

UNIT 1

SHORT ANSWER QUESTIONS

1) What are the results of analyses that are performed on the software engineering industry in mid 1990's?

Ans: All the three analyses reached to the same general conclusion: the success rate of the software projects is low.

The three analyses results can be summarized as follows:

1. Only about 10% of software projects are delivered successfully on time, within initial budget, and meets user requirements
2. The management discipline is more of a discriminator in success or failure than are technology advances
3. The level of software scrap and rework is indicative of an immature process.

2) What is the paper presented by Winston Royce in 1970's and what are the primary key points of the paper?

Ans: Winston Royce in 1970 presented a paper titled "managing the development of large scale software systems"

This paper made three primary points

1. There are two essential steps common to the development of computer programs: **analysis and coding**
2. In order to manage and control all of the intellectual freedom associated with software development, introduces several other steps, including system requirements definition, software requirements definition, program design, and testing. This is called the large scale system approach.
3. The basic framework described in the waterfall model is risky and invites failure.

3) List out the necessary improvements for the waterfall model?

- Ans:
1. Program design comes first
 2. Document the design:
 3. Do it Twice:
 4. Plan, control and monitor testing:
 5. Involve the customer:

4) What are the symptoms of the projects that exhibits troubles frequently?

- Ans:
1. Protracted Integration and Late Design Breakage
 2. Late Risk resolution
 3. Requirements-Driven Functional Decomposition
 4. Adversarial Stakeholder Relationships

5. Focus on Documents and Review Meetings

5) What are the parameters for cost estimation?

Ans: The cost of the software can be estimated by considering the following things as parameters to a function.

- 1) Size: Which is measured in terms of the number of Source Lines of Code or the number of function points required to develop the required functionality?
- 2) Process: Used to produce the end product, in particular the ability of the process is to avoid nonvalue adding activities (rework, delays, and communications overhead).
- 3) Personnel: The capabilities of software engineering personnel, and particularly their experience with the computer science issues and the application domain issues of the project.
- 4) Environment: Which is made up of the tools and techniques available to support efficient software development and to automate the process?
- 5) Quality: It includes its features, performance, reliability, and flexibility. The relationship among these parameters and estimated cost can be calculated by using,
$$\text{Effort} = (\text{Personnel}) (\text{Environment}) (\text{Quality}) (\text{Size}^{\text{Process}})$$

6) what are the important topics b/w vendors and developers about cost estimation models?

Ans: In order to estimate the cost of a project the following three topics should be considered,

- Which cost estimation model to use?
- Whether to measure software size in SLOC or function point.
- What constitutes a good estimate?

7) What constitutes a good s/w cost estimate?

Ans: The good estimate has the following attributes:

It is conceived and supported by the project manager, architecture team, development team and test team accountable for performing the work.

It is accepted by all stakeholders as ambitious but realizable.

It is based on a well defined s/w cost model with a credible basis.

It is based on a database of relevant project experience that includes similar processes, similar technologies, similar environment, similar quality requirements and similar people.

It is defined in enough detail so that its key risk areas are understood and the probability of success is objectively assessed.

8) What are the risks in waterfall model?

- Ans:**
1. Protracted Integration and Late Design Breakage
 2. Late Risk resolution
 3. Requirements-Driven Functional Decomposition
 4. Adversarial Stakeholder Relationships

5. Focus on Documents and Review Meetings

LONG ANSWER QUESTIONS

1) **Explain the improvements to the basic waterfall model that would eliminate most of the development risks?**

Ans: The basic improvements of waterfall model are

1. Program design comes first:

- Preliminary design phase b/w the software requirements generation phase and the analysis phase.
- Because of the early design the program can sense the consequences of all the constraints like storage, timing and data flux.
- If there is any resources are insufficient or operational design is wrong, it can be recognized at this early stage.

Steps to implement this early program design:

- Start the design process with program designers.
- Design, define, and allocate the data processing modes even at the risk of being wrong.
- Write an overview document.

2. Document the design:

The amount of document required on most software programs is more.

Why do we need so much documentation?

1. For communication b/w designers, managers and customers.
2. In early phases documentation is the design.
3. Later it is used to test, maintenance teams.

3. Do it Twice:

- The version delivered to the customers is actually the second version.
- In the first version, the team members can quickly sense trouble in design if any, model them, model alternatives, finally arrive at an error-free-program.

4. Plan, control and monitor testing:

- The main resources of project are manpower, computer time, and management judgment in test phase.
- This is the phase of greatest risk in terms of cost and schedule.

In testing phase, the important things to be done are:

1. Employ a of test specialists who were not responsible for original design.
2. Employ visual inspections to spot obvious errors(wrong address, miss placing of operators)

3. Test every logic path
4. Employ final checkout on target computer

5. Involve the customer:

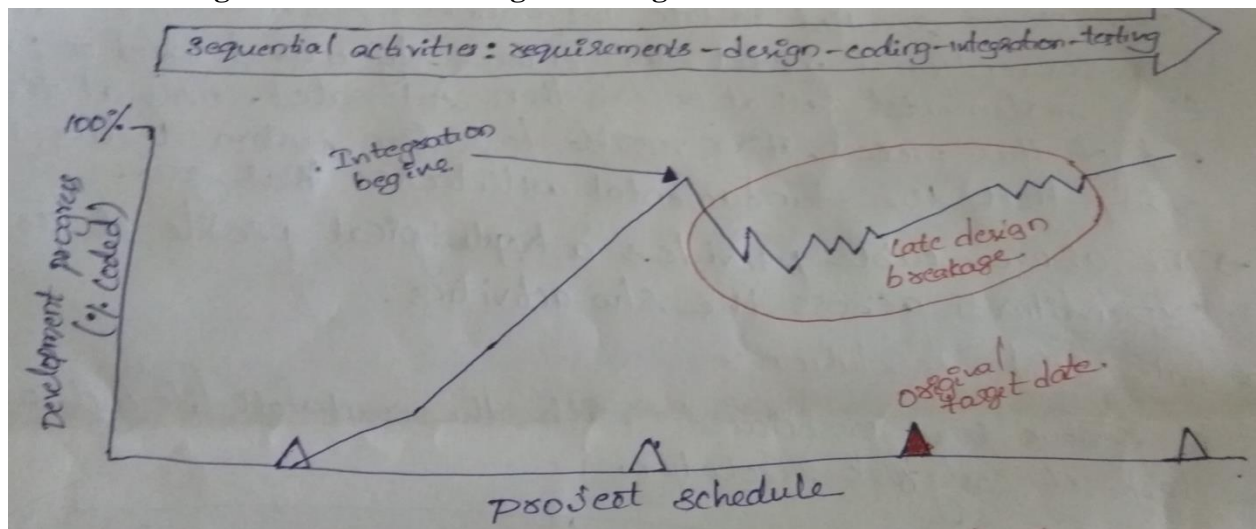
- It is important to involve the customer in a formal way so that he has committed himself at earlier points before final delivery by conducting some reviews such as,

- Preliminary software review during preliminary program design step.
- Critical software review during program design.
- Final software acceptance review following testing.

2) Explain the symptoms of the projects that exhibit troubles frequently?

Ans:

1. Protracted Integration and Late Design Breakage:



In the conventional model, the entire system was designed on paper, then implemented all at once, then integrated. Only at the end of this process was it possible to perform system testing to verify that the fundamental architecture was sound.

- Here the testing activities consume 40% or more life-cycle resources.

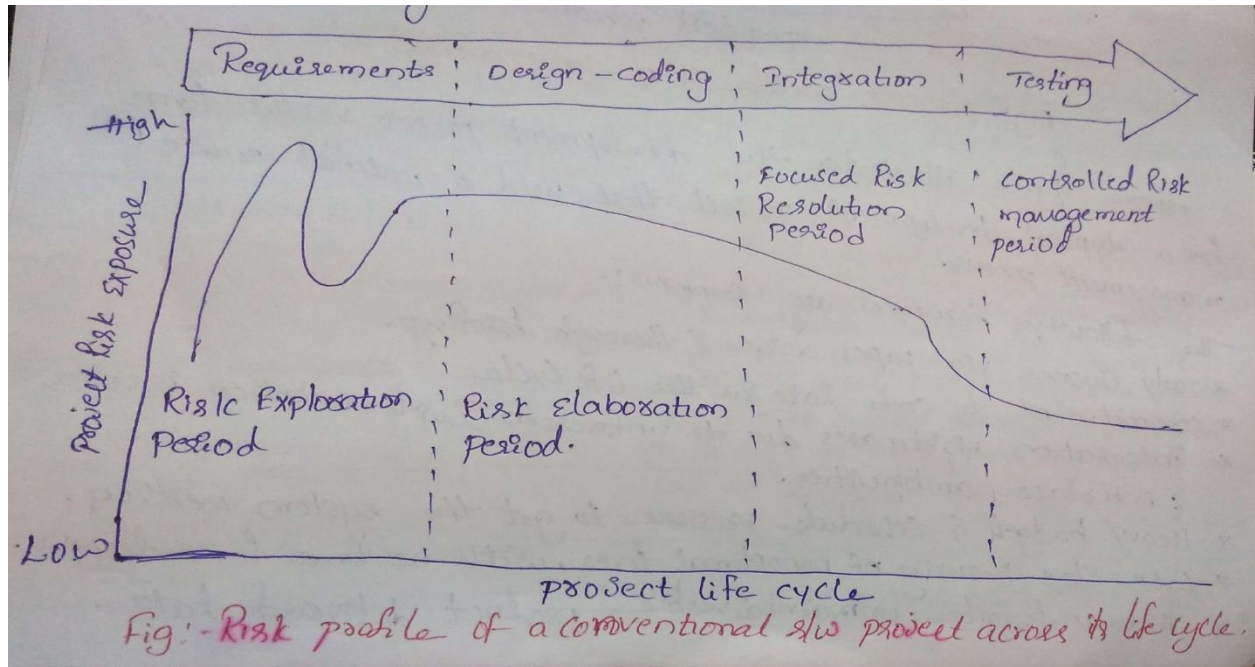
Expenditures by activity for a conventional s/w project.

Activity	Cost
Management	5%
Requirements	5%
Design	10%
Code and unit test	30%
Integration and Test	40%
Deployment	5%

Environment	<u>5%</u>
Total	100%

2. Late Risk resolution:

A serious issues associated with the waterfall life cycle was the lack of early risk resolution. The risk profile of a waterfall model is



- It includes four distinct periods of risk exposure, where risk is defined as “the probability of missing a cost, schedule, feature, or quality goal”.
- In requirements phase risk is unpredictable.
- In design phase, risk exposure stabilized.
- In coding some of the risks got resolved.
- In integration many real design issues were resolved and engineering tradeoffs were made.
- Resolving the risks in late in the lifecycle was very expensive.

3. Requirements-Driven Functional Decomposition

- s/w development process is requirement driven
- Before begin the development activities all the requirements must be complete and clear.
- It treats all the requirements equally begin the development.
- This equal treatment of all requirements would waste the time of the project and effort.

4. Adversarial Stakeholder Relationships

- The conventional process tended to result in adversarial stakeholder relationships because of the difficulties of requirements specification.
- The following sequence of events was typical for most contractual software efforts:

- -The contractor prepared deliverable document that captured an intermediate artifact and delivered it to the customer for approval.
- -The customer was expected to provide comments (within 15 to 30 days)
- -The contractor integrated these comments and submitted a final version for approval (within 15 to 30 days)

5. Focus on Documents and Review Meetings

- The conventional process focused on various documents that attempted to describe the software product.
- Contractors produce literally tons of paper to meet milestones and demonstrate progress to stakeholders, rather than spend their energy on tasks that would reduce risk and produce quality software.
- Presenters and audience reviewed the simple thing that they understood rather than the complex and important issues.
- Most design reviews resulted in low engineering and high cost in terms of the effort and schedule involved in their preparation and conduct.

3) What are the Barry Boehm's Top 10 "Industrial Software Metrics": for conventional software management performance?

Ans:

- 1) Finding and fixing a software problem after delivery costs 100 times more than finding and fixing the problem in early design phases.
- 2) You can compress software development schedules 25% of nominal (small), but no more.
- 3) For every \$1 you spend on development, you will spend \$2 on maintenance.
- 4) Software development and maintenance costs are primarily a function of the number of source lines of code.
- 5) Variations among people account for the biggest difference in software productivity.
- 6) The overall ratio of software to hardware costs is still growing. In 1955 it was 15:85; in 1985, 85:15.
- 7) Only about 15% of software development effort is devoted to programming.
- 8) Software systems and products typically cost 3 times as much per SLOC as individual software programs. Software-system products cost 9 times as much.
- 9) Walkthroughs catch 60% of the errors.
- 10) 80% of the contribution comes from 20% of the contributors.
 - 80% of the engineering is consumed by 20% of the requirements.
 - 80% of the software cost is consumed by 20% of the components.
 - 80% of the errors are caused by 20% of the components.
 - 80% of the software scrap and rework is caused by 20% of the errors.
 - 80% of the resources are consumed by 20% of the components.
 - 80% of the engineering is accomplished by 20% of the tools.

- 80% of the progress is made by 20% of the people.

4) Explain the pragmatic software cost estimation?

Ans : If there is no proper well-documented case study then it is difficult to estimate the cost of the software. It is one of the critical problems in software cost estimation.

- But the cost model vendors claim that their tools are well suitable for estimating iterative Development projects.

In order to estimate the cost of a project the following three topics should be considered,

- Which cost estimation model to use?

- Whether to measure software size in SLOC or function point.

- What constitutes a good estimate?

- There is a lot of software cost estimation models are available such as, COCOMO, CHECKPOINT, ESTIMACS, Knowledge Plan, Price-S, ProQMS, SEER, SLIM, SOFTCOST, and SPQR/20.

Of which COCOMO is one of the most open and well-documented cost estimation models.

- The software size can be measured by using 1) SLOC 2) Function points.
- Most software experts argued that the SLOC is a poor measure of size. But it has some value in the software Industry.
- SLOC worked well in applications that were custom built why because of easy to automate and Instrument.
- Now a days there are so many automatic source code generators are available and there are so many advanced higher-level languages are available. So SLOC is a uncertain measure.
- The main advantage of function points is that this method is independent of the technology and is Therefore a much better primitive unit for comparisons among projects and organizations.
- The main disadvantage of function points is that the primitive definitions are abstract and measurements are not easily derived directly from the evolving artifacts.
- Function points is more accurate estimator in the early phases of a project life cycle. In later phases, SLOC becomes a more useful and precise measurement basis of various metrics perspectives.
- The most real-world use of cost models is bottom-up rather than top-down.
 - The software project manager defines the target cost of the software, and then manipulates the parameters and sizing until the target cost can be justified.

5). Explain software economics?

Ans: The cost of the software can be estimated by considering the following things as parameters to a function.

1) Size: Which is measured in terms of the number of Source Lines of Code or the number of function points required to develop the required functionality?

- 2) Process: Used to produce the end product, in particular the ability of the process is to avoid nonvalue adding activities (rework, delays, and communications overhead).
- 3) Personnel: The capabilities of software engineering personnel, and particularly their experience with the computer science issues and the application domain issues of the project.
- 4) Environment: Which is made up of the tools and techniques available to support efficient software development and to automate the process?
- 5) Quality: It includes its features, performance, reliability, and flexibility. The relationship among these parameters and estimated cost can be calculated by using,

$$\text{Effort} = (\text{Personnel}) (\text{Environment}) (\text{Quality}) (\text{Size}^{\text{Process}})$$

1) **Conventional:** 1960 and 1970, Craftsmanship. Organizations used custom tools, custom Processes, and virtually all custom components built in primitive languages. Project performance was highly predictable.

2) **Transition:** 1980 and 1990, software engineering. Organizations used more-repeatable processes and off-the-shelf tools, and mostly (>70%) custom components built in higher level languages. Some of the components (<30%) were available as commercial products like, OS, DBMS, Networking and GUI.

3) **Modern practices:** 2000 and later, software production.

- 70% component-based,

- 30% custom

Conventional Transition Modern Practices

- 1960s – 1970s - 1980s –1990s - 2000 and on

- Waterfall model - Process improvement - Iterative development

- Functional design - Encapsulation - based - Component-based

- Diseconomy of scale- Diseconomy of scale- ROI

Environments /tools:

Custom Off-the-shelf, separate Off-the-shelf, Integrated

Size:

- 100% custom 30% component-based 70% component-based

- 70% custom 30% custom

Process:

- Ad hoc Repeatable Managed/measured

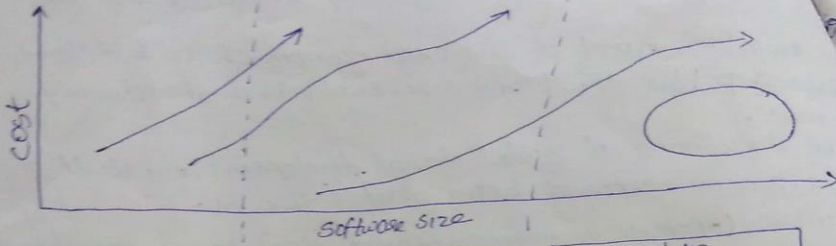
Typical Project Performance:

- Always: Infrequently: Usually:

- Over budget On budget On budget

- Over schedule On schedule On schedule

Target objective: improved ROI



- 1960s - 1970s
 - waterfall model
 - functional design
 - Dir. economy of scale

- 1980s - 1990s
 - process improvement
 - Encapsulation-based
 - Dir. economy of scale

- 2000 and on
 - iterative development
 - component-based
 - Return on investment

corresponding environment, size, and process technologies

Conventional
 Environments/Tools:
 custom
 SRE:
 100% custom
 Process:
 Ad hoc

Transition
 Environments/Tools:
 off-the-shelf, separated
 size:
 50% component-based,
 50% custom
 Process:
 Repeatable

modern practices
 Environments/Tools:
 of-the-shelf, integrated
 size:
 70% component-based,
 30% custom
 Process:
 managed/measured

Typical project performance

predictably bad
 always:
 over budget
 over schedule

unpredictable
 infrequently:
 on budget
 on schedule

predictable
 usually:
 on budget
 on schedule

Fig: Three generations of s/w economics, leading to the target objective.

UNIT II

SHORT ANSWER QUESTIONS

1) What are the parameters of software cost model and what are the dimensions around these parameters?

Ans: The basic parameters of the software cost model are size, process, personnel, environment, and quality.

Important dimensions around these parameters are:

1. Reducing the size or complexity of what needs to be developed
2. Improving the development process
3. Using more-skilled personnel and better teams (not necessarily the same thing)
4. Using better environments (tools to automate the process)
5. Trading off or backing off on quality thresholds

2) what are the booch reasons for object oriented projects success?

Ans:

1. An OO-model of the problem and its solution encourages a common vocabulary between the end user of a system and its developers, thus creating a shared understanding of the problem being solved.
2. The use of continuous integration creates opportunities to recognize risk early and make incremental corrections without weaken the entire development effort.
3. An OO-architecture provides a clear separation among different elements of a system, creating firewalls that prevent a change in one part of the system from the entire architecture.

3) what are the characteristics of a successful object oriented project?

Ans:

- 1) A ruthless focus on the development of a system that provides a well understood collection of essential minimal characteristics.
- 2) The existence of a culture that is centered on results, encourages communication.
- 3) The effective use of Object Oriented-modeling.
- 4) The existence of a strong architectural vision. (based on stakeholders concerns, business requirements, scope, constraints and principles.)
- 5) The application of a well-managed iterative and incremental development life cycle.

4) What are the characteristics of Organizations that translates reusable components into commercial products?

Ans:

1. They have an economic motivation for continued support.

2. They take ownership of improving product quality, adding new features and transitioning to new technologies.
3. They have a sufficiently broad customer base to be profitable.

5) **What are the Boehm offered staffing principles:**

Ans:

- 1) **The principle of top talent:** Use better and fewer people.
- 2) **The principle of job matching:** Fit the tasks to the skills and motivation of the people available.
- 3) **The principle of career progression:** An organization does best in the long run by helping its people to self-actualize.
- 4) **The principle of team balance:** Select people who will complement and synchronize with one another.
- 5) **The principle of phase-out:** Keeping a misfit on the team doesn't benefit anyone.

6) **What are the common methods to achieve staffing?**

Ans:

1. If people are already available with required skill set, just take them
2. If people are already available but do not have the required skills, re-train them
3. If people are not available, recruit trained people
4. If you are not able to recruit skilled people, recruit and train people.

7) **What are the attributes of successful s/w project manager?**

Ans:

1. **Hiring skills:** Few decisions are as important as hiring decisions. Placing the right person in the right job seems obvious but is surprisingly hard to achieve.
2. **Customer-interface skill:** Avoiding adversarial relationships among stake-holders is a prerequisite for success.
3. **Decision-making skill:** a software project manager was able to take right decision in right time.
4. **Team-building skill:** Teamwork requires that a manager establish trust, motivate progress, transition average people into top performers, eliminate misfits.
5. **Selling skill:** Successful project managers must sell all stakeholders (including themselves) on decisions and priorities, sell candidates on job positions, and sell achievements against objectives.

LONG ANSWER QUESTIONS

1) Explain reducing the software product size?

Ans:

Reducing software product size:

- The most significant way to improve affordability and ROI is usually to produce a product that achieves the design goals with minimum amount of human generated source material.
- Component based development is introduced for reducing the source language size necessary to achieve a s/w solution.
- This size reduction is the primary motivation behind
 - ✓ Improvements in higher order languages.(c++, Ada 95, java, etc.).
 - ✓ Automatic code generators.
 - ✓ Reuse of commercial components.(os, windows environment, DBMS, n/w's, etc.)
 - ✓ Object oriented technologies.(UML, visual modeling tools, architecture frameworks.)

❖ **Languages:**

- UFP's (Universal Function Points) are useful estimators for language-independent in the early life cycle phases.
- The basic units of function points are:
 - i. External user inputs (outside of the s/m to inside)
 - ii. External outputs (process of data passes from inside to outside.)
 - iii. Internal logical data groups.(user identified group of logically related data that resides within the application and maintained through external i/p's.)
 - iv. External data Interfaces.(user identified group of logically related data that is used for reference purposes only, data resides outside and maintain by another application external i/p's.)
 - v. External inquiries.(process with both i/p and o/p components that results in data retrieval from one or more internal logical files and external interface files.).
- SLOC metrics are useful estimators for software after a candidate solution is formulated and an implementation language is known.
- The substantial data have been documented relating sloc to function points.

LANGUAGE	SLOC PER UFP
Assembly	320
C	128
FORTRAN 77	105
COBOL 85	91
Ada 83	71
C++	56
Ada 95	55
Java	55
Visual Basic	35

- Universal function points can be used to indicate the relative program sizes required to implement a given functionality.

EX: To achieve a given application with a fixed number of function points, one of the following program sizes would be required

10,00,000 lines of assembly languages.

4,00,000 lines of C.

2,20,000 lines of Ada 83.

1,75,000 lines of Ada 95 or C++.

- The values indicate the relative expressiveness provided by various languages.
- Further commercial components and automatic code generators can further reduce the size of human-generated source code, which in turn reduces the size of the team and the time needed for development.
- To the example, by adding a commercial DBMS, commercial GUI builder, and commercial middleware could reduce the effective size of development to the following final size:
75,000 lines of Ada 95 or C++ plus integration of several commercial components.
- Reducing size usually increases understandability, changeability & reliability.

❖ **Object-oriented Methods and Visual Modeling:**

Some studies concluded that Object-Oriented programming languages appear to benefit both software productivity and software quality. One of such Object-Oriented method is UML- Unified Modeling Language.

Booch has described three other reasons that certain object oriented projects success.

1. An OO-model of the problem and its solution encourages a common vocabulary between the end user of a system and its developers, thus creating a shared understanding of the problem being solved.
2. The use of continuous integration creates opportunities to recognize risk early and make incremental corrections without weaken the entire development effort.
3. An OO-architecture provides a clear separation among different elements of a system, creating firewalls that prevent a change in one part of the system from the entire architecture.

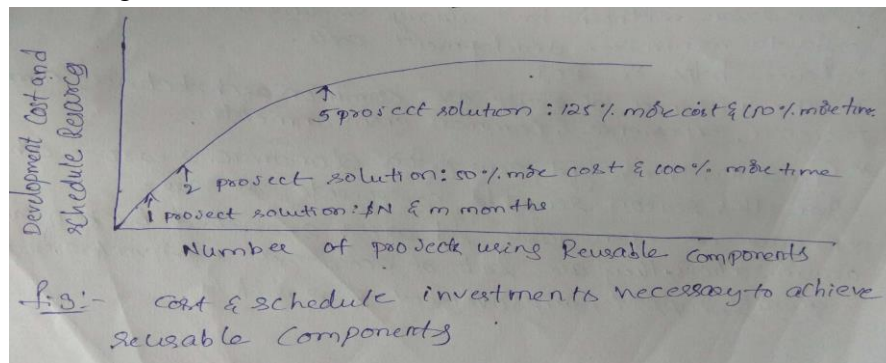
Booch also summarized five characteristics of a successful object oriented project:

- 1) A ruthless focus on the development of a system that provides a well understood collection of essential minimal characteristics.
- 2) The existence of a culture that is centered on results, encourages communication.
- 3) The effective use of Object Oriented-modeling.
- 4) The existence of a strong architectural vision. (based on stakeholders concerns, business requirements, scope, constraints and principles.)
- 5) The application of a well-managed iterative and incremental development life cycle.

❖ **Reuse:**

- Reusing existing components and building reusable components have been natural software engineering activities.
- Software design methods have always implicit with reuse in order to minimize development costs.
- Reuse helps on ROI
- The instances of reuse are common architecture, common processes, experience & common environments.
- **Organizations that translates reusable components into commercial products has the following characteristics:**
 1. They have an economic motivation for continued support.
 2. They take ownership of improving product quality, adding new features and transitioning to new technologies.
 3. They have a sufficiently broad customer base to be profitable.
- TO succeed in the marketplace for commercial components an organization needs following elements:-

- ❖ a development group
- ❖ support infrastructure
- ❖ product-oriented sales
- ❖ marketing infrastructure



❖ **COMMERCIAL COMPONENTS:**

- The use of commercial components is certainly desirable as a means of reducing custom development, it has not proven.

APPROACH	ADVANTAGES	DISADVANTAGES
Commercial components	<ul style="list-style-type: none"> Predictable license costs Broadly used, mature technology Available now Dedicated support organization Hardware/software independence Rich in functionality 	<ul style="list-style-type: none"> Frequent upgrades Up-front license fees Recurring maintenance fees Dependency on vendor Run-time efficiency sacrifices Functionality constraints Integration not always trivial No control over upgrades and maintenance Unnecessary features that consume extra resources Often inadequate reliability and stability Multiple-vendor incompatibilities
Custom development	<ul style="list-style-type: none"> Complete change freedom Smaller, often simpler implementations Often better performance Control of development and enhancement 	<ul style="list-style-type: none"> Expensive, unpredictable development Unpredictable availability date Undefined maintenance model Often immature and fragile Single-platform dependency Drain on expert resources

2) Explain how to improve software processes?

Ans: In Software oriented organizations, there are many processes and sub processes.

- There are three distinct process perspectives
 1. **Metaprocess:** It is an Organization's policies, procedures, and practices for pursuing a software- intensive line of business. The focus of this process is of organizational economics, long-term strategies, and a software ROI.
 2. **Macroprocess:** A project's policies, procedures and practices for producing a complete software product within certain cost, schedule, and quality constraints. The focus of the macroprocess is on creating an sufficient instance of the metaprocess for a specific set of constraints.
 3. **Microprocess:** A projects team's policies, procedures, and practices for achieving an artifact of a software process. The focus of the microprocess is on achieving an intermediate product baseline with sufficient functionality as economically and rapidly as practical
- All project processes consist of productive activities and overhead activities
- Productive activities result in tangible progress toward the end product. For software efforts, these activities include prototyping, modeling, coding, debugging and user documentation.
- Overhead activities that have an intangible impact on the end product are required in plan preparation, documentation, progress monitoring, testing, integration etc...
- The objective of process improvement is to maximize the allocation of resources to productive activities and minimize the impact of overhead activities on resources such as personal, computers, and schedule.
- Better process can have even greater effect in reducing the time, it takes team to achieve the product vision with the required quality

Schedule improvement has at least three dimensions:

1. We could take N-step process and improve the efficiency of each step.
2. We could take N-step process and eliminate some steps so that it is now only M-step process.
3. We could take an N-step process and use more concurrency in the activities being performed or the resources being applied.

TABLE 3-4. *Three levels of process and their attributes*

ATTRIBUTES	METAPROCESS	MACROPROCESS	MICROPROCESS
Subject	Line of business	Project	Iteration
Objectives	Line-of-business profitability Competitiveness	Project profitability Risk management Project budget, schedule, quality	Resource management Risk resolution Milestone budget, schedule, quality
Audience	Acquisition authorities, customers Organizational management	Software project managers Software engineers	Subproject managers Software engineers
Metrics	Project predictability Revenue, market share	On budget, on schedule Major milestone success Project scrap and rework	On budget, on schedule Major milestone progress Release/iteration scrap and rework
Concerns	Bureaucracy vs. standardization	Quality vs. financial performance	Content vs. schedule
Time scales	6 to 12 months	1 to many years	1 to 6 months

3) Explain about improving team effectiveness?

Ans: COCOMO model suggests that the combined effects of personnel skill and experience can have an impact on productivity.

- Balance and coverage are two most important features of excellent teams.
- Teamwork is much more important than the sum of the individuals.
- A project manager need to select the skilled people and place them in respective positions.
- **Some maxims of team management include the following:**
 - ✓ A well-managed project can succeed with a nominal engineering team.
 - ✓ Mismanaged project will almost never succeed, even with an expert team of engineers.
 - ✓ A well-architected system can be built by a nominal team of software builders.
 - ✓ A poorly architected system will flounder even with an expert team of builders.

Boehm offered the following five staffing principles:

- 1) **The principle of top talent:** Use better and fewer people.
- 2) **The principle of job matching:** Fit the tasks to the skills and motivation of the people available.
- 3) **The principle of career progression:** An organization does best in the long run by helping its people to self-actualize.
- 4) **The principle of team balance:** Select people who will complement and synchronize with one another.
- 5) **The principle of phase-out:** Keeping a misfit on the team doesn't benefit anyone.

In general staffing is achieved by these common methods:

1. If people are already available with required skill set, just take them

2. If people are already available but do not have the required skills, re-train them
 3. If people are not available, recruit trained people
 4. If you are not able to recruit skilled people, recruit and train people.
- Software project managers need many leadership qualities in order to enhance team effectiveness.
 - **Some of the attributes of successful s/w project manager are:**
 1. **Hiring skills:** Few decisions are as important as hiring decisions. Placing the right person in the right job seems obvious but is surprisingly hard to achieve.
 2. **Customer-interface skill:** Avoiding adversarial relationships among stake-holders is a prerequisite for success.
 3. **Decision-making skill:** a software project manager was able to take right decision in right time.
 4. **Team-building skill:** Teamwork requires that a manager establish trust, motivate progress, transition average people into top performers, eliminate misfits.
 5. **Selling skill:** Successful project managers must sell all stakeholders (including themselves) on decisions and priorities, sell candidates on job positions, and sell achievements against objectives.

4) Explain about improving Automation through software environment?

Ans:

- Some of the configuration management environments which provide the foundation for executing and implementing the process are Planning tools, Quality assurance and analysis tools, Test tools, and User interfaces provide crucial automation support for evolving the software engineering artifacts.
- A highly integrated environment is necessary to both facilitate and to enforce management control of the process.
- An environment that provides semantic integration (process of interrelating information from diverse sources) and process automation (automate the cost that contain cost) can improve productivity, improve software quality, and accelerate the adoption of modern techniques.

Round-trip engineering is a term used to describe the key capability of environments that support iterative development.

- It also describes the environment support needed to change an artifact freely and other artifacts changed automatically.

Forward engineering is the automation of one engineering artifact from another, more abstract representation.

- Example : compilers and linkers have provided automated transition of source code into executable code.

Reverse engineering is the generation of modification of more abstract representation from an existing artifact. Ex: creating visual design model from a source code.

- Example: Creating design model from a source code representation.

5) Explain how to achieve required quality?

Ans:

Key elements that improve overall software quality include the following:

- Focusing on powerful requirements and critical use case early in the life
- Focusing on requirements completeness and traceability late in the life cycle
- Focusing throughout the life cycle on a balance between requirements evolution, design evolution, and plan evolution
- Using metrics and indicators to measure the progress and quality of an architecture as it evolves from high-level prototype into a fully biddable product
- Providing integrated life-cycle environments that support early and continuous configuration control, change management, rigorous design methods, document automation, and regression test automation
- Using visual modeling and higher level languages that support architectural control, abstraction, reliable programming, reuse, and self-documentation
- Early and continuous close look into performance issues through demonstration-based evaluations

In order to evaluate the performance the following sequence of events are necessary,

- 1) Project inception
- 2) Initial design review
- 3) Mid-life-cycle design review
- 4) Integration and test

TABLE 3-5. *General quality improvements with a modern process*

QUALITY DRIVER	CONVENTIONAL PROCESS	MODERN ITERATIVE PROCESSES
Requirements misunderstanding	Discovered late	Resolved early
Development risk	Unknown until late	Understood and resolved early
Commercial components	Mostly unavailable	Still a quality driver, but trade-offs must be resolved early in the life cycle
Change management	Late in the life cycle, chaotic and malignant	Early in the life cycle, straightforward and benign
Design errors	Discovered late	Resolved early
Automation	Mostly error-prone manual procedures	Mostly automated, error-free evolution of artifacts
Resource adequacy	Unpredictable	Predictable
Schedules	Overconstrained	Tunable to quality, performance, and technology
Target performance	Paper-based analysis or separate simulation	Executing prototypes, early performance feedback, quantitative understanding
Software process rigor	Document-based	Managed, measured, and tool-supported

6) What are the principles of conventional software engineering?

Ans: 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:** In order to improve the quality of the product we need to involving the customer, select the prototyping, simplifying design, conducting inspections, and hiring the best people.
- 3) **Give products to customers early:** No matter how hard you try to learn user’s 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:** Whenever a problem is raised most engineers provide a solution. Before we try to solve a problem, be sure to explore all the alternatives and don’t be blinded by the understandable solution.
- 5) **Evaluate design alternatives:** After the requirements are agreed upon, we must examine a variety of architectures and algorithms and choose the one which is not used earlier.
- 6) **Use an appropriate process model:** For every project, there are so many prototypes (process models). So select the best one that is exactly suitable to our project.
- 7) **Use different languages for different phases:** Our industry’s main aim is to provide simple solutions to complex problems. In order to accomplish this goal choose different languages for different modules/phases if required.
- 8) **Minimize intellectual distance:** We have to design the structure of a software is as close as possible to the real-world structure.
- 9) **Put techniques before tools:** An un disciplined software engineer with a tool becomes a dangerous, undisciplined software engineer.
- 10) **Get it right before you make it faster:** It is very easy 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 the code:** Examine the detailed design and code is a much better way to find the errors than testing.
- 12) **Good management** is more important than good technology
- 13) **People are the key to success:** Highly skilled people with appropriate experience, talent, and training are key. The right people with insufficient tools, languages, and process will succeed.
- 14) **Follow with care:** Everybody is doing something but 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 engineer do wrong?”. Similarly if the software fails, we ask the same. So the fact is in every engineering discipline, the best methods can be used to produce poor results and the most out of date methods to produce stylish design.

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 they 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 create 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 Tom 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 become more and more disorganized.

29. People and time are not interchangeable. Measuring a project solely by person-months makes little sense.

30) Expert excellence. Your employees will do much better if you have high expectations for them.

7) What are the principles of modern software management?

Ans:

1) Base the process on an architecture-first approach: (Central design element)

- Design and integration first, then production and test

2) Establish an iterative life-cycle process: (The risk management element)

- Risk control through ever-increasing function, performance, quality.

With today's sophisticated 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 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: (The technology element)

Moving from LOC mentally to component-based mentally is necessary to reduce the amount of human-generated source code and custom development. A component is a cohesive set of preexisting lines of code, either in source or executable format, with a defined interface and behavior.

4) Establish a change management environment: (The control element)

- Metrics, trends, process instrumentation

The dynamics of iterative development, include concurrent workflows by different teams working on shared artifacts, necessitates objectively controlled baseline.

5) Enhance change freedom through tools that support round-trip engineering: (The automation element)

- Complementary tools, integrated environment

Round-trip engineering is the environment support necessary to automate and synchronize engineering information in different formats. Change freedom is necessary in an iterative process.

6) Capture design artifacts in rigorous, model-based notation:

- A model-based approach supports the evolution of semantically rich graphical and textual design notations.

- Visual modeling with rigorous notations and formal machine-processable language provides more objective measures than the traditional approach of human review and inspection of ad hoc design representations in paper doc.

7) Instrument the process for objective quality control and progress assessment:

- Life-cycle assessment of the progress and quality of all intermediate product must be integrated into the process.

- The best assessment mechanisms are well-defined measures derived directly from the evolving engineering artifacts and integrated into all activities and teams.

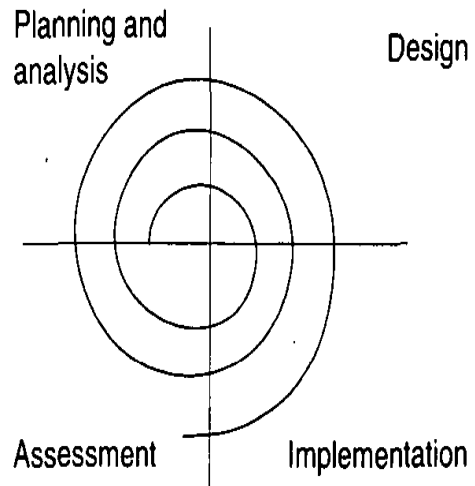
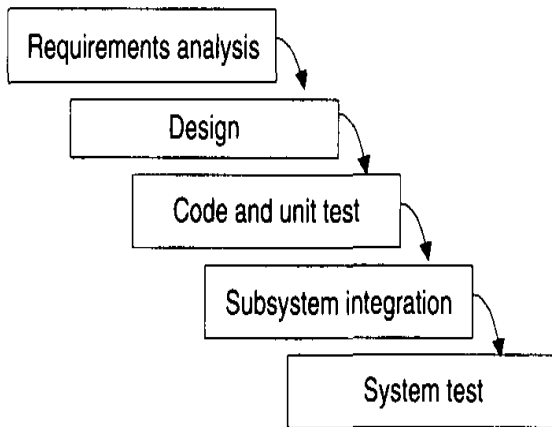
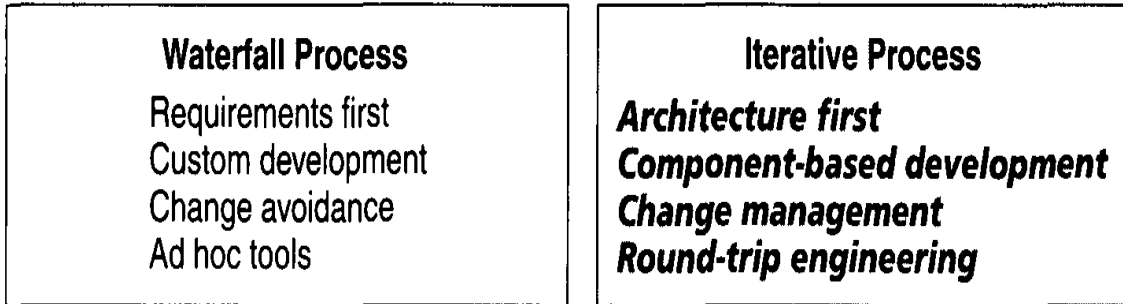
8) Use a demonstration-based approach to assess intermediate artifacts:

Transitioning from whether the artifact is an early prototype, a baseline architecture, or a beta capability into an executable demonstration of relevant provides more tangible understanding of the design tradeoffs, early integration and earlier elimination of architectural defects.

9) Plan intermediate releases in groups of usage scenarios with evolving levels of detail.

10) Establish a configurable process that economically scalable:

No single process is suitable for all software developments. The process must ensure that there is economy of scale and ROI.



UNIT III

1) What are the characteristics of an unsuccessful project?

Ans: Most unsuccessful projects exhibit one of the following characteristics:

- ✚ An overemphasis on research and development
 - Construction of engineering baselines is delayed.
- ✚ An overemphasis on product
 - Rush to judgment designs, premature work by overeager coders and continuous hacking.

2) Define engineering and production stages?

Ans:

- ✚ **The engineering stage:** Less predictable but smaller teams doing design and production activities. This stage is decomposed into two distinct phases *inception and elaboration*.
- ✚ **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*.

3) What are the stages in s/w development process?

Ans: 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

4) What are primary objective of inception phase?

Ans:

- Establishing the project's software scope and boundary conditions, including an 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 scenanos
- Estimating the cost and schedule for the entire project (including detailed estimates for the elaboration phase)
- Estimating potential risks (sources of unpredictability)

5) What are the essential activities of inception phase?

Ans:

- 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.

6) What are the primary objectives of elaboration phase?

Ans:

- Baselining the architecture as rapidly as practical (establishing a configuration-managed snapshot in which all changes are rationalized, tracked, and maintained)
- Baselining the vision
- Baselining a high-fidelity plan for the construction phase
- Demonstrating that the baseline architecture will support the vision at a reasonable cost in a reasonable time

7) What are the essential activities of elaboration phase?

Ans:

- Elaborating the vision.

- Elaborating the process and infrastructure.
- Elaborating the architecture and selecting components.

8) What are the primary objectives of construction phase?

Ans:

- 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

9) What are the primary objectives of transition phase

Ans:

- Achieving user self-supportability
- Achieving stakeholder concurrence that deployment baselines are complete and consistent with the evaluation criteria of the vision
- Achieving final product baselines as rapidly and cost-effectively as practical

10) What are the essential activities of construction phase?

Ans:

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

11) What are the essential activities of transition phase?

Ans:

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

12) What are artifact sets?

Ans:

- To make the development of a complete software system manageable, distinct collections of information are organized into artifact sets.
- Artifact* represents cohesive information that typically is developed and reviewed as a single entity.
- Life-cycle software artifacts are organized into five distinct sets that are roughly partitioned by the underlying language of the set: management (ad hoc textual formats), requirements (organized text and models of the problem space), design (models of the solution space), implementation (human-readable programming language and associated source files), and deployment (machine-process able languages and associated files).

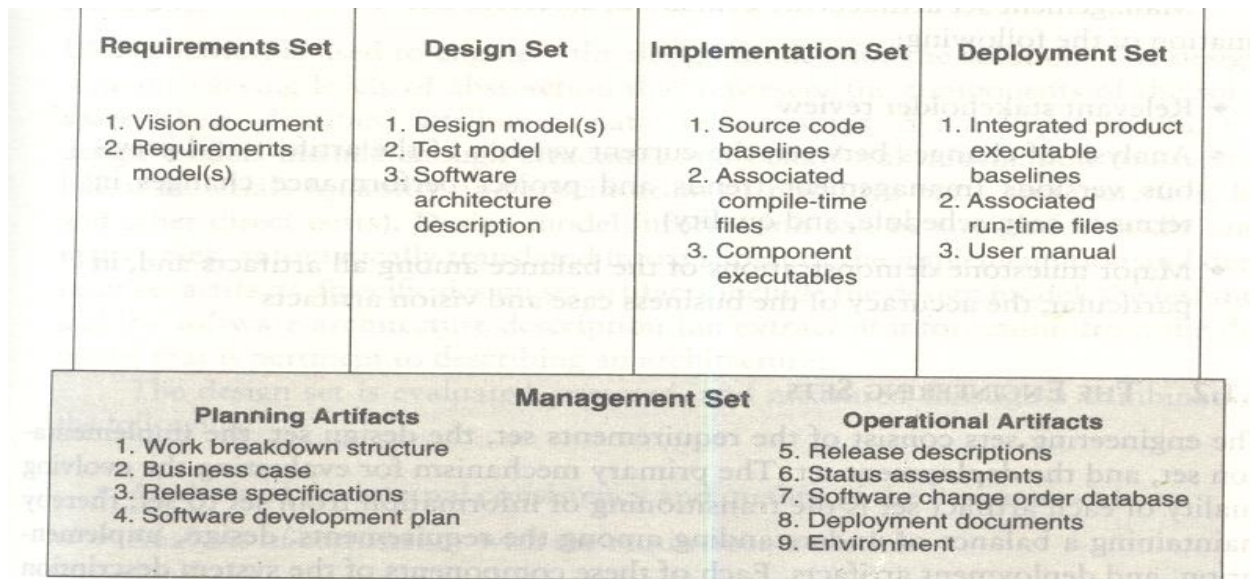


FIGURE 6-1. Overview of the artifact sets

13) Write about management sets?

Ans:

- 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

14) What is Vision Document?

Ans: 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.

15) What is Architecture Description?

Ans: 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.

16) What is Software User Manual?

Ans: 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.

17) What is Business Case?

Ans: 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.

18) What is Software Development Plan ?

Ans: 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 shelfware) and understanding and acceptance by managers and practitioners alike.

19) What is Work Breakdown Structure ?

Ans: 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.

20) What is Release Descriptions ?

Ans: Release description documents describe the results of each release, including performance against each of the evaluation criteria in the corresponding release specification. Release baselines should be accompanied by a release description document that describes the evaluation criteria for that configuration baseline and provides substantiation (through demonstration, testing, inspection, or analysis) that each criterion has been addressed in an acceptable manner. Figure 6-7 provides a default outline for a release description.

21) What is Status Assessments ?

Ans: 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

22) What is Environment ?

Ans: 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.

23) What is Deployment ?

Ans: 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.

24) What is Management Artifact Sequences ?

Ans: 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. Figure 6-8 identifies a typical sequence of artifacts across the life-cycle phases.

25) What is Software Change Order Database?

Ans: 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.

26) What is Release Specifications ?

Ans: 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.

LONG ANSWER QUESTIONS

1) Explain the stages of s/w development process?

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

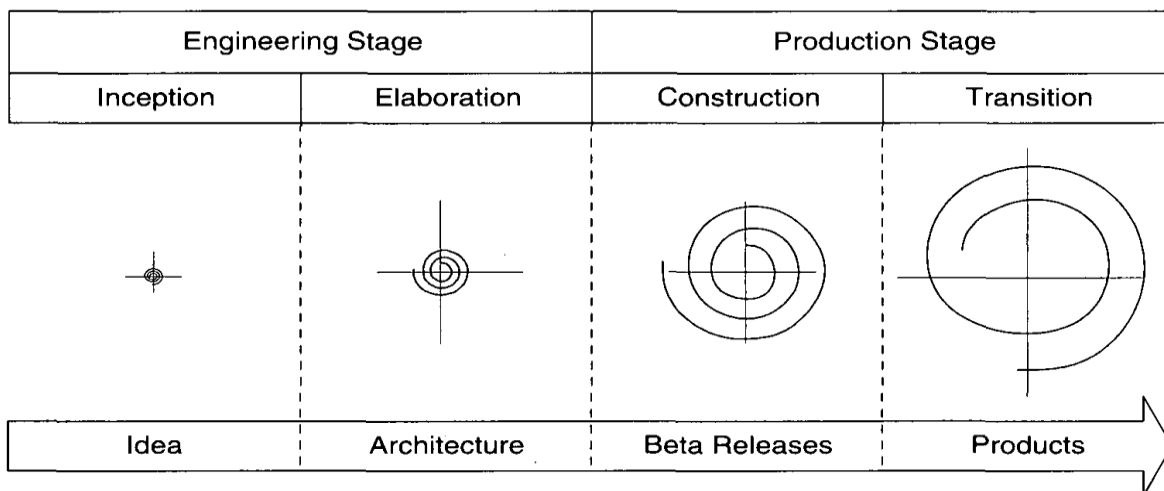
There are two stages in the software development process

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

TABLE 5-1. The two stages of the life cycle: engineering and production

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

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



In the above figure the size of the spiral corresponds to the inactivity of the project with respect to the breadth and depth of the artifacts that have been developed.

- This inertia manifests itself in maintaining artifact consistency, regression testing, documentation, quality analyses, and configuration control.
- Increased inertia may have little, or at least very straightforward, impact on changing any given discrete component or activity.
- However, the reaction time for accommodating major architectural changes, major requirements changes, major planning shifts, or major organizational perturbations clearly increases in subsequent phases.

2) Describe inception phase?

Ans: The overriding goal of the inception phase is to achieve concurrence among stakeholders

on the life-cycle objectives for the project.

PRIMARY OBJECTIVES

- Establishing the project's software scope and boundary conditions, including an 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 unpredictability)

ESSENTIAL ACTIVITIES

- Formulating the scope of the project. The information repository should be sufficient to define the problem space and derive the acceptance criteria for the end product.
- Synthesizing the architecture. An information repository is created that is sufficient to demonstrate the feasibility of at least one candidate architecture and an initial baseline of make/buy decisions so that the cost, schedule, and resource estimates can be derived.
- Planning and preparing a business case. Alternatives for risk management, staffing, iteration plans, and cost/schedule/profitability trade-offs are evaluated.

PRIMARY EVALUATION CRITERIA

- Do all stakeholders concur on the scope definition and cost and schedule estimates?
- Are requirements understood, as evidenced by the fidelity of the critical use cases?
- Are the cost and schedule estimates, priorities, risks, and development processes credible?
- Do the depth and breadth of an architecture prototype demonstrate the preceding criteria? (The primary value of prototyping candidate architecture is to provide a vehicle for understanding the scope and assessing the credibility of the development group in solving the particular technical problem.)
- Are actual resource expenditures versus planned expenditures acceptable

3). Describe Elaboration phase?

Ans: 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 can 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, & risk.

PRIMARY OBJECTIVES

- Baseline the architecture as rapidly as practical (establishing a configuration-managed snapshot in which all changes are rationalized, tracked, and maintained)
- Baseline the vision
- Baseline a high-fidelity plan for the construction phase
- Demonstrating that the baseline architecture will support the vision at a reasonable cost in a reasonable time

ESSENTIAL ACTIVITIES

- Elaborating the vision.
- Elaborating the process and infrastructure.
- Elaborating the architecture and selecting components.

PRIMARY EVALUATION CRITERIA

- Is the vision stable?
- Is the architecture stable?
- Does the executable demonstration show that the major risk elements have been addressed and credibly resolved?
- Is the construction phase plan of sufficient fidelity, and is it backed up with a credible basis of estimate?
- Do all stakeholders agree that the current vision can be met if the current plan is executed to develop the complete system in the context of the current architecture?
- Are actual resource expenditures versus planned expenditures acceptable?

4) Describe construction phase?

Ans:

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

- The construction phase represents a production process, in which emphasis is placed on managing resources and controlling operations to optimize costs, schedules, and quality.

PRIMARY OBJECTIVES

- Minimizing development costs by optimizing resources and avoiding unnecessary scrap and rework
- Achieving adequate quality as rapidly as practical
- Achieving useful versions (alpha, beta, and other test releases) as rapidly as practical

ESSENTIAL ACTIVITIES

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

PRIMARY EVALUATION CRITERIA

- Is this product baseline mature enough to be deployed in the user community? (Existing defects are not obstacles to achieving the purpose of the next release.)
- Is this product baseline stable enough to be deployed in the user community? (Pending changes are not obstacles to achieving the purpose of the next release.)
- Are the stakeholders ready for transition to the user community?
- Are actual resource expenditures versus planned expenditures acceptable?

5) Describe transition phase?

Ans:

- 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 users and maintainers
- The transition phase concludes when the deployment baseline has achieved the complete vision.

PRIMARY OBJECTIVES

- Achieving user self-supportability
- Achieving stakeholder concurrence that deployment baselines are complete and consistent with the evaluation criteria of the vision
- Achieving final product baselines as rapidly and cost-effectively as practical

ESSENTIAL ACTIVITIES

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

EVALUATION CRITERIA

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

6) Explain engineering sets?

Ans: The engineering sets consist of the requirements set, the design set, the implementation set, and the deployment set.

Requirements Set

Requirements artifacts are evaluated, assessed, and measured through a combination of the following:

- Analysis of consistency with the release specifications of the management set
- Analysis of consistency between the vision and the requirements models
- Mapping against the design, implementation, and deployment sets to evaluate the consistency and completeness and the semantic balance between information in the different sets
- Analysis of changes between the current version of requirements artifacts and previous versions (scrap, rework, and defect elimination trends)
- Subjective review of other dimensions of quality

Design Set

UML notation is used to engineer the design models for the solution. The design set contains varying levels of abstraction that represent the components of the solution space (their identities, attributes, static relationships, dynamic interactions). The design set is evaluated, assessed, and measured through a combination of the following:

- Analysis of the internal consistency and quality of the design model
- Analysis of consistency with the requirements models

- Translation into implementation and deployment sets and notations (for example, traceability, source code generation, compilation, linking) to evaluate the consistency and completeness and the semantic balance between information in the sets
- Analysis of changes between the current version of the design model and previous versions (scrap, rework, and defect elimination trends)
- Subjective review of other dimensions of quality

Implementation set

The implementation set includes source code (programming language notations) that represents the tangible implementations of components (their form, interface, and dependency relationships)

Implementation sets are human-readable formats that are evaluated, assessed, and measured through a combination of the following:

- Analysis of consistency with the design models
- Translation into deployment set notations (for example, compilation and linking) to evaluate the consistency and completeness among artifact sets
- Assessment of component source or executable files against relevant evaluation criteria through inspection, analysis, demonstration, or testing
- Execution of stand-alone component test cases that automatically compare expected results with actual results
- Analysis of changes between the current version of the implementation set and previous versions (scrap, rework, and defect elimination trends)
- Subjective review of other dimensions of quality

Deployment Set

The deployment set includes user deliverables and machine language notations, executable software, and the build scripts, installation scripts, and executable target specific data necessary to use the product in its target environment.

Deployment sets are evaluated, assessed, and measured through a combination of the following:

- Testing against the usage scenarios and quality attributes defined in the requirements set to evaluate the consistency and completeness and the semantic balance between information in the two sets
- Testing the partitioning, replication, and allocation strategies in mapping components of the implementation set to physical resources of the deployment system (platform type, number, network topology)
- Testing against the defined usage scenarios in the user manual such as installation, user-oriented dynamic reconfiguration, mainstream usage, and anomaly management
- Analysis of changes between the current version of the deployment set and previous versions (defect elimination trends, performance changes)

- Subjective review of other dimensions of quality

Each artifact set is the predominant development focus of one phase of the life cycle; the other sets take on check and balance roles. As illustrated in Figure 6-2, each phase has a predominant focus: Requirements are the focus of the inception phase; design, the elaboration phase; implementation, the construction phase; and deployment, the transition phase. The management artifacts also evolve, but at a fairly constant level across the life cycle.

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

1. Management: scheduling, workflow, defect tracking, change management, documentation, spreadsheet, resource management, and presentation tools
2. Requirements: requirements management tools
3. Design: visual modeling tools
4. Implementation: compiler/debugger tools, code analysis tools, test coverage analysis tools, and test management tools

Deployment: test coverage and test automation tools, network management tools, commercial components (operating systems, GUIs, RDBMS, networks, middleware), and installation tools.

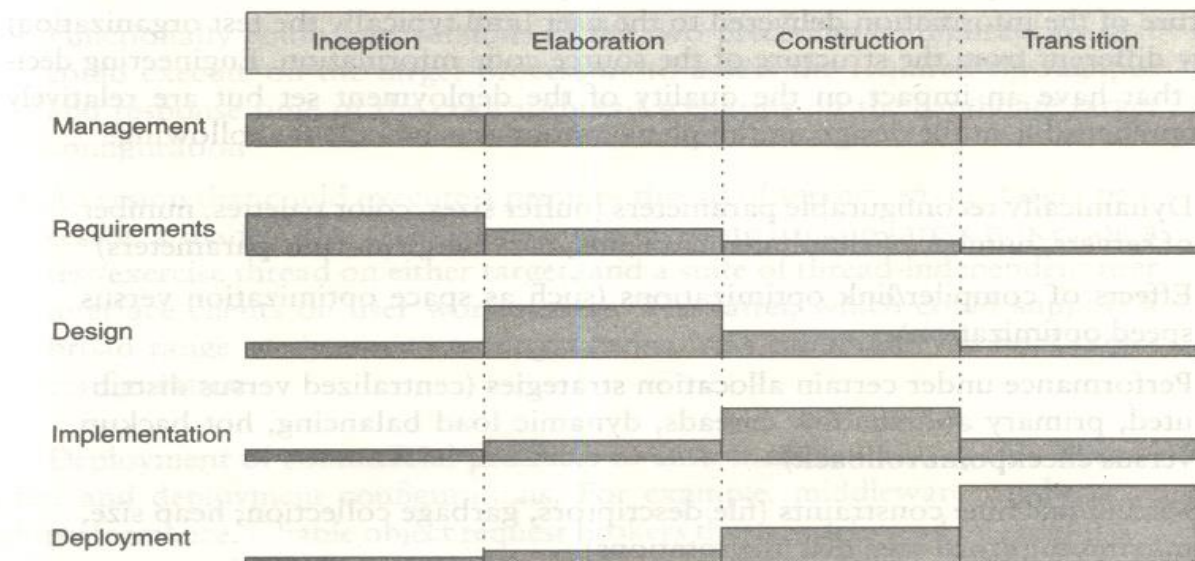


FIGURE 6-2. *Life-cycle focus on artifact sets*

Implementation Set versus Deployment Set

The separation of the implementation set (source code) from the deployment set (executable code) is important because there are very different concerns with each set. The structure of the information delivered to the user (and typically the test organization) is very different from the structure of the source code information. Engineering decisions that have an impact on the quality of the deployment set but are relatively incomprehensible in the design and implementation sets include the following:

- Dynamically reconfigurable parameters (buffer sizes, color palettes, number of servers,

number of simultaneous clients, data files, run-time parameters)

- Effects of compiler/link optimizations (such as space optimization versus speed optimization)
- Performance under certain allocation strategies (centralized versus distributed, primary and shadow threads, dynamic load balancing, hot backup versus checkpoint/rollback)
- Virtual machine constraints (file descriptors, garbage collection, heap size, maximum record size, disk file rotations)
- Process-level concurrency issues (deadlock and race conditions)

Platform-specific differences in performance or behavior

7) Write short notes on artifact evolution over the life cycle?

Ans:

- + Each state of development represents a certain amount of precision in the final system description.
- + Early in the life cycle, precision is low and the representation is generally high.
- + Eventually, the precision of representation is high and everything is specified in full detail. Each phase of development focuses on a particular artifact set.
- + At the end of each phase, the overall system state will have progressed on all sets, as illustrated in Figure

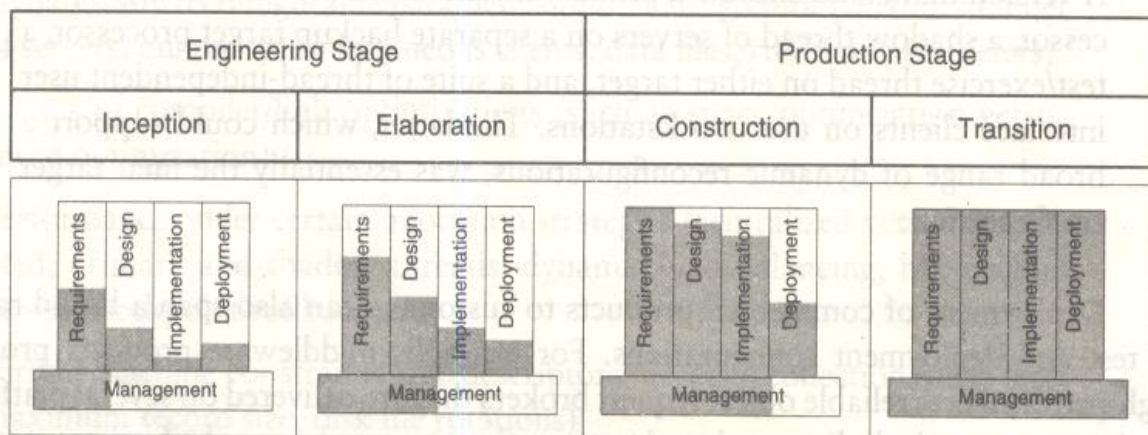


FIGURE 6-3. Life-cycle evolution of the artifact sets

- + The **inception** phase focuses mainly on critical requirements usually with a secondary focus on an initial deployment view.
- + During the **elaboration phase**, there is much greater depth in requirements, much more breadth in the design set, and further work on implementation and deployment issues.
- + The main focus of the **construction** phase is design and implementation.
- + The main focus of the **transition** phase is on achieving consistency and completeness of the deployment set in the context of the other sets.

8) Describe test artifacts?

Ans:

- The test artifacts must be developed concurrently with the product from inception through deployment. Thus, testing is a full-life-cycle activity, not a late life-cycle activity.
 - The test artifacts are communicated, engineered, and developed within the same artifact sets as the developed product.
 - The test artifacts are implemented in programmable and repeatable formats (as software programs).
 - The test artifacts are documented in the same way that the product is documented.
 - Developers of the test artifacts use the same tools, techniques, and training as the software engineers developing the product.
- Test artifact subsets are highly project-specific; the following example clarifies the relationship between test artifacts and the other artifact sets.
- Consider a project to perform seismic data processing for the purpose of oil exploration. This system has three fundamental subsystems:
 - (1) a sensor subsystem that captures raw seismic data in real time and delivers these data to
 - (2) a technical operations subsystem that converts raw data into an organized database and manages queries to this database from
 - (3) a display subsystem that allows workstation operators to examine seismic data in human-readable form.

Such a system would result in the following test artifacts:

- Management set. The release specifications and release descriptions capture the objectives, evaluation criteria, and results of an intermediate milestone. These artifacts are the test plans and test results negotiated among internal project teams. The software change orders capture test results (defects, testability changes, requirements ambiguities, enhancements) and the closure criteria associated with making a discrete change to a baseline.
- Requirements set. The system-level use cases capture the operational concept for the system and the acceptance test case descriptions, including the expected behavior of the system and its quality attributes. The entire requirement set is a test artifact because it is the basis of all assessment activities across the life cycle.
- Design set. A test model for no deliverable components needed to test the product baselines is captured in the design set. These components include such design set artifacts as a seismic event simulation for creating realistic sensor data; a "virtual operator" that can support unattended, after-hours test cases; specific instrumentation suites for early

demonstration of resource usage; transaction rates or response times; and use case test drivers and component stand-alone test drivers.

- Implementation set. Self-documenting source code representations for test components and test drivers provide the equivalent of test procedures and test scripts. These source files may also include human-readable data files representing certain statically defined data sets that are explicit test source files. Output files from test drivers provide the equivalent of test reports.
- Deployment set. Executable versions of test components, test drivers, and data files are provided.

9) Explain in detail about management artifacts?

Ans: 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. Figure 6-4 provides a default outline for a business case.

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 shelfware) and understanding and acceptance by managers and practitioners alike. Figure provides a default outline for a software development plan.

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

I.	Context (scope, objectives)
II.	Software development process
A.	Project primitives
1.	Life-cycle phases
2.	Artifacts
3.	Workflows
4.	Checkpoints
B.	Major milestone scope and content
C.	Process improvement procedures
III.	Software engineering environment
A.	Process automation (hardware and software resource configuration)
B.	Resource allocation procedures (sharing across organizations, security access)
IV.	Software change management
A.	Configuration control board plan and procedures
B.	Software change order definitions and procedures
C.	Configuration baseline definitions and procedures
V.	Software assessment
A.	Metrics collection and reporting procedures
B.	Risk management procedures (risk identification, tracking, and resolution)
C.	Status assessment plan
D.	Acceptance test plan
VI.	Standards and procedures
A.	Standards and procedures for technical artifacts
VII.	Evolutionary appendixes
A.	Minor milestone scope and content
B.	Human resources (organization, staffing plan, training plan)

FIGURE 6-5. Typical software development plan outline

Work Breakdown Structure

Work breakdown structure (WBS) is the vehicle for budgeting and collecting costs. To monitor and control a project's financial performance, the software project manager must have insight into project costs and how they are expended. The structure of cost accountability is a serious project planning constraint.

Software Change Order Database

Managing change is one of the fundamental primitives of an iterative development process. With greater change freedom, a project can iterate more productively. This flexibility increases the content, quality, and number of iterations that a project can achieve within a given schedule. Change freedom has been achieved in practice through automation, and today's iterative development environments carry the burden of change management. Organizational processes that depend on manual change management techniques have encountered major inefficiencies.

Release Specifications

The scope, plan, and objective evaluation criteria for each baseline release are derived from the vision statement as well as many other sources (make/buy analyses, risk management concerns, architectural considerations, shots in the dark, implementation constraints, quality thresholds). These artifacts are intended to evolve along with the process, achieving greater fidelity as the life cycle progresses and requirements understanding matures. Figure 6-6 provides a default outline for a release specification

I.	Iteration content
II.	Measurable objectives
	A. Evaluation criteria
	B. Followthrough approach
III.	Demonstration plan
	A. Schedule of activities
	B. Team responsibilities
IV.	Operational scenarios (use cases demonstrated)
	A. Demonstration procedures
	B. Traceability to vision and business case

FIGURE 6-6. *Typical release specification outline*

Release Descriptions

Release description documents describe the results of each release, including performance against each of the evaluation criteria in the corresponding release specification. Release baselines should be accompanied by a release description document that describes the evaluation criteria for that configuration baseline and provides substantiation (through demonstration, testing, inspection, or analysis) that each criterion has been addressed in an acceptable manner. Figure 6-7 provides a default outline for a release description.

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

I.	Context
	A. Release baseline content
	B. Release metrics
II.	Release notes
	A. Release-specific constraints or limitations
III.	Assessment results
	A. Substantiation of passed evaluation criteria
	B. Follow-up plans for failed evaluation criteria
	C. Recommendations for next release
IV.	Outstanding issues
	A. Action items
	B. Post-mortem summary of lessons learned

FIGURE 6-7. *Typical release description outline*

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. Figure 6-8 identifies a typical sequence of artifacts across the life-cycle phases.

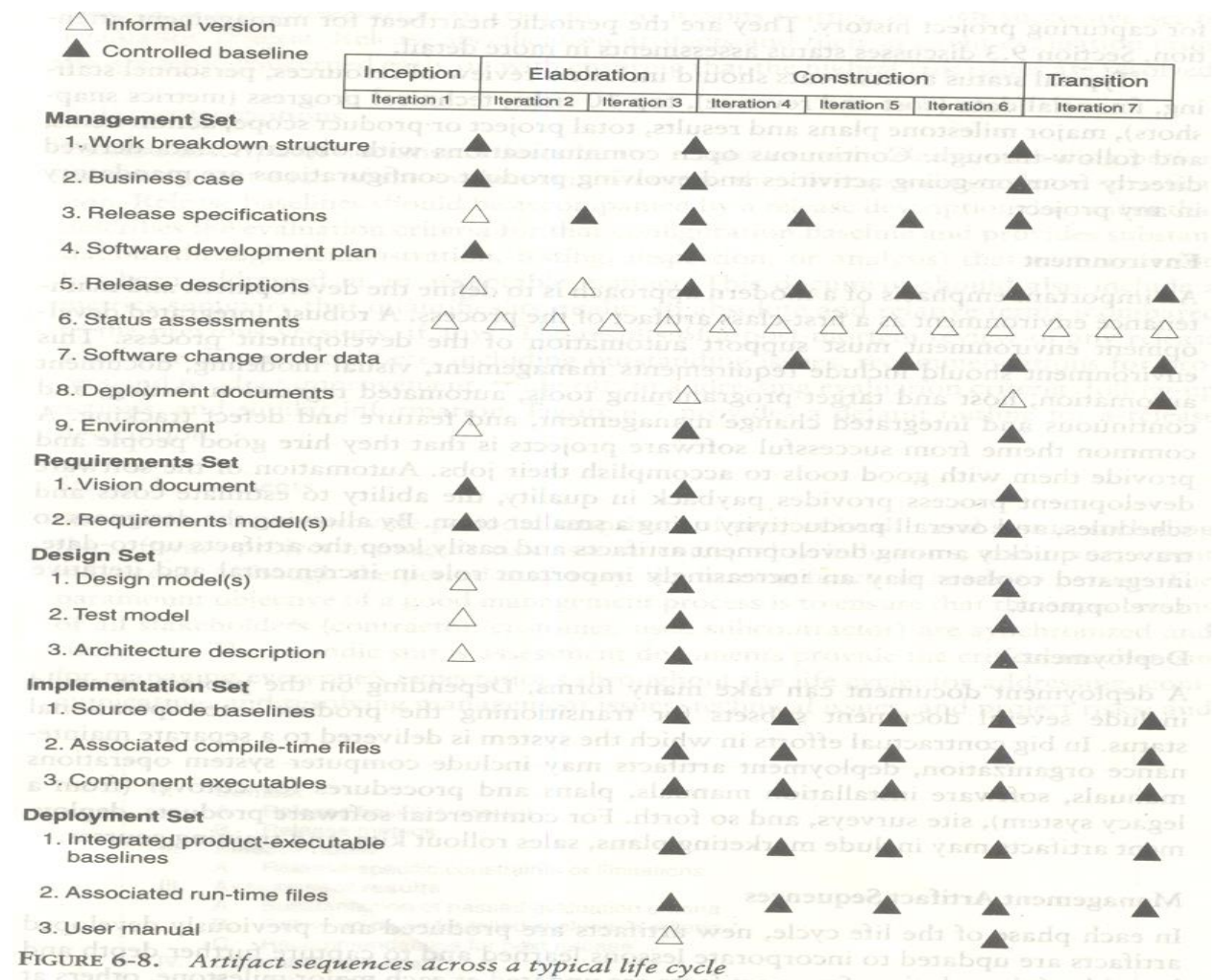


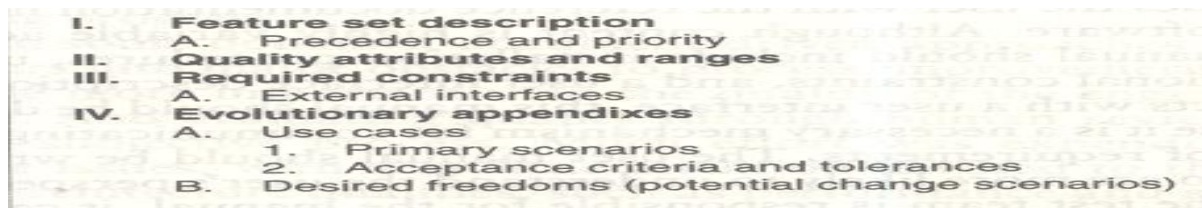
FIGURE 6-8. Artifact sequences across a typical life cycle

10) Explain engineering artifacts?

Ans: 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 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. Figure provides a default outline for a vision document.

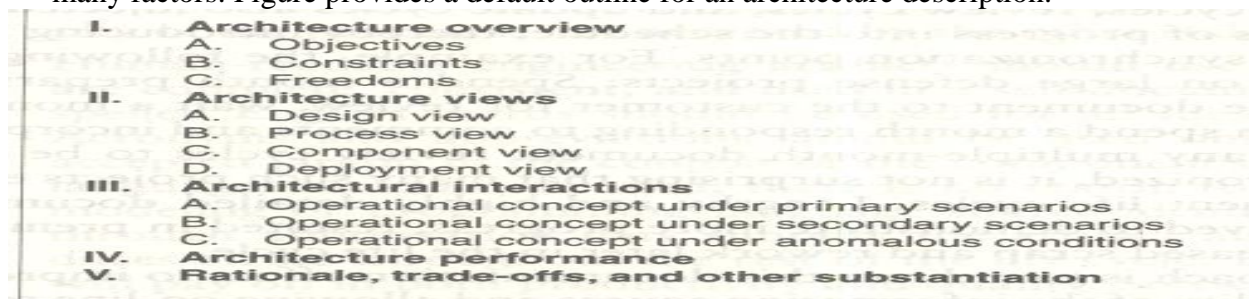


I.	Feature set description
A.	Precedence and priority
II.	Quality attributes and ranges
III.	Required constraints
A.	External interfaces
IV.	Evolutionary appendixes
A.	Use cases
1.	Primary scenarios
2.	Acceptance criteria and tolerances
B.	Desired freedoms (potential change scenarios)

FIGURE 6-9. Typical vision document outline

Architecture Description

- The architecture description provides an organized view of the software architecture under development.
- It is extracted largely from the design model and includes views of the design, implementation, and deployment sets sufficient to understand how the operational concept of the requirements set will be achieved.
- The breadth of the architecture description will vary from project to project depending on many factors. Figure provides a default outline for an architecture description.



I.	Architecture overview
A.	Objectives
B.	Constraints
C.	Freedoms
II.	Architecture views
A.	Design view
B.	Process view
C.	Component view
D.	Deployment view
III.	Architectural interactions
A.	Operational concept under primary scenarios
B.	Operational concept under secondary scenarios
C.	Operational concept under anomalous conditions
IV.	Architecture performance
V.	Rationale, trade-offs, and other substantiation

FIGURE 6-10. Typical architecture description outline

Software 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.

11) Write a short note on pragmatic artifacts?

Ans:

- Conventional document driven approach squandered incredible amounts of engineering time on developing, polishing, formatting, reviewing, updating and distributing.
- The most important reasons for the importance of the document are
 - There are no engineering methods or languages for requirements specification or design. Only the paper document with adhoc texts and graphical representation is the only format.
 - Conventional languages of implementation and deployment were unstructured. So, to present the details of s/w structure and behavior to other interested reviewers, a more human readable format was need.
- A more effective approach is to redirect this documentation effort for improving and understanding the information source and allowing on-line review of the native information source by using smart browsing and navigation tools.
- This approach can eliminate a huge unproductive source of scrap and rework in the process.
- Some of the issues are
 - **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.

12) Explain in detail about model based architecture?

Ans: *Software architecture* is the central design problem of a complex software system in the same way an *architecture* is the software system design.

- The ultimate goal of the engineering stage is to converge on a stable architecture baseline.
- Architecture is not a paper document. It is a collection of information across all the engineering sets.
- Architectures are described by extracting the essential information from the design models.
- A *model* is a relatively independent abstraction of a system.
- A *view* is a subset of a model that abstracts a specific, relevant perspective.

ARCHITECTURE : A MANAGEMENT PERSPECTIVE

- The most critical and technical product of a software project is its architecture
- If a software development team is to be successful, the interproject communications, as captured in software architecture, must be accurate and precise.

From the management point of view, three different aspects of architecture

1. An *architecture* (the intangible design concept) is the design of software system it includes all engineering necessary to specify a complete bill of materials. Significant make or buy decisions are resolved, and all custom components are elaborated so that individual component costs and construction/assembly costs can be determined with confidence.

2. An *architecture baseline* (the tangible artifacts) is a slice of information across the engineering artifact sets sufficient to satisfy all stakeholders that the vision (function and quality) can be achieved within the parameters of the business case (cost, profit, time, technology, people).

3. An *architectural description* is an organized subset of information extracted from the design set model's. It explains how the intangible concept is realized in the tangible artifacts.

The number of views and level of detail in each view can vary widely. For example the architecture of the software architecture of a small development tool.

There is a close relationship between software architecture and the modern software development process because of the following reasons:

1. A stable software architecture is nothing but a project milestone where critical make/buy decisions should have been resolved. The life-cycle represents a transition from the engineering stage of a project to the production stage.
2. Architecture representation provide a basis for balancing the trade-offs between the problem space (requirements and constraints) and the solution space (the operational product).
3. The architecture and process encapsulate many of the important communications among individuals, teams, organizations, and stakeholders.
4. Poor architecture and immature process are often given as reasons for project failure.
5. In order to proper planning, a mature process, understanding the primary requirements and demonstrable architecture are important fundamentals.
6. Architecture development and process definition are the intellectual steps that map the problem to a solution without violating the constraints; they require human innovation and cannot be automated.

ARCHITECTURE: A TECHNICAL PERSPECTIVE

- Software architecture include the structure of software systems, their behavior, and the patterns that guide these elements, their collaborations, and their composition.
- An architecture framework is defined in terms of views is the abstraction of the UML models in the design set. Where as architecture view is an abstraction of the design model, include full breadth and depth of information.

Most real-world systems require four types of views:

- 1) Design: describes architecturally significant structures and functions of the design model.
- 2) Process: describes concurrency and control thread relationships among the design, component, and deployment views.
- 3) Component: describes the structure of the implementation set.
- 4) Deployment: describes the structure of the deployment set.

The design set include all UML design models describing the solution space.

- The design, process, and use case models provide for visualization of the logical and behavioral aspect of the design.
- The component model provides for visualization of the implementation set.

The deployment model provides for visualization of the deployment set.

1. The *use case view* describes how the system's critical use cases are realized by elements of the design model. It is modeled statistically by using use case diagrams, and dynamically by using any of the UML behavioral diagrams.
2. The *design view* describes the architecturally significant elements of the design model. It is modeled statistically by using class and object diagrams, and dynamically using any of the UML behavioral diagrams.
3. The *process view* addresses the run-time collaboration issues involved in executing the architecture on a distributed deployment model, including logical software topology, inter process communication, and state mgmt. it is modeled statistically using deployment diagrams, and dynamically using any of the UML behavioral diagrams.

4. The *component view* describes the architecturally significant elements of the implementation set. It is modeled statistically using component diagrams, and dynamically using any of the UML behavioral diagrams.

The *deployment view* addresses the executable realization of the system, including the allocation of logical processes in the distributed view to physical resources of the deployment network. It is modeled statistically using deployment diagrams, and dynamically using any of UML behavioral diagrams.

Architecture descriptions take on different forms and styles in different organizations and domains. At any given time, an architecture requires a subset of artifacts in engineering set.

- An architecture baseline is defined as a balanced subset of information across all sets, whereas an architecture description is completely encapsulated within the design set.

Generally architecture base line include:

- 1) Requirements
- 2) Design
- 3) Implementation
- 4) Deployment

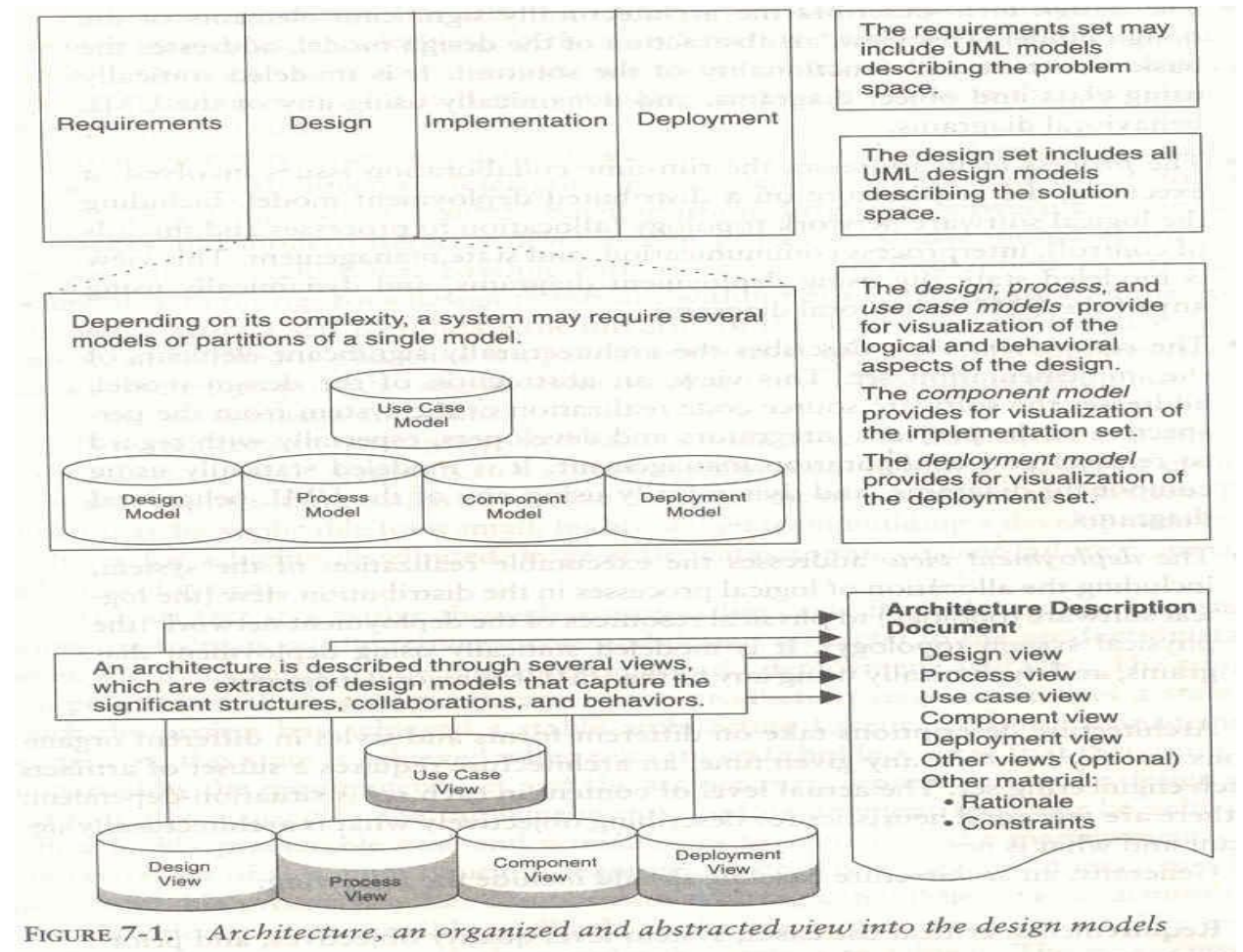


FIGURE 7-1. Architecture, an organized and abstracted view into the design models