software projects focused on the sequential development of software artifacts:
build the requirement, construct a design model, compile and test the implementation for Deployment. This process works for small scale and purely custom developments.

5.1. The Artifacts set:

Introduction: - 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. Whereas the remaining four artifacts are captured rigorous notations that support automated analysis and browsing.

1) **<u>The Management Set</u>**:-

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

Project personnel: project manager, architects, developers, testers, marketers, administrators

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

- 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.

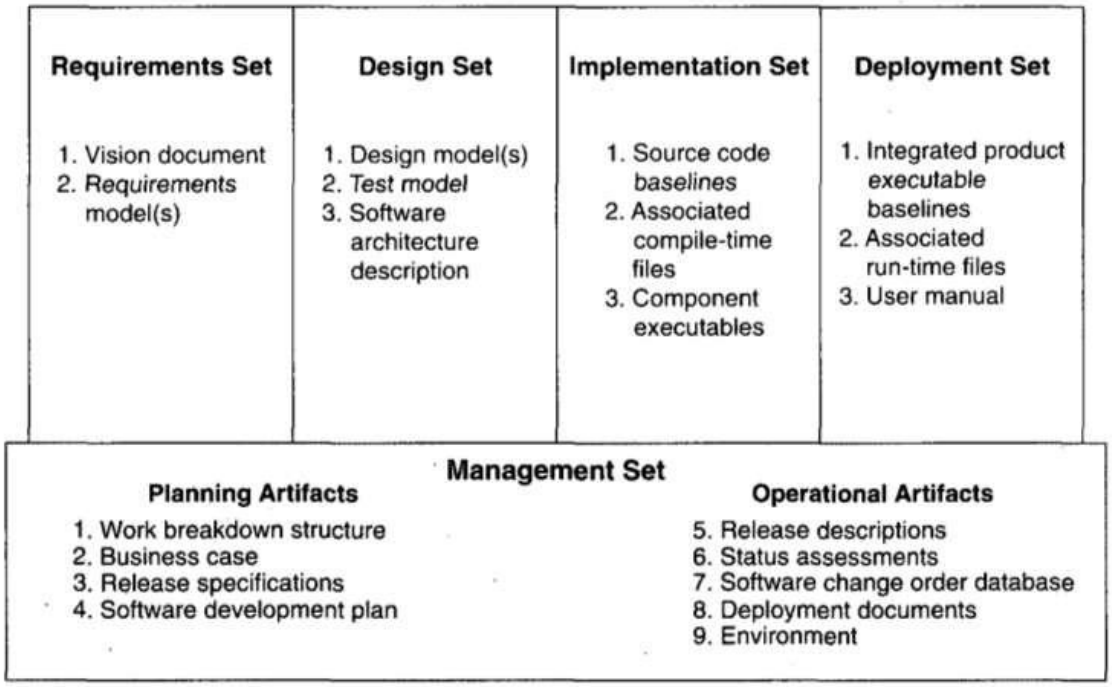


FIGURE 6-1. *Overview of the artifact sets*

2) **<u>Requirement Set:</u>**

- The Requirement set, Design set, Implementation set, Deployment set are combine called as Engineering Set.
- The requirements set are 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 the following

- Analysis of the consistency with the release specification of the management set.
- Analysis of the consistency between the vision and the requirement models.
- Mapping against the ***Design set, Implementation set, Deployment set***
- Analysis of the changes between current version and previous versions of the artifacts.

3) **<u>Design Set:</u>**

UML notations are used to engineer the design models for the solution.

- Design model includes structural and behavioral information to represent the problem domain.
- Design Set artifacts are evaluated, assessed and measured through a combination of the following:

- Analysis of the internal consistency and quality of the design model.
- Analysis of the consistency with the requirement models.
- Translation into implementation and deployment sets and notations.
- Subjective review of other dimensions of quality

4) **<u>Implementation Set:</u>**

- Implementation Set artifacts includes source code (as implementation of components) their form, interfaces, and executables necessary for stand-alone testing of components.
- Tools used in Implementation are debuggers, compilers, code analyzers, test coverage analysis tools, test management tools.
- These executables are the primitive parts needed to construct the end products including custom components, APIs, other reusable or legacy components in some programming languages.
- Implementation Set artifacts are evaluated, assessed and measured through a combination of the following:
    - Analysis of consistency with the design models
    - Assessment of component source or executable files.
    - Execution of stand-alone component test cases that automatically compare expected results with actual result.
    - Analysis of the changes between current version and previous versions of the artifacts.
    - Subjective review of other dimensions of quality

5) **Deployment Set:**

- Deployment set artifacts normally include the deliverables and machine language notations, executable software, build scripts, installation scripts, and executable target specific data necessary to use the product in its target environment.
- Tools used in setting things up for / getting ready for deployment are test coverage and test automation tools; network management tools, commercial components (OS, GUI, DBMSs, middleware, installation tools, etc.)
- Deployment Set artifacts are evaluated, assessed and measured through a combination of the following:
- Tested against defined use scenarios in the user manual such as installation, user-oriented dynamic reconfiguration, mainstream usage, and anomaly management.
- Testing against the usage scenarios in the user manual such as installation, mainstream usage, and anomaly management.

5.2. Test Artifacts

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

- Whatever the document-driven approach that was applied to software development is also followed by the software testing people.
- Test teams built system test plan documents, system test procedures document, integration test plan documents, unit test plan document, stubs or drivers.
- This document driven approach caused the same problems for the test activities that it did for the development activities.
- The test artifacts must be developed concurrently with the product from inception through deployment. Thus Testing is a full life-cycle activity.
- These test artifacts are communicated, engineered and developed within the same artifact sets as the developed product.
- The test artifacts are documented in the same way that the product is documented.
- Developers of the test artifacts use the same tools and techniques as the software engineers.

Management Artifacts:

- Work Breakdown Structure
- Business Case
- Release Specifications
- Software Development Plan
- Release Descriptions
- Status Assessments
- Software change Order Database
- Deployment

- Environment
- Management Artifact Sequences

1. **Work Breakdown Structure (WBS):**
   - WBS is a VERY commonly used _document'which is absolutely essential for Management!!
   - A WBS is the vehicle for budgeting and collecting costs.
   - It is essential for tracking expenses throughout development – all phases and activities.
   - Basically it focuses on budgeting, monitoring, and controlling project costs and deals with specifications like:
     - How resources are expended for activities undertaken
     - Trends and projections….

2. **Business Case:-**
   The business case artefact provides all the information necessary to determine whether the project is worth investing in.
   - It includes expected costs, expected revenues, technical and management plans, consideration of risk, expected ROI (Return on Investment), etc.
   - It also deals with the effects of NOT investing in project, etc.
   - Normally a Business Case is accommodated in text and graphics.
   - The main purpose is to transform Vision document into economic terms so that the organization can make accurate predictions on the ROI.

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*

3. **Release Specifications:**
   - Release Specifications deals with documentation accompanying every release.
   - The features of Release Specifications are derived from Vision statement, development, and testing.
   - scope, plan, and objective evaluation criteria for each baseline release are derived from the vision statement as well as many other sources.

   Two kinds of requirements info addressed in Release specifications:
   - Vision statement
   - Evaluation Criteria.

Vision Statement:

- Vision Statement is evolutionary and in Vision Statement High level requirements are modeled
- Vision statement serves as Contract between developers and customer
- Vision usually contains the Use Case Model and Use Case Descriptions. (SRS)

Evaluation criteria:

- Evaluation criteria contains details (often lower level) on how to evaluate the project.
- It deals with questions like ―How do we know / demonstrate that we achieved the objectives of the iteration?

4. **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.

Discusses required artifacts, who will do what, quality checkpoints, the development environment, configuration control board planning and procedures; change management, base lining, assessment, risk and status assessment, standards to be adhered to, etc.

- Covers whole spectrum of development activity.

Status Assessment:

- 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 snap-shots), major milestone plans and results, total project or product scope, action items, and follow-through.

5. **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.

6. **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.

7. **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.

8. **Engineering Artifacts:**
   - Vision Document
   - Architecture Description
   - Software User Manual

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, plaits, and technology.
- A good vision document should change slowly.
- The vision document is written from the user's perspective, focusing on the essential features of the system and acceptable levels of quality.
- The vision document should contain at least two appendixes. The first appendix should describe the operational concept using use cases (a visual model and a separate artifact). The second appendix should describe the change risks inherent in the vision statement, to guide defensive design efforts.
- The vision statement should include a description of what will be included as well as those features considered but not included.
- The vision document provides the contractual basis for the requirements visible to the stakeholders.

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.

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**

FIGURE 6-10. *Typical architecture description outline*

5.6 Pragmatic Artifacts:

- **People want to review information but don't understand the language of the artifact**.

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

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

- **Human-readable engineering artifacts should use rigorous notations that are complete, consistent, and used in a self-documenting manner.**

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

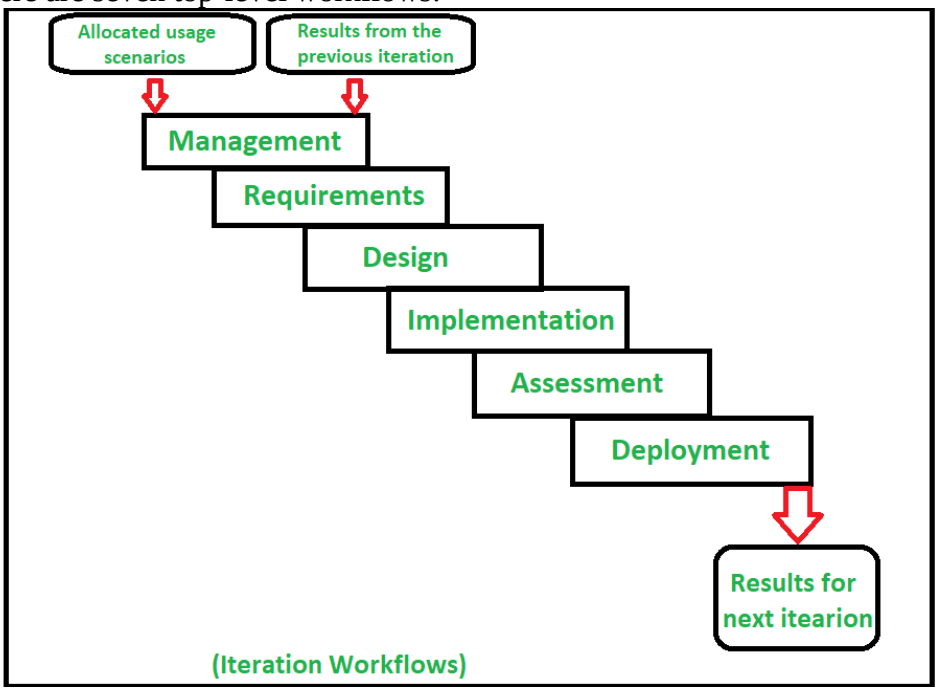
- **Useful documentation is self-defining: It is documentation that gets used.**

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

On-line and Web-based artifacts can be changed easily and are viewed with more skepticism because of their inherent volatility.

6. Software Process Workflows:

The term *workflow* is used to mean a thread of cohesive and mostly sequential activities. There are seven top-level workflows:



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

6.1 Approaches for Workflows:

- ***Architecture-first approach:*** Extensive requirements analysis, design, implementation, and assessment activities are performed before the construction phase, when full-scale implementation is the focus.

- ***Iterative life-cycle process:*** Some projects may require only one iteration in a phase; others may require several iterations. The point is that the activities and artifacts of any given workflow may require more than one pass to achieve adequate results.
- ***Round-trip engineering:*** Raising the environment activities to a first-class workflow is critical. The environment is the tangible embodiment of the project's process, methods, and notations for producing the artifacts.

- ***Demonstration-based approach:*** Implementation and assessment activities are initiated early in the life cycle, reflecting the emphasis on constructing executable subsets of the evolving architecture.

6.2 **ITERATION WORKFLOWS:**

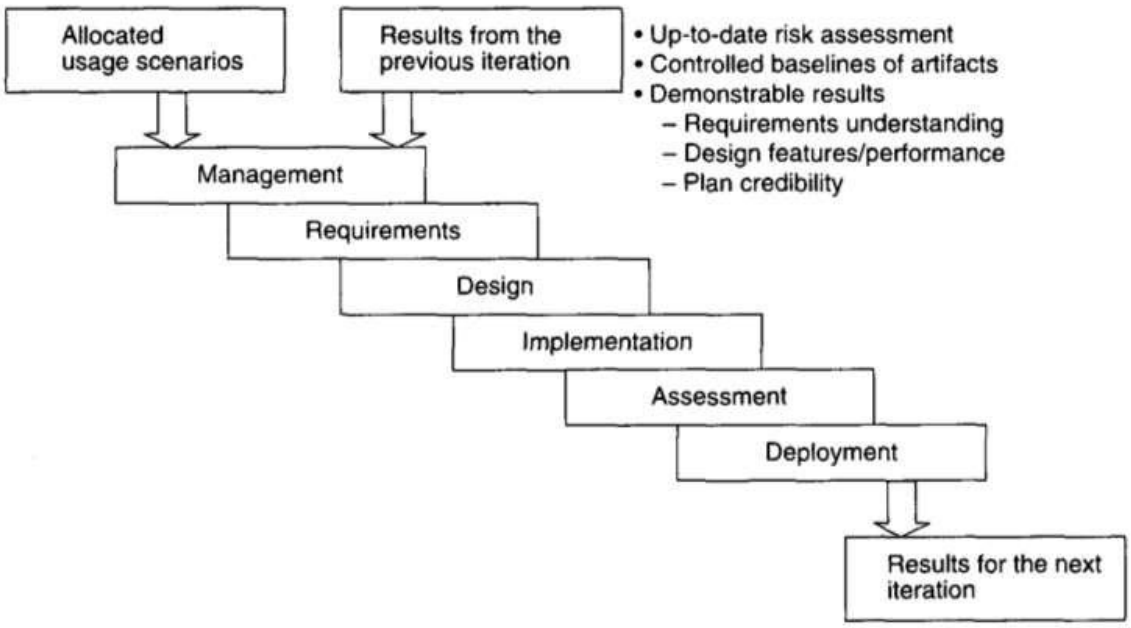


FIGURE 8-2. *The workflow of an iteration*

1. **Management:**
   Iteration planning to determine the content of the *release* and develop the detailed plan for the iteration; assignment of work pack-ages, or tasks, to the development team

2. **Environment:**
   Evolving the software change order database to reflect all new baselines and changes to existing baselines for all product, test, and environment components

3. **Requirements:**
   Analyzing the baseline plan, the baseline architecture, and the baseline requirements set artifacts to fully elaborate the use cases to be demonstrated at the end of this iteration and their evaluation criteria; updating any requirements set artifacts to reflect changes necessitated by results of this iteration's engineering activities

4. **Design:**
   Evolving the baseline architecture and the baseline design set arti-facts to elaborate fully the design model and test model components necessary to demonstrate against the evaluation criteria allocated to this iteration; updating design set artifacts to reflect changes necessitated by the results of this iteration's engineering activities

5. **Implementation:**

   Developing or acquiring any new components, and enhancing or modifying any existing components, to demonstrate the eval-uation criteria allocated to this iteration; integrating and testing all new and modified components with existing baselines

6. **Assessment:**

   Evaluating the results of the iteration, including compliance with the allocated evaluation criteria and the quality of the current base-lines; identifying any rework required and determining whether it should be performed before deployment of this release or allocated to the next release; assessing results to improve the basis of the subsequent iteration's plan

7. **Deployment:**

   transitioning the release either to an external organization(such as a user, independent verification and validation contractor, or regulatory agency) or to internal closure by conducting a post-mortem so that lessons learned can be captured and reflected in the next iteration